Өспүрүмдөр үчүн Python

Супер Баатырдай программалоону үйрөнүңүз!

Жеймс Р. Пейн

APRESS®

Python for Teenagers: Learn to Program like a Superhero!

James R. Payne
Deerfield Beach, FL, USA

Өспүрүмдөр үчүн Python: Супер Баатырдай программалоону үйрөнүңүз!/ Котор.Ш.Каримова. – Б.: 2021 – 317 б.

ISBN-13 (pbk): 978-1-4842-4549-1
ISBN-13 (electronic): 978-1-4842-4550-7
https://doi.org/10.1007/978-1-4842-4550-7

Бул эмгек автордук укук менен корголгон. Материал толугу менен же жарым-жартылай которууга, кайра басып чыгарууга, иллюстрацияларды кайрадан колдонууга, айтууга, эфирге чыгарууга, микрофильмде же башка ар кандай физикалык каражаттарга көчүрүп алууга, ошондой эле маалыматты берүү же сактоо жана электрондук адаптация, компьютердик программа же иштелип чыккан окшош методиканы колдонуу сыяктуу бардык укуктар басып чыгаруучу тарабынан сакталат.

Бул китепте аталыштар, логотиптер жана соода маркасынын сүрөттөрү болушу мүмкүн. Товардык белгинин аталышы, логотиби же сүрөтү кезиккен сайын товардык белгини колдонуунун ордуна, биз товардык белгинин укуктарын бузбоо максатында, аталыштарды, логотиптерди жана сүрөттөрдү редакциялык максаттарда гана жана белгинин беделине зыян келтирбестен колдонобуз.

Бул басылмада соода аталыштарын, соода маркаларын, тейлөө белгилерин жана ушул сыяктуу терминдерди колдонуу, эгерде көрсөтүлбөсө дагы, алардын менчик укугунун предмети экендиги жөнүндө ой-пикирин билдирүү катары кабыл алынбашы керек.

Бул китептеги кеңештер жана маалыматтар жарыяланган күнгө карата туура жана так деп эсептелгени менен, авторлор да, редакторлор дагы, басмаканасы дагы кетирилген каталар же кемчиликтер үчүн эч кандай мыйзамдуу жоопкерчиликти ала алышпайт. Басып чыгаруучу ушул документте камтылган материалдарга карата эч кандай кепилдик бербейт.

Апресс Медиа ЖЧКсынын директору: Велмоед Спар
Мазмун боюнча жооптуу редактор: Тодд Грин
Өнүктүрүү редактору: Жеймс Маркхам
Координатор – редактор:Жилл Бальзано

Китептин мукабасы eStudioCalamar тарабынан жасалган

Китептин мукаба сүрөтү Freepik (www.freepik.com) булагынан алынган

Китеп Springer Science+Business Media (New York, 233 Spring Street, 6th Floor, New York, NY 10013) тарабынан сатыкка чыгарылган. Тел: 1-800-SPRINGER, факс: (201) 348-4505, e-mail: orders-ny@springersbm.com, сайт:www.springeronline.com аркылуу буюртмалар кабыл алынат. Apress Medica, LLC Калифорнияда Springer Science + Business Media Finance Inc (SSBM Finance Inc) тарабынан негизделген. SSBM Finance Делавэрде жайгашкан.

Бул китепте жазуучу шилтеме берген ар кандай булак же башка кошумча материалдар окурмандарга www.apress.com/978-1-4842-4549-1 дареги аркылуу GitHub эсебинде жеткиликтүү. Кененирээк маалымат алуу үчүн http://www.apress.com/source-code дарегине кириңиз.

Copyright © 2019 by James R. Payne

Бардык укуктар корголгон.

Кыргызча басылмасы © 2021 «Борбор Азия аймактык институту» коомдук бирикмеси

Долбоор «Борбор Азия аймактык институту» коомдук бирикмеси тарабынан аткарылды.
Басылма Америка Кошмо Штаттарынын элчилиги тарабынан каржыланган.
Акысыз таратылат.
Бул китеп кесипкөй котормочулар тарабынан которулган жана текшерилген.
Каталар кокусунан кетирилиши мүмкүн.

Ар дайым ишенгендиги жана менин компьютерге кыйкырганымды байкамаксан болгондугу үчүн менин аялым Уитни Пейнге арнайм...

Ата-энем Ронни жана Шэрон Пейн, ошондой эле менин бир тууганым Ронни Пейнге арналат, алардын аттарында табышмактуу түрдө "Рон" бар, аларга мен Бэтман болгум келгендигин айтсам дагы, ар дайым жашоодо каалаган адам боло аласын деп айтышкан.

Көп жылдар мурун мага күлкүлүү комикс каармандарынын ааламын түзүүгө жардам берген Доржан Уильямска ыраазымын. Чоң-кичине көйгөйлөрдү чечүүгө, анын ичинде өзүмдүн короомдогу ажыдаарды өлтүрүп, жумушумду бүтүрүүгө көңүл бурушум үчүн жардам бергендиги үчүн Эрик Миллерге ыраазычылык. Николас Рини мени программалоо жана комикс китептери менен тааныштырганына ыракмат, ансыз бул китеп болбойт эле. Нэнси Пакард менен Уэнди Уайттын китепке батпай турган көлөмдүү сөздөрдү колдонуп, идея беришкенине ыраазымын.

Эски Дев Шед командасынын мүчөлөрүнө: Дженнифер Руггиериге ырахмат, Чарльз Фагундес жана Кит Ли праграммалоого жардамы үчүн жана (көбүнчө) менин чөйчөгүмдүн ашып атканын эске салгандыгыңар үчүн ыраазымын. Хосе Эскаланте, ушул жерде сизге ыраазычылыгымды билдирем, анткени Джон Синаны жалгыз сиз гана көрө алгансыз. Энрике Стоун ... сен эмне кылганыңды билесиң.

Мага жардам берген Софи "Бульдог" Пейнге өзгөчө ыраазымын, бул китепте сүрөтүңдү колдонгонума жана ашканада ар дайым жакшы жардамчы болгонуңа ырахмат.

Мен көз ачып жумганча көп нерсени бүтүрүүгө жардам берген гений Таноско ыраазычылыгымды билдире албасам, капа болмокмун.

Ал эми Ричард С. мага "аларды тебелеп-тепсөөгө" жардам берди. Акырында, мага дем берген бир нече жазуучуларга ырахмат: А. Ли Мартинес, Нил Гайман, Фрэнк Миллер, Алан Мур, Джим Старлин жана Стивен Кинг - силер тезирээк жаза албайсыңарбы?

Мазмуну

Автор жөнүндө

Джеймс Р.Пейн 10 жашында эле программалоо менен таанышкан. Ал Lemonade Stand сыяктуу текстке негизделген оюндарды ойноп жатып артыкчылыкка ээ болуу үчүн хакерликти баштап, көп өтпөй Dungeons & Dragons стилинде өзүнүн текстке негизделген роль ойноочу оюндарын түзүп, сүйүктүү комикс китептеринен шыктанган. Алгачкы күндөрдүн кызыгуусу сакталып, ал өзүнүн карьерасында программалоо дүйнөсүнө кайра кайрылып келет.

Пейн - программалоого, веб иштеп чыгууга жана интернет-маркетингге арналган 14 веб-сайттан жана онлайн басылмалар форумунан турган Developer Shed компаниясынын мурдагы башкы редактору жана менеджери. Ал дээрлик бардык тилдерди жана платформаларды камтыган миңден ашуун программалоо жана маркетинг макалаларын жазган. Анын биринчи китеби, Beginning Python (Wrox Press), 2010-жылы басылып чыккан.

Мындан сырткары, ал оюндардан баштап аэрокосмостук жана аэронавтикага чейинки ар кандай темаларды камтыган 2000ден ашуун макалаларын жарыялаган. Чоңдор үчүн жана жаштарга арналган фантазиялык китептерди жазган.

Пейн бул китепти келечек муундарды программалоого шыктандырат деген үмүт менен өнүгүүгө болгон сүйүүсүн улантуу үчүн жазууну чечти. Пейн жөнүндө www.jamesrpayne.com дареги аркылуу кененирээк таанышсаңыздар болот.

Техникалык серепчи жөнүндө

Андреа Гавана 16 жылдай Python программасын түзүп, 1990-жылдардын аягынан бери башка программалоо тилдери менен да иштеп келет. Ал химиялык инженерия тармагында магистр даражасына ээ. Андреа - учурда Даниянын Копенгаген шаарындагы Total компаниясынын өнүгүү боюнча инженери.

Андреа жумушта жана көңүл ачуу үчүн код жазганды жакшы көрөт жана Python программалоо тилине негизделген бир нече ачык булактуу долбоорлорго катышкан. Анын сүйүктүү хоббилеринин бири - Python программалоосу, бирок ал велосипед тебүүнү, сууда сүзүүнү, үй-бүлөсү жана достору менен жайлуу кечки тамакты жакшы көрөт.

Бул - анын техникалык сынчы катары үчүнчү китеби.

Ыраазычылык

Менден китеп жазууну суранган, менин идеяларымды уккан Тодд Гринсиз бул китеп жаралмак эмес.

Джилл Бальзано, координациялык редактор, укмуштай түйшүктүү мезгилде жумушумду алдыга жылдырууда баа жеткис салым кошкон жана ансыз бул эмгек эч качан ишке ашмак эмес.

Джеймс Маркхам жана Андреа Гавана менин каталарымдын бардыгын таап, ушул карыган жашымда дагы үйрөнө турган көп нерсем бар экендигин далилдешти. Эски ит жаңы амалдарды үйрөнө алат деп ким ойлоптур.

Apress басмасынын жалпы редакциялык жамаатына ыраазычылык билдирем, алар менен иштешүү мага жагымдуу болду жана менин жакшы көргөн жумушумду жасоого, күлкүлүү комикс каармандарды жаратууга жардам беришти.

КИРИШҮҮ

Бул китеп кимдерге арналган?

Бул китеп Python программалоо тилинде алгачкы коддорду жазуу менен түрдүү деңгээлдеги программаларды түзүүнү каалаган өспүрүмдөр үчүн арналып жазылды. Ошондой эле, китеп техникалык жактан 13 жаштан 18 жашка чейинки өспүрүмдөргө арналат дегенибиз менен, чындыгында, китептин мазмунунда камтылган материалдар ар кандай курактагы адамдар үчүн да пайдалуу (Мисалы, мага окшоп деп айтсам да болот!). Ынтызар адам түрдүү маселелердин алгоритмдик чыгарылыштарын (а) Python тилинде программалоону үйрөнгүсү келсе, (б) алгачкы коддорду жазуу менен программаны кантип түзүү керек же (в) Python программалоо тилинин мүмкүнчүлүктөрүн учурдагы өзүнүн билим топтомуна кошууну кааласа, өздөштүрүп алып кетиши мүмкүн.

Баарынан маанилүүсү - эгер сиз бул китепти колуңузга кармасаңыз, демек, сиз эр жүрөк авантюристсиз. Анда бул китеп сизге арналып жазылган экен деп билиңиз. Келечек сиздей жаш баатырлардыкы экенин жакшы билет чыгарсыз. Код жазуу өнөрүн үйрөнүүгө дилгир болуп, дүйнөнү ыплас хакерлерден, шектүү программаланган тиркемелерден жана жасалма интеллектуалдык роботтордун зыянынан коргойсуз!

Ошентип, сиз алтынчы класста же колледжде окусаңыз да, бул китеп сизге чоң кубат, алгачкы түрткү бере алат. Албетте, бул китептеги окуу материалдарын үйрөнүп бүткөндөн кийин сиз тоону томкоруп же башыңызга оор жүктү көтөрө албайсыз. Бирок, сиз компьютерлердеги программалоо тилинде сүйлөй билип, мыкты программаларды түзө аласыз.

Мындан артык алгачкы жардам, жөндөмгө шыктандыруу бар болду бекен?

Бул китептен эмнелерди үйрөнүүгө болот?

Китептин 1-бөлүмдө программалоо жана Python программалоо тили тууралуу жалпы маалымат берилет. Андан кийин программаны компьютердин операциялык системасына (аракеттер системасына) орнотуу көрсөтүлөт. Бул сизге Python программалоо тили аркылуу өзүңүздүн эмгектериңизди жаратууга жана кодуңузду текшерүүгө мүмкүнчүлүк түзөт.

2-бөлүмдө биз математикалык амалдар менен аракеттенген функцияларды талкуулайбыз (бөлүү, кошуу, кемитүү жана көбөйтүү сыяктуу амалдар) жана Python колдонгон ар кандай *маалыматтардын түрлөрү* жөнүндө билебиз. Биз ошондой эле кызыктуу "Супер Баатыр генератору - 3000" колдонмосунун пайдубалын кура баштайбыз.

3-бөлүм тексттер менен кантип иштөө керектигин үйрөтөт. Python сунуш кылган сактоонун ар кандай түрлөрүн карап чыгабыз. Биз жалпы саптык функцияларды карап чыгып, биздин "Супер Баатыр генератору - 3000" тиркемесинин дагы бир бөлүмүн курабыз.

Кээде программа колдонуучунун сын-пикирине же башка таасирлерге жараша белгилүү бир чараларды көрүшү керек болот. Бул *чечим кабыл алуу* деп аталып, ал 4-бөлүмдүн темасы болуп эсептелет.

Программалоо логикасы жана циклдер - кайталоолор деп аталат, анда белгилүү бир шарттардын негизинде код кайталанып же “айлана” алат – бул тема 5-бөлүмдө камтылган.

6-бөлүм ушул убакка чейин үйрөнгөнүңүздү өркүндөтүү курсу болуп саналат. “Супер Баатыр генератору - 3000” толук нускасын куруп бүтүрүү үчүн алган бардык билимибизди жумшайбыз. Жыйынтыгында, сиз кайталангыс супер күчкө, ысымдарга жана согуш статистикасына ээ баатырларды өзүңүз каалагандай түзө аласыз!

7-бөлүмдө биз алдыңкы ыкмаларды үйрөнө баштайбыз. Чыныгы программист болуу үчүн эффективдүүлүктү үйрөнүп, түзгөн коддордо каталарды азайтыңыз. Бул жерде модулдар жана орнотулган функциялар иштеп башташат. Булардын эмне экендигин жана эмне үчүн сиздин жашооңузду ушунчалык жеңилдетерин ушул кызыктуу бөлүмдөн биле аласыз!

8-бөлүм андан да өркүндөтүлгөн темаларды камтыйт. Атап айтканда, *объектиге багытталган программалоонун (ООП)* негиздерин камтып, *объектилерди* жана *класстарды* карап, *полиморфизм* деп аталган түшүнүктү ачыктайбыз.

Китептин мазмунун кызыктуу берүү үчүн 9-бөлүмдө маалымат структураларынын ар кандай түрлөрү, анын ичинде кортеждер жана сөздүктөр каралат.

10-бөлүм бизди каталогдордун ичиндеги файлдарды түзүүнүн жана алар менен иштөөнүн жолдорун тааныштырат.

Мага өзгөчө жаккан бөлүмдөрүмдүн бири - 11-бөлүм. Ал менин жүрөгүмө жакын жана кымбат “Питон оюндар үчүн” темасын камтыйт. Биз видео оюндар

дүйнөсүндө саякаттап, видео оюн элементтери, анын ичинде үн, анимация жана башка нерселер менен кантип иштөөнү үйрөнөбүз!

Колдонуучулардын аракеттери менен өз ара байланышкан оюндарды түзүүнүн жолдорун үйрөнүү жана сүрөттөрдү оюндун ичинде кыймылдатуу чындыгында эле оюндарды жагымдуу кылат.

12-бөлүм оюн темасын улантат жана атайын оюн анимациясына басым жасайт.

13-бөлүм деп кабатыр болбоңуз. Бул учурда 13 саны - сиз үчүн бактылуу сан! Анткени биз азырынча талкуулай элек Python программалоо тилинин өз бөлүмдөрүнө туура келбеген жерлерине көчүп барабыз. Буга мүчүлүштүктөрдү *оңдоо* же бузулган кодду табуу кирет. Ошондой эле өркүндөтүлгөн модулдарды жана башка темаларды да карайбыз.

Акырында, биз 14-бөлүмдө бардыгын жыйынтыктап, Python программисти катары жумуш табуу, жалпы маектешүү суроолору, Python тилинин келечеги жана ийгиликке жетүүнүн жолдору сыяктуу ар кандай темаларды талкуулап, биздин сүйүктүү программалоо тили боюнча *көп берилүүчү суроолорго* жооп беребиз.

Ошентип, эми эмнени үйрөнө турганыбызды билдик. Келгиле, үстүбүзгө супер каармандын кийимдерин кийип, бийиктиктерге секирүүгө даярданалы! Башкача айтканда, билим алууга аттаналы!

Эмне үчүн мен программалоону баштадым?

Мен программалоону илгери эле, интернет же уюлдук телефон пайда боло электе, жапайы динозаврлар жерди аралап жүргөн мезгилде эле баштаганмын. Ал кезде компьютерлерде бүгүнкүдөй сүрөттөр жок болуп, бардыгы текстке негизделген эле. Алтургай, биздин оюндардын көпчүлүгү ошондой болчу. Эстесең, үрөйүң учат. Элестете да албайсың! Бизде анимация жана графика менен ойнолгон *анча–мынча* компьютердик оюндар бар болгон болсо, алардын өлчөмү аз болуп, азыркыдай синематик эмес эле.

Агам экөөбүзгө бир компьютерде иштөө бактысы буюрган эле. Ата-энем компьютердин эмнеге колдонуларын билишпесе да: "Бул - келечектин шайманы. Биздин балдарыбызды “келечектин Адамы” кылат..." - деп болжолдошсо керек.

Кайсы бир деңгээлде алар туура эле кылышкан экен. Эгер алар агам экөөбүзгө компьютер сатып беришпегенде, анда азыр менин эмне кылып жүргөнүм белгисиз болмок. Ким билет? Албетте, анда бул китепти силерге арнап жазмак да эмесмин. Силерге баатыр сымал программалоону да үйрөтүүгө жардам бере алмак эмесмин!

Бирок чоң салмактуу башаламан электроникадан жасалган “Apple II” мени өзүнө тартып, азгырууга кудурети жеткен жок. Анткен менен, менде укмуштуудай оюндары бар “Nintendo Entertainment System” (NES) бар болчу. Аны бүгүнкү күнгө чейин дагы деле ойноймун.

Мени компьютерге аралаштырган Николас деген досум болду. Ал компьютерлерди программалоону жакшы билген. Бир күнү ал мага артыкчылыкка ээ болуш үчүн бир нече текстке негизделген бизге жаккан оюндардын кодун кантип "бузуп" кирүүнүн жолдорун көрсөттү. Бул видео оюнда өзүңүздүн жеке кодуңузду түзүүгө окшош эле нерсе бар. Тактап айтканда, биз “Lemonade Stand” деп аталган оюнду ойнодук. Ал сиздин үйүңүздүн сыртында туруп, үй шартында жасалган лимонад сатуу менен бирдей эле, сиз эч качан чыныгы акча таппайсыз жана күнгө да күйбөйсүз.

Оюнду сиз эки доллар менен баштадыңыз. Бул каражат чыныгы киреше табууга араң эле жетмек. Бирок, оюнду баштаган кодду караарыбыз менен, бир аз эле сөздү өзгөртүп койсок, каалаган акчадан баштай аларыбызды түшүндүк. Көп өтпөй, мен дүйнөдөгү биринчи миллионер, “Lemonade Stand” магнаты болдум. Ошентип, мен компьютерге байландым да калдым.

Ушундан көп өтпөй биз өзүбүздүн видео оюндарды түзө аларыбызды түшүнүп, дал ошондой кылдык. Биздин сүйүктүү комикстердин жана “Dungeons & Dragons” оюндарынын негизинде түзүлгөн татаал ролдук оюндар боюнча (RPG), досторубузга бир катар суроолорду берип, андан кийин алардын жоопторунун негизинде шылдыңдаган программаларга чейин түздүк!

Мунун баары ошол мезгилде акылга сыйбаган нерсе сыяктуу сезилгени менен, артка кылчайып карасам, бул менин программалоого жана белгилүү деңгээлде жазууга болгон сүйүүмдүн пайдубалын түптөөгө жардам бергенин билдим (бирок мен ага чейин эле эрте жаза баштаганмын). Ошол жайкы программалоо кызыктуу болбосо, мен андан бери башыман өткөн сонун окуяларды, досторду, жумуштарды жана жазуу мүмкүнчүлүктөрүн эч качан көрмөк эмесмин.

Эң негизгиси, мен эч качан программалоого *чын дилимден кызыкмак* эмесмин.

Урматтуу окурман, сизге үйрөтөм деп үмүттөнүп жаткан нерсем – сиздерди программалоону жана мүмкүнчүлүктөрдү өмүр бою сүйүүгө багыттоо. Булардын бардыгы бир нерсеге негизделет. Ал - компьютердик программаларды жазуунун жана код жазуунун кызыктуу жана кубанычтуу учурлары.

Албетте, тиркемелерди программалоо сиздин башыңызды оорутушу мүмкүн. Түнкүсүн башыңызды клавиатурага ургулап, мониторго бир нече саат бою

кыйкырасыз. Анткени сиз бир эле жерде кашааны унутуп калганыңыз үчүн программаңыз иштебей калат.

Бирок, сиз өзүңүз же башка бир программист кетирген катаны тапсаңыз, анда бардык мезгилдердин эң улуу *программисти* болуп чыга келгениңизди түшүнгөндө, ошол жеңиштүү учурга эч нерсени салыштырбайсыз!

Программалоодо аткара ала турган жана аткара албай турган нерселер

Бул китепти окуп жатканда, сиз алдыга бир аз секирип же бир-эки көнүгүүнү өткөрүп жиберүүнү каалашыңыз мүмкүн. Жашоодо мааниси бар бардык нерселердей эле, биздин бул кеңешибиз программалоону үйрөнүүнүн мааниси да жогору экенин билдирет. Эгерде сиз бизден шектенип жаткан болсонуз, анда сиз өзүңүздү гана алдаган болосуз.

Сиздин ийгиликке жетишиңиз үчүн жардам берүүгө, бул китепти окуу жана программалоону үйрөнүү үчүн жасай турган жана жасабай турган нерселер төмөнкүлөр:

Китепти баштан-аяк *окуп* чыгыңыз. Кайсы бир бөлүмдү же көнүгүүнү өткөрүп жибергиңиз келсе, бул китеп жөн гана *программалоо* тилинин эмес, башка дагы көп нерселердин пайдубалын түптөөгө жарамдуу экендигин, сиз менен кошо ала турган программалоо принциптерин коддоо техникасы бар экендигин жана анын теориясы жана түшүнүгү *башка* тилдерде дагы колдонула тургандыгын унутпаңыз.

Кодду ушул китептен же башка булактан *көчүрбөңүз* (санарип көчүрмөсү бар деп эсептесеңиз). Андан көрө, код жазууга убакыт бөлүңүз, ошондо сиз программа жазганга кызыгып, балким, кайталоо жолу менен айрым коддорду эсиңизге сактап каласыз.

Код менен тажырыйба *жүргүзүңүз.* Код жазганды үйрөнүүнүн эң жакшы жолдорунун бири - бул тажрыйба. Китептен мисал көрсөңүз, кээ бир параметрлерин өзгөртүп, эмне болуп кеткенин байкаңыз. Болушу мүмкүн болгон эң жаман нерсе – жеңилип калуу ызасы. Эң мыктысычы? Жаңы бир нерсени үйрөнөрүңүз!

Интернеттен “Python” жөнүндө башка окуу куралдарды жана кантип иштөө ыкмасын кароодон *коркпоңуз.* Бул китеп программалоону жаңы үйрөнүп баштагандардын пайдубалын түптөшү керек. Бирок ал сизге бардык нерсени үйрөтпөйт – китептин уландысы ошол максат үчүн! Эгер салыштырмалуу мисалдарды издөөнү чечсеңиз, анда макаланын датасын жана “Python” версиясын карап көрүңүз. Эгерде версия ушул китепте колдонулуп жаткан версияга дал

келбесе (Python 3), сиздин кодуңуз иштебей калышы мүмкүн жана сиз өзүңүздү аябай адаштырып алган болосуз.

Кодуңузду *документтештирип алыңыз*. Биз бул теманы карай элекпиз. Бирок азырынча, *документтештирүү* - кодуңуздун блокторунда же бөлүмдөрүндө чакан комментарийлерди калтыруу дегенди билдирет; ал сиздин (же келечекте дагы бир программисттин) коддун белгилүү бир бөлүгү менен эмне кылууну ойлонуштуруп жатканыңыз саналат. Python эң эле көп окула турган тил болгону менен, ар бир программисттин коддору ар башка болот. Ал эми сизге көрүнгөн нерсе башкаларга дайыма эле байкала бербейт. Ошондой эле, кийинчерээк өзүңүздүн кодуңузду карап чыгууга туура келсе, 10 жыл мурун таңкы саат төрттө эмне кылууга аракет кылып жүргөнүңүздү эсиңизге салат!

Кодуңузду *пландаштырыңыз*. Башкача айтканда, программаңыздын кантип толугу менен иштешин каалай турганыңызды жазып, андан кийин аны кичинекей бөлүктөргө бөлүңүз. Мындан соң, ушул чакан бөлүмдөрдү алып, ар бир бөлүккө эмнени коддоо керектигин белгилеп алыңыз. Ошентип, сизде капысынан эле жасалбаган, аткарыла турган иш-аракеттердин жол-картасы бар.

Акырында кодуңузду тез-тез *сыноодон* өткөрүп, жумушуңузду тез-тез сактап туруңуз. Биз программисттер баш-отубуз менен ишке киришип, бир нече саат бою бир нерсени жалгаштырганды уланта беребиз. Бирок, эгерде биз кодубузду текшерип, файлдарыбызды сактап турууну унутуп калсак, анда бир нече сааттар бою иштеген ишибизди жоготуп алуу коркунучу пайда болот. Андан да кооптусу - табууга кыйын болгон көйгөйлөр менен программа түзүп алабыз.

1-БӨЛҮМ

КОМПЬЮТЕРДИК ПРОГРАММАЛОО ЖАНА PYTHON МЕНЕН ТААНЫШУУ

Компьютердик программалоо адатта жогорку деңгээлде өздөштүргөн колдонуучулар тарабынан "коддоо" деп аталат. Бул - тиркемелерди же программаларды түзүү чеберчилиги. Бул программалар математиканын жөнөкөй маселелерин чыгаруу менен Ютубдан биздин сүйүктүү видеолорубузду көрүүдөн баштап, сүйүктүү видео оюндарыбыздагы келгиндердин сансыз тобун жок кылууга, атүгүл, чыныгы космостук кемени тышкы мейкиндикке учурууга мүмкүнчүлүк берет.

Мен компьютердик программалоону "көркөм өнөр" деп атайм, анткени ал, чынында эле, көркөм өнөр. Каалаган убакта бир нерсе жаратсаңыз, көркөм чыгармачылыкка берилип кетесиз. Албетте, компьютер коду - программабызды түзүү үчүн программалык кабыкка кирген сөздөр (бул тууралуу кийинчерээк сө болмокчу). Балким, сиздин кодуңуз көчөдөгү жөнөкөй адам үчүн сулуу көрүнбөсө керек. Ал көркөм сүрөт көргөзмөсүнөн орун албайт. Бирок сиздин программанын бир бөлүгү сиз өзүңүз жараткан нерсени жасаганда көзгө көрүнөт. Мына ошондо сизге мындан өткөн сыйкырдуу эч нерсе жоктой сезилет.

Мүмкүн, алар парашют менен секирген бульдогдор.

Компьютердик программа ар кандай формада жана өлчөмдө болот. Сиздин жумушчу үстөлүңүздө иштеген колдонмодон же сүйүктүү видео оюн консолуңузда ойногон оюндан тышкары, программалар уюлдук телефондо, мобилдик тиркемелердин формасында да болот. Ал гана эмес, муздаткыч, апаңыздын минивени, торт бышыруучу меш сыяктуу нерселерди иштеткен программалык камсыздоолорду таба аласыз.

Ошондой эле роботтор тууралуу, алардын аскерлерине кийинчерээк токтолобуз.

Азырынча компьютер программасы - бул *программалоо тилинде* түзүлгөн компьютерге же башка аппаратка логикалык бир катар көрсөтмөлөрдү аткарууну сунуш кылган коддордун жыйындысы.

Программалоо тилдери жөнүндө

Жогоруда айтылгандай, компьютердик программа программалоо тили аркылуу жазылат. Сиз, мен жана бүткүл дүйнө жүзү күн сайын кадимки тилибизде сүйлөгөндөй эле, компьютер тилдери да ар кандай формада жана көлөмдө болот. Алардын көпчүлүгү адистерге түшүнүктүү болгону менен, кодду жаңы баштаган адам аны күнүмдүк сүйлөшүүдө колдонууга аракет кылса, акылы ордунда эмес адам сүйлөгөндөй сезилет. Ошол диалог төмөнкүдөй көрүнүшү мүмкүн:

Кадимки адам: Саламатсызбы, кандайсыз?

Программист (Сиз): Print Мен жакшымын! Input, Сен кандайсың?

Бактыга жараша, бардык катышуучулар үчүн компьютерлер программалоо тилдеринде эркин сүйлөйт (бир жагынан, биздин *түзүүчү* досубузга рахмат - бирок кийинчерээк бул тууралуу көбүрөөк маалымат болот) жана сиз киргизген сүйлөмдөрдүн эң татаалын оңой эле түшүнө алат.

Бул китепти максатына жеткирүү үчүн биз ар тараптуу, бирок үйрөнүүгө оңой тилдердин бири *Python программалоо тилине* кайрылабыз. Бул ысым коркунучтуу угулганы менен, андан да жаман ат болушу мүмкүн болгондугу эсиңизде болсун. Айталы, Кобра деп атап койсок болот. Чындыгында, тил сойлоочулардын атынан эмес, тескерисинче, Улуу Британияга таандык *Питон Монти жана Учуучу Цирк* аттуу эски телекомедиянын атынан алынган.

Сиздин биринчи тапшырмаңыз: Ата-энеңизден ошол шоу жөнүндө сураңыз. Бир нече сааттан кийин жолугушканга чейин!

- Ой, сиз кайтып келип келдиңизби? Абдан жакшы. Алардын айткандары кандайдыр бир мааниге ээ болду бекен? Балким, мааниси жок болгон чыгар. Бирок эч нерсе эмес. Бул китепти колдонуп, программалоону үйрөнүү үчүн британ комедиясынын татаалдыгын түшүнүүнүн кажети жок. Сизге керектүү нерсе - бул окууга болгон **каалоо, компьютер жана алдыңыздагы китептин баракчалары.**

Python программалоо тили тууралуу маалымат

Python - жогорку деңгээлдеги, динамикалык, интерпретацияланган, объектиге багытталган программалоо тили. Мунун бардыгы бир аз олуттуу угулса да, андан эч качан чочулабаңыз! Ушул китептин аягында сиз досторуңузду жогорудагыдан алда канча коркунучтуу сүйлөмдөр менен таң калтыра аласыз! Негизи, айтайын дегеним - Python негизги машина деңгээлиндеги тил эмес, ошондуктан компьютер сиз эмне деп айтып жаткандыгыңызды түшүнүшү үчүн, аны компьютердик тилге "*топтоо*" үчүн "*котормочу"* керек.

Бул котормочу сиздин кодуңузду кабыл алып, аны компьютер өзү так түшүнгөн 1 жана 0 сериялары айландырат же *түзөт*. Мунун бардыгы арткы планда болот, андыктан аны азырынча жакшы түшүнбөсөңүз, кабатыр болбоңуз.

Python - 1980-жылдардын аягында пайда болгон салыштырмалуу жаңы программалоо тили. Сиздин атаңыз күлкүлүү муруту менен, апаңыз *Wham* жана *Poison* сыяктуу топторду укканда чыккан!

Бул тилди жараткан адам Гвидо Ван Россум аттуу компьютердин генийи болгон. Ага кооз, бирок көп деле мааниси жок “Benevolent Dictator for Life” - Жашоо үчүн Айкөл Диктатор деген наам берилген. Технология сыяктуу эле, программалоо тилдери да өркүндөп-өсүп турат. Python да алардан айырмаланбайт. Бул жылдар аралыгында бул тил бир нече жолу жаңыланып, учурда Python 3 версиясы колдонулууда.

Python башка программалоо тилдеринен эмнеси менен айырмаланат?

Python башка программалоо тилдеринен бир топ башкача келип, ошону менен бирге, маанилүү жактары менен айырмаланат. Жаңы баштоочулар үчүн Python программалоо тилин үйрөнүү жана колдонуу бир эле класстагы Java, C ++ сыяктуу тилдерине караганда оңой. Python программалоо тилинде түзүлгөн программалар дагы аз убакытты талап кылат. Анткени ал жалпысынан азыраак код талап кылат. Бул жарым-жартылай Python программалоо тилинин *маалымат түрлөрү* менен байланыштуу болот. Бул терминди жакынкы бөлүмдө кеңири талкуулайбыз.

Python табияты ар тараптуу келет. Python негизги тандоо болбосо дагы, аны дээрлик ар бир аренада, анын ичинде оюн, жумушчу үстөлдөгү компьютердин программалык камсыздоосунда, мобилдик тиркемелер жана виртуалдык аныктык системасы үчүн тиркемелерде колдонсо болот. Бул ошондой эле тармактык программалоодогу жана компьютердин коопсуздук куралдар пакетиндеги маанилүү курал болуп саналат.

Python программалоо тилинин артыкчылыктары

Азыркы учурда Python - дүйнөдөгү эң көп колдонулган программалоо тили жана ал эң ылдам өсүп келе жаткан программа. Минтип айтканга жакшы негиз бар. Төмөндө Python тилинин программистке пайда келтире ала тургандыгы тууралуу бир нече ыкмалары берилген:

- **Өндүрүмдүүлүктүн жогорулашы:** Python программисттин эмгек өндүрүмдүүлүгүн жогорулатат. Берилген убакытта алар он эсе көп жумушту бүтүрө алышат! Бул сөзсүз түрдө учкан октон ылдамыраак!
- **Кеңейтүү:** Python программалоо тилинин бир чоң артыкчылыгы - кеңири китепканага ээ экендиги. Китепкана - бул программаңызга кошо турган бардык коддордун жыйындысы. Бул китепканалар программанын жалпы мүнөздөмөсү болгон нерселерди камтыйт жана сизди кодду кайра-кайра жазуудан куткарат. Мисалы, татаал математикалык теңдемени чыгаруу үчүн коддун бир бөлүгүн жазуунун ордуна, китепкананы колдонуп, баш оорудан кутулсаңыз болот.
- **Уйрөнүүгө жөнөкөй:** Программист болуунун эң бир оор бөлүгү - кээде кодуңуз иштебей калат. Ушундай болгондо, сиз дагы бир жолу кодуңузду, же башка бирөөнүн кодун дагы бир жолу окуп чыгып, эмне үчүн программаңыз өз ишин так аткарбай жаткандыгын билип алышыңыз мүмкүн. Бактыга жараша, Python тилин окуу оңой жана түшүнүктүү. Бул тил татаал тилдерге караганда көйгөйлөрдү табууну бир топ жеңилдетет.
- **Портативдүү:** Python көптөгөн платформаларда жана тутумдарда иштейт. Демек, программаларыңыз кеңири аудиторияны камтыйт.
- **Нерселер интернети** (*internet of things* - IoT): Нерселер интернети санарип жырткычтарга толгон сыйкырдуу дүйнө сыяктуу сезилиши мүмкүн жана кандайдыр бир деңгээлде, ошондой да болуп саналат. Нерселер интернети сиз күнүмдүк үйдөн тапкан акылдуу нерселерден - ачкычтардан, эшиктин туткаларынан, тостерлерден, тиричилик техникасынан турат. Бул

тиричилик техникаларын үн буйруктары жана мобилдик түзмөктөр аркылуу башкарууга болот. Бул аларды примитивдүү мурункуларга караганда интерактивдүү кылат. Албетте, сенин апаң менен атаң идиш жуугуч машинага дайыма кыйкырышкан. Буюмдар аларды укту болду бекен? Эми болсо, IoT жана Python сыяктуу тилдердин жардамы менен алар угушу мүмкүн! Ошентсе да, анын ичине дагы деле болсо идиш-аякты өзүңүз саласыз!

- **Python алкактары**: Алкактар программанын скелеттерине окшош келет. Алар сиз иштеп жаткан программалык камсыздоонун түрүндө бар болгон жалпы элементтердин кодун талап кылбастан, айрым тиркемелердин негиздерин тез орнотууга мүмкүнчүлүк берет. Бул нерсе программалоочулардын убактысын үнөмдөп, код жазуу учурларда боло турган каталардын санын азайтат. Python программалоо тилин жаңы программаны ишке ашырууну өтө тездик менен жүргүзө турган көптөгөн алкактар колдойт!
- **Python кызыктуу**: Python - үйрөнүүгө кызыктуу тил. Бул тил менен программалоону баштоо оңой гана болбостон, Python жамааты көптөгөн кызыктуу иш-чараларды өткөрүп турат. Мисалы, көптөгөн адамдар Python кодун поэзия түрүндө жазышып, жыл сайын программистердин жөндөмүн текшерүүгө жардам берүү үчүн көптөгөн Python "*мелдештерин*" өткөрүп турушат.
- **Python ийкемдүү**: Python ушунчалык көп колдонулгандыктан жана аны дүйнө жүзү боюнча көптөгөн компаниялар колдонуп жаткандыктан, Python үйрөнгөндөн кийин жумуш табуу башка тилдерге караганда жеңилирээк. Мындан тышкары, эгер сизге берилген жумуш жакпаса, анда ар дайым Python көндүмдөрүн колдонуп, башка жолду байкап көрсөңүз болот. Мисалы, коддоо тиркемелери кызыксыз экендигин байкасаңыз, тармактык башкарууга өтүп же IT коопсуздук фирмасында иштесеңиз болот.

Булар Python программалоо тилинин пайдаларынын жана артыкчылыктарынын айрымдары гана.

Python программалоо тилинин мисалдары

Python программалоо тилин дүйнө жүзү боюнча канча компания колдонгонун айтууга мүмкүн болбосо да, бул тилге таянган бир катар кызыктуу ишканалар бар. Төмөндө алардын айрымдары гана саналган:

- Wayne Enterprises (Batman's Alter Ego корпорациясы): Ооба, биз, чындыгында, муну билбейбиз, бирок сонун эмеспи!
- Google: Издөө системасынын гиганты жана галактиканын бир күндүк башкаруучусу Google Python программалоо тилин негизделгенден бери колдонуп келе жатат. Себеби аны иштеп чыгуучулар программаларды тездик менен кура алышат жана кодду оңой колдоно алышат.
- Facebook жана Instagram: Python - ушул эки социалдык медиа платформада жалгыз гана колдонулган программа болбосо да, алардын эң маанилүүлөрүнүн бири. Facebook Python программалоо тилин анын кеңири китепканаларынын жардамы менен колдонот. Ошол эле учурда

Instagram Python программалоо тилинин Django веб-алкагынын бекем колдоочусу болуп эсептелет. Веб алкактарын ушул китептен кийинчерээк кененирээк чагылдырабыз.

- Netflix: Эгер сиз платформадагы кинолордун күйөрманы болсоңуз, анда сиз Netflix үчүн чоочун эмессиз. Компания Python программалоо тилин биринчи кезекте маалымат талдоо мүмкүнчүлүктөрү үчүн жана коопсуздук максатында колдонот.
- Видео оюндар: Battlefield 2 жана Civilization 4 – Python программалоо тилине таянган эки видео оюн. Баарынан кызыгы - Civilization Python программалоо тилин башкалар менен катар өзүнүн жасалма интеллект сценарийлери үчүн колдонот.
- Мамлекеттик агенттиктер жана мекемелер: Мамлекеттик органдар жана мекемелер, анын ичинде НАСА, Улуттук Аба ырайы кызматы жана АКШ Улуттук чалгындоо кызматы Python программалоо тилин колдонушат. Бирок анын кандайча колдонулганы өтө жашыруун сыр болуп эсептелет! Бизди унаа токтотуучу жайда ичи акчага толгон портфель менен тосуп алыңыз. Биз сизге бул тууралуу кийинчерээк кенен айтып беребиз!

Сиздин биринчи Python программаңыз

Учурда Python коду эмнеге окшош деп ойлонуп жатсаңыз керек. Ооба, коркпоңуз! Мен сизге үзүндүнүн үлгүсүн көрсөтөм. Кийинчерээк, компьютеribизге Python жана IDLE (интегралдык өнүгүү чөйрөсү) орнотулгандан кийин, кодду иш жүзүндө көрүү үчүн аны иштетип көрүңүз же иштетсеңиз болот. Бирок, азырынча, тилге чөмүлүүдөн мурун, жөн гана аракет кылып көрсөңүз жакшы болмок деп ойлонуп турам.

Адатта, программисттер биринчи жолу коддун саптарын жазганда, алар өздөрүн дүйнөгө тааныштыруунун метафоралык жолу катары "Салам, Дүйнө" деп аталган программа түзүшөт. Бирок, жаңы баштаган супер баатырлар - же терс каармандар (эч ким айыптабайт) - бизге көңүл бурдура турган бир нерсе керек.

Мына, биринчи Python программаңыз!

```
print("Асманды караңыз! Бул кушпу? Бул учакпы?")
print("Дун дун дун дун дун дун дун дун")
print("Жок. Бул жөн гана пижамасы менен учкан жигит. Кана, иштегиле!")
```

Эгер сиз ушул кодду иштете турган болсоңуз, анда натыйжасы:

```
Асманды караңыз! Бул кушпу? Бул учакпы?
Дун дун дун дун дун дун дун дун
Жок. Бул жөн гана пижамасы менен учкан жигит. Кана, иштегиле!
```

Келгиле, кодду дагы бир аз карап чыгалы. *Print* () деген бөлүк *функция* катары белгилүү, анын милдети компьютерге - - колдонуучунун экранына бир нерсени *басып чыгаруу*. Ачылуучу жана жабылуучу кашаанын () ортосундагы

текст - бул функцияны камсыз кылган нерсе *параметр*. Тырмакча "" белгилеринин ортосундагы белгилер сап-string катары белгилүү.

Эгер бул азырынча маанисиз болсо, кабатыр болбоңуз - бул теманы кийинки бөлүмдө кененирээк карап чыгабыз. Азырынча, бул Python коду кандай экендигин билип алыңыз. Мүмкүн, сиз бул программанын эмне кыларын мен сизге айтканга чейин эле билип алгандырсыз; Python программалоо тилин ушунчалык мыкты кылган нерселердин бири - анын жеңил окула тургандыгы!

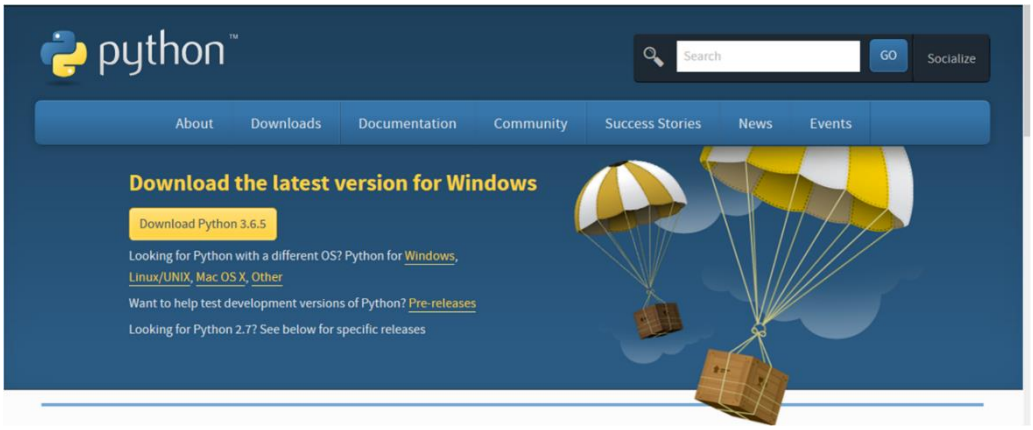
Python иштөө мейкиндигин орнотуу

Бул бөлүмдө биз ар кандай *операциялык системаларга* Python иштөө мейкиндигин орнотууну үйрөнөбүз. Операциялык система - бул компьютер менен иштешүүгө мүмкүнчүлүк берген программалык камсыздоо. Сиз *Microsoft Windows* жана *Mac OS X* сыяктуу белгилүүлөрүн жакшы билсеңиз керек. Сиз орноткон Python версиясы компьютериңизде кайсы операциялык системаны колдонгонуңузга жараша өзгөрүлүп турат. Мындан сырткары, Python иштөө мейкиндигин *Linux* жана *Ubuntu* операциялык системаларына орнотууну үйрөнөбүз.

Python иштөө мейкиндигин Windows операциялык системасына орнотуу

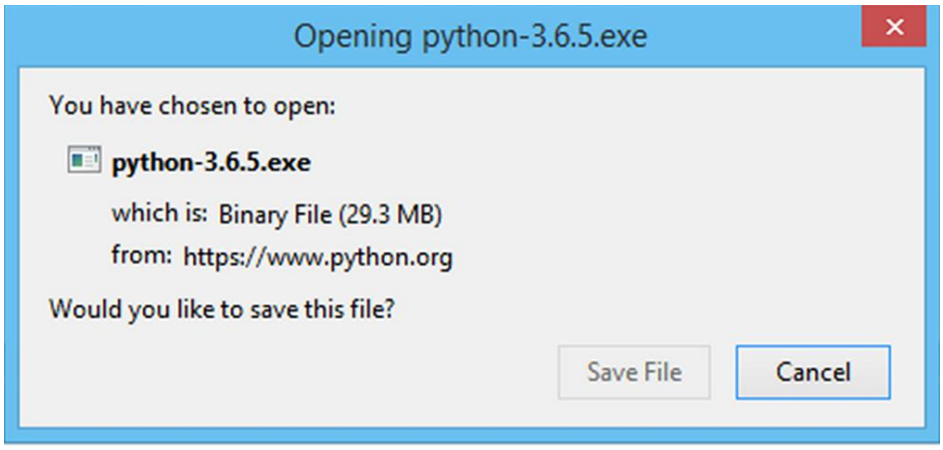
Баштоо үчүн, веб-браузерди ачып, Python расмий баракчасына кириңиз жана анын жүктөө барагына өтүңүз: www.python.org/downloads/ (1-1-сүрөт).

Программанын учурдагы нускасы - 3.6.5; сиз бул китепти окуган кезде, андан да жаңы нускасы чыгышы мүмкүн. Кандай болбосун, Windows үчүн акыркы нусканы жүктөп алуу үчүн *Python жүктөө - Download Python* баскычын басыңыз. Кааласаңыз, ылдый жылдырып, мурунку нускасын жүктөп алсаңыз болот (алардын 3.X же андан жогору нуска экендигин текшериңиз, анткени 2.X жана 3.X нускаларында дал келбөө маселелери бар). Бирок, бул китептеги түшүндүрөлөр үчүн 3.6.5 же андан кийинки нускаларын колдонуу эң жакшы болот.



Сүрөт 1-1. *Python.org website*

Файлды сактоону сураган сүрөт пайда болот. *Файлды сактоо - Save File* – баскычын басасыз (1-2-сүрөт) жана аны “иш үстөлүңүзгө” же оңой эсте калган жерге сактап коюңуз.



Сүрөт 1-2. *Python орнотуу файлдары үчүн файл диалогун сактаңыз*

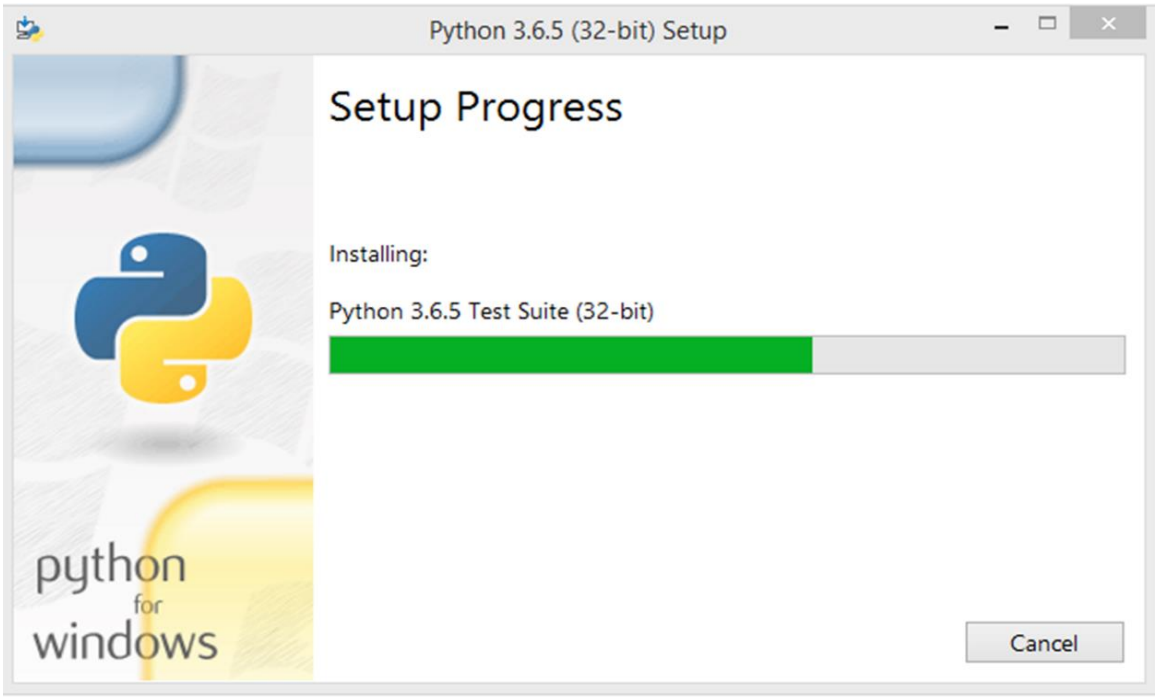
Иш үстөлүңүзгө (же файл сакталган жерге) өтүп, аны эки жолу басыңыз. Ал 1-3- сүрөткө окшош көрүнүшү керек.

Сүрөт 1-3. *Python .EXE орнотуу файлынын сүрөтү*

Орнотуучу ишке кирет жана сизден "Азыр орнотуу" - “Install Now” же "Орнотууну ыңгайлаштыруу" - “Customize Installation”- керекпи деп сурайт. Оңой болуш үчүн орноткучка "Азыр орнотууга" уруксат беребиз. Ошол баскычты басуудан мурун, "Бардык колдонуучулар үчүн ишке киргизгичти орнотуу"- “Install launcher for all users” - жана "PATH Python 3.6 кошуу"- “Add Python 3.6 to PATH”- деген экөөнү тең белгилеңиз. Андан кийин "Азыр орнотуу" деген тандоону басыңыз (1-4-сүрөт).

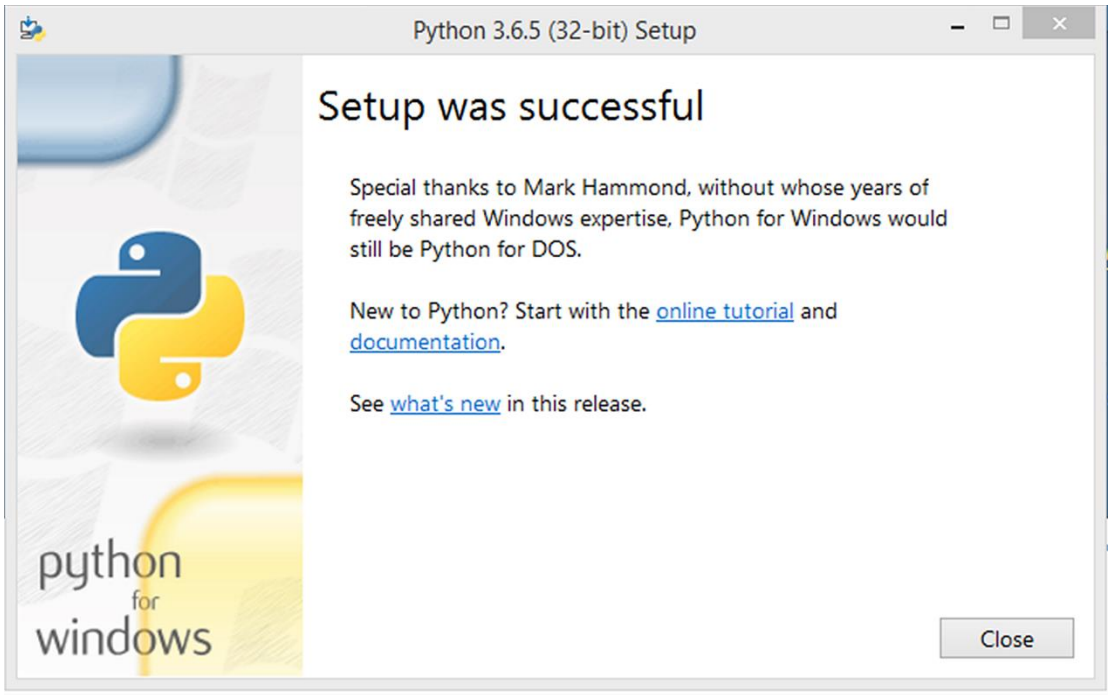
Орнотууну улантуу үчүн Windows системасынан уруксат сураган калкыма терезе пайда болушу мүмкүн. Эгер ошондой болсо, анда программанын уланышына уруксат бериңиз. Сизге орнотуу прогрессин көрсөткөн жаңы калкыма терезе пайда болот (1-5-сүрөт).

Сүрөт 1-4. *Python орнотуу экраны*



Сүрөт 1-5. *Python орнотуу прогрессинин экраны*

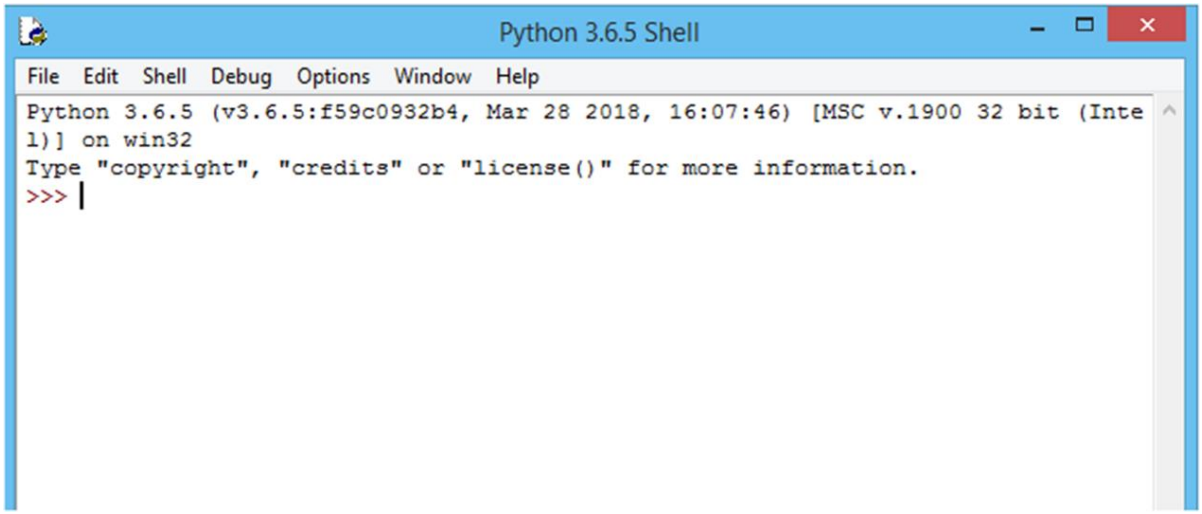
Орнотуу аяктагандан кийин, төмөндөгүдөй экранды көрө аласыз. Орнотууну аяктоо үчүн *Жабуу* – *Close* баскычын басыңыз (1-6-сүрөт).



Сүрөт 1-6. *Python орнотуу ийгиликтүү аяктады*

Эми компьютериңизде Python орнотулган болушу керек. Аны Python 3.6 деп белгиленген “Start” менюсунан таба аласыз (же кайсы нускасын орнотсоңуз дагы).

Python иштөө мейкиндигин колдонууда, биринчи кезекте, коддун саптарын, тесттик коддорду жазуу, иштетүү коддору жана Python файлдарын түзө турган өнүгүү чөйрөсүнүн бөлүгү болгон кабык – shell - көрүнөт. 1-7-сүрөттө Python Shell ишке киргизилгенден кийин кандайча пайда болорун көрсөткөн мисал келтирилген.



Сүрөт 1-7. *Python Shell*

Бул кабык терезенин жогору жагында сиз учурдагы версиясын жана башка маалыматтарды көрө аласыз. Ошондой эле үч жебени көрө аласыз (>>>). Булар буйрук сабы деп аталат жана ушул жерде сиз программага берген көрсөтмөлөрүңүздү жазасыз.

Тереңирээк чөмүлүүгө даярсызбы? Келгиле, жөнөкөй кодду жазып, эмне болуп жатканын көрөлү! Буйрук сабына төмөнкүнү киргизиңиз:

```
print("Асманды караңыз! Бул кушпу? Бул учакпы?")
```

Бүткөндөн кийин, Enter баскычын бассаңыз, анда төмөнкүдөй натыйжаны көрүшүңүз керек (1-8-сүрөт):

```
>>> print("Асманды караңыз! Бул кушпу? Бул учакпы?")
Асманды караңыз! Бул кушпу? Бул учакпы?
>>>
```

Сүрөт 1-8. *Python Shell терезесинде жазылган код*

Эгер андай болбосо, анда кодуңузду дагы бир жолу текшериңиз, кашаа () жана тырмакча “” белгилерин коюуну эсиңизден чыгарбаңыз.

Биз түздөн-түз кабыкта иштеп жаткандыктан, биздин код реалдуу убакыт режиминде аткарылат же иштетилет. Бул учурда, ал компьютердин экранына тексттин сабын басып чыгарууну айткан бир эле коддуу сапты иштетип койду.

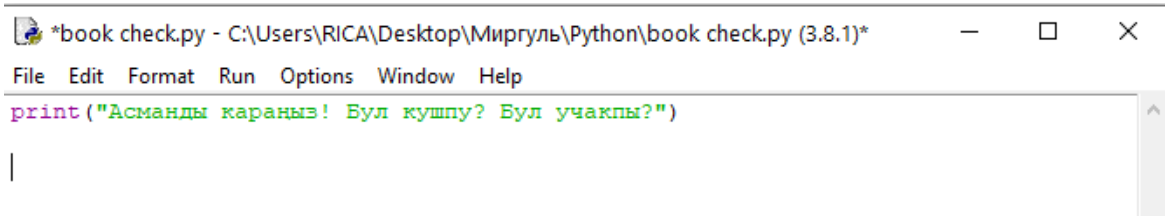
Чыныгы чөйрөдө биз Python файлдарын түзүп, программаларыбызды кийинчерээк колдонуу үчүн сактап кала

алабыз жана программабызды иштеткен сайын, миңдеген саптарды кайрадан жазуудан кутулабыз.

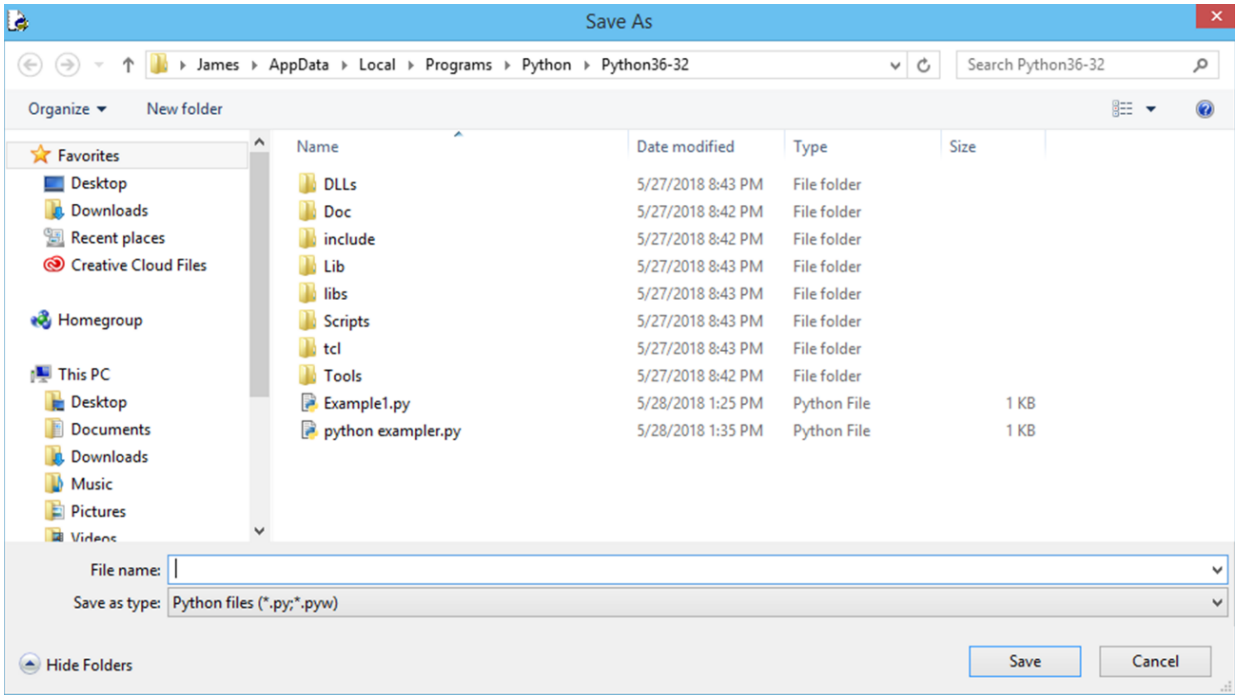
Бактыга жараша, Python IDLE - же өнүгүү чөйрөсү - бизге Python файлдарын, башкача айтканда extension.py менен аяктаган файлдарды оңой эле түзүп берет. Болгону *File*, андан соң *New File* баскычын басуу керек (1-9, 1-10 жана 1-11-сүрөттөрдү караңыз).

Сүрөт 1-9. *Жаңы түзүлгөн .py файлы*

Жаңы терезе калкып чыгат. Бул жерде сиз кодуңузду жазып, кийинчерээк сактай аласыз. Айткандай эле, жаңы эле колдонгон мисал кодун киргизели. Андан соң *File*, андан кийин *Save* баскычын басасыз.



Сүрөт 1-10. *.py файлында жазылган мисал код*



Сүрөт 1-11. *Python каталогун көрсөткөн диалог кутучасын сактоо*

Файлдын атын киргизип, Save баскычын басып, файлды түзүп бүтүрүңүз. Бул китептин максаттары үчүн, нерселерди жөнөкөй кылып, *Example1. Py* деп ат берели.

Мына бүттү - сиз биринчи чыныгы Python программаңызды түздүңүз. Бул программаны иштетүү үчүн, Run баскычын басып, андан соң Run Module баскычын тандаңыз. Эми сиздин программа Python кабыгында иштейт! (Сүрөт 1-12).

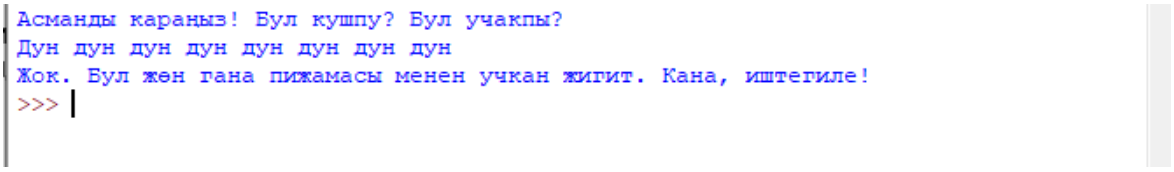
Сүрөт 1-12. *.Py файлынын натыйжасы - Python Shell*

Эми, бардыгын жыйынтыктап көрөлү: ушул бөлүмдүн башында жазган алгачкы программабыз эсиңиздеби? Келгиле, аны биздин Example1.py файлына

киргизип, аяктагандан кийин Save баскычын бир жолу басып коёлу. Бул жерде дагы бир жолу код бар:

```
print("Асмандан караңыз! Бул кушпу? Бул учакпы?")
print("Дун дун дун дун дун дун дун дун")
print("Жок. Бул жөн гана пижамасы менен учкан жигит. Кана, иштегиле!")
```

Файлды сактап койгондон кийин Run баскычын басып, Run Module баскычын тандап, толук кодду иш жүзүндө көрүңүз! (Figure 1-13).



Сүрөт 1-13. *Python Shell .py файлынын дагы бир мисалы*

Python иштөө чөйрөсүн башка операциялык системаларга орнотуу

Бул китепте Windows операциялык системасы иштеген компьютерде орнотулган Python колдонулат. *И*чиндеги код *ар кандай* компьютерде иштесе дагы, негизинен Python иштөө чөйрөсүн орнотуу сиздин ишмердүүлүгүңүз кандай операциялык системада иштегенизге жараша өзгөрүп турат.

Mac OS X операциялык системасына орнотуу үчүн веб браузерди ачып, www.python.org/downloads/mac-osx/ сайтына өтүңүз. "Акыркы Python 3 чыгарылышы" - “Latest Python 3 Release” шилтемесин тандап, Setup жана Installation командаларын аткаруу менен нускамаларды жана сунуштарды аткарыңыз.

Python иштөө чөйрөсүн Unix/Linux операциялык системасына орнотуу үчүн, браузерди ачып, www.python.org/downloads/source сайтына өтүңүз. "Акыркы Python 3 чыгарылышы" - “Latest Python 3 Release”- үчүн шилтемени басып, Setup жана Installation командаларын ишке киргизүү менен көрсөтмөлөрдү аткарыңыз.

Бул эпизоддо

Бул бөлүмдө биз, албетте, көп нерсени баяндадык, бирок кийинки бөлүмдө бериле турган маалыматтар менен салыштырмалуу бул эч нерсе эмес! Бул жерде кыскача тизме келтирилген. Эгер сиз кааласаңыз, ушул убакытка чейин камтылган окуу материалдардын жыйынтыгын (достор, биз азыр баатырларды программалайбыз, биз лингводо да сүйлөшүшүбүз керек!) карап көрүңүз.

- Python - бул компьютерлерди, мобилдик шаймандарды, видео оюндарды, жасалма интеллект системаларын, Нерселер интернетинин (IoT) түзмөктөрүн, вебке негизделген тиркемелерди, ал тургай, виртуалдык чындыкты программалоого мүмкүндүк берген программалоонун тили.

- Программа же тиркеме - бул компьютерге же шайманга аткарууга бир катар көрсөтмөлөрдү берүүгө мүмкүндүк берген коддордун тобу.
- Python программисттери программалоо, тармактык администрация, IT коопсуздугу, видео оюндарды иштеп чыгуу, мобилдик тиркемелерди иштеп чыгуу, соттук-компьютердик илим жана башка тармактарда иштей алат.
- Python бир нече платформаларда иштейт, анын ичинде Windows ЖК, Mac компьютерлери, мобилдик шаймандар, Unix/Linux системалар жана башкалар бар.
- Python "этикалык, хакерлик" инструменттер деп аталган көндүмдөрдүн жана модулдардын жыйындысы аркылуу хакерликтин алдын алуу үчүн колдонулат.
- IDLE комплекстүү өнүгүү чөйрөсүн билдирет. Ал жерде биз Python кодубузду жана файлдарыбызды түзөбүз.
- Python тарабынан түзүлгөн файлдар ".py" кеңейтүүсүндө аяктайт.
- Бул китепти жазуу учурунда Python 3.6.5 колдонулган. Эгер сиз бул китепти окуп жатсаңыз, анда сөзсүз түрдө ушул нусканы же кийинчерээк чыккан нускасын колдонуңуз.
- Print () функциясы текстти колдонуучунун экранына жазып чыгарууга мүмкүнчүлүк берет. Мисалы, print ("Салам") компьютер экранына: Салам деген текстти басып чыгарат.
- Python дүйнө жүзү боюнча көптөгөн уюмдар жана компаниялар, анын ичинде Facebook, Google, Snapchat жана башкалар тарабынан колдонулат!
- Python - дүйнөдө эң көп колдонулган жана тездик менен өсүп келе жаткан компьютердеги программалоонун тили.

2 - БӨЛҮМ

БИЗ ҮЧҮН БАРДЫГЫ МААНИЛҮҮ

Элестүү айтканда, жабдуулар жана коргоочу кийимдер менен камсыздалган супер баатырга биз эми гана окшоштук (б.а. биз программаны орнотуп, кантип колдонууну үйрөндүк). Эми жаңы супер күчтөрүбүздү сынап көрүүгө кез келди! Биздин биринчи каршылашыбыз ким? Балким, бардык мезгилдердеги кыялы эң оор, менменсинген текебер, мектептеги далай окуучуларды жүрөгүнүн үшүн алып, билбегендерди кылмышкерге айландырган бирөө бар. Ал ким болушу мүмкүн? Анын аты ким?

Албетте, ал - математика.

Билем, башталышында бул предмет көпчүлүгүңөргө жага бербейт чыгар. Башыңарды оорутуп эле эсеп чыгара бересиңер. Бирок, чынын айталы, математика жана андан да маанилүүсү - математикалык функциялар программалоо дүйнөсүнүн наны менен майы болуп саналат. Математика болбосо, бизге көптөгөн мүмкүнчүлүктөрдү берген бир дагы сонун компьютерлерди жана мобилдик түзмөктөрдү жасай алмак эмеспиз. Математика болбосо компьютердик оюндар, космос мейкиндигиндеги кемелер, биздин башаламан болуп чачылып жаткан бөлмөлөрүбүздү тазалоого жардам бере турган келечектин роботтору болмок эмес.

Математика болбосо, биз жоголгон цивилизацияга айланмакпыз. Ошондуктан, бул бөлүмдүн максаты - математика менен кантип достошуу керек жана Python программалоо тилине мүнөздүү математикалык функцияларды колдонуп, жөнөкөй же татаал математикалык теңдемелерди кантип түзүү керек экендигин үйрөтүү.

1-бөлүмдөн үйрөнгөн print() функциясы сыяктуу эле, биз талкуулай турган математикалык функциялар, колдонмонун жалпы элементтерин коддобой туруп эле, алдын-ала курулган иш-аракеттерди жасоого мүмкүнчүлүк берет. Ошентип, мисалы, компьютерге кошуу деген эмне экендигин жана иш жүзүндө сандарды кантип кошуу керектиги жөнүндө көп коддорду жазбайбыз (эсиңизде болсун, компьютер биз айтканды гана жасайт. Ал азырынча өзү ойлоно албайт!). Эгерде биз муну нөлдөн баштап жасасак, анда миңдеген саптар талап кылынмак. Биз болгону жөнөкөй нерсени киргизишибиз керек. Мисалы:

```
1+1
```

Алгач муну Python кабыгына жазыңыз. Сиз жазганда, ал 2 деген жоопту дароо эле бериши керек.

Сиз мектепте үйрөнгөн математика сыяктуу эле, Python да негизги математикалык функцияларды түшүнүү үчүн курулган. Эгер сиз 8/2 көрсөңүз, сиздин мээңиз бул теңдеме бөлүүнү камтыйт деп түшүнөт. Эгер сиз + белгисин көрсөңүз, анда ал кошууну, - белгиси кемитүүнү билдирет. Python бул белгилерди да жакшы

түшүнөт жана алардын негизинде математикалык эсептөөлөрдү жүргүзөт. Python кабыгына 2+2-1 деп киргизип көрүңүз.

Мындай учурда, Python 3тү чыгарып, жалпы *математикалык операторлорду* түшүнөрүн көрсөтөт. Python оператору төмөнкүлөрдү камтыйт: +, - жана / * бир нечесин атаса болот.

Көбөйтүү жөнүндө эмне айтууга болот? Муну жазыңыз:

2x2

Ал жерде эмне болду? Программа биз күткөндөй 4 тү чыгарып берген жок. Анын ордуна ал "*SyntaxError:"* – "*Синтаксистик ката*" кырдаалын кайтарып берди. *SyntaxErrors* деген билдирүү синтаксисте жазуу жүзүндөгү тексте бир нерсе ***туура эмес*** дегенди билдирет. Сиз программанын жакшы иштешине тоскоол болгон катаны Python файлына киргизип жатасыз.

Башкача айтканда, Python сизди түшүнбөй жатат.

Бул жердеги чечим жөнөкөй: Python көбөйтүү оператору "x" эмес, жылдызча (*). Биздин *SyntaxError* билдирүүсүн оңдоо үчүн туура эмес көбөйтүү операторун туурасы менен алмаштыруу керек, мисалы:

2*2

Эми аны жазсаңыз, анда сиз күткөн жооп - 4 саны чыгат.

Оператордун артыкчылыгы

Аябагандай олуттуу математиканын супер күчү түшүнүүгө өтө кыйын сезилген түшүнүктөрү менен бизди коркутат. Андан эч качан коркпоңуз! Биздин супер баатыр эсептөө күчүбүз турганда математиканын эң татаал табышмактары бизге тоскоолдук жарата албайт.

Аракет кылсаңыз, берекет болорун унутпаңыз!

Python программалоо тилинде эсептөөлөрдү жүргүзүүдө биз ар дайым *оператордун артыкчылыгы* деген нерсени билишибиз керек. Бул Python программалоо тилинде математикалык тапшырмаларды аткаруунун өзгөчө мисалы болуп саналат. Айрым операторлордун артыкчылыгы жогору, башкача айтканда, алар башка операторлорго караганда биринчи орунда турушат. Сиз элестеткендей, бул программисттин башын айландырып, ал тургай, эң тажрыйбалуу деген адистер эсептөөлөрдү киргизүүдө ката кетириши мүмкүн.

Операторлордун иерархиясы аларды аткаруунун артыкчылыгына ылайык кандайча иштээрин тагыраак элестетүү үчүн бул жерде Python программалоо тилиндеги операторлордун тизмеси келтирилген. Эскертүү: бул операторлордун айрымдары сизге тааныш эместир. Азыр буга көп камтама болбоңуз. Биз аларды ушул китепте кеңири чагылдырып беребиз.

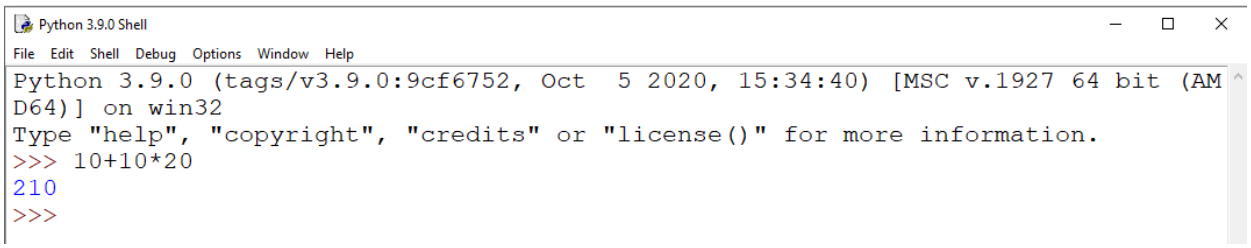
- ** (Даражага көтөрүү)
- *, / (Көбөйтүү жана бөлүү)

- +, - (Кошуу жана кемитүү)
- in, not in, is, is not, <, <=, >, >=, !=, == (Булар бир маанини экинчисине салыштырууга мүмкүндүк берет)
- and, or, not - (Логикалык операторлор)
- if, elif, else (Чындык экенин аныктоо инструкциясы)

Жөнөкөй болуш үчүн, келгиле, негизги операторлор менен иштейли: *, /, + жана - (көбөйтүү, бөлүү, кошуу жана кемитүү). Төмөнкүлөрдү кабыгыңызга териңиз:

```
10 + 10 * 20
```

Бул теңдемеде биз 10 кошуу 10дун мааниси 20га көбөйтүлгөндө кандай болот деп сурап жатабыз. Адатта, жооп 400 болот деп күтсөк болот. Анткени биринчи маани - 10 кошуу 10 - 20га барабар болушу керек. Андан кийин биз бул жоопту (20) 20га көбөйтөбүз. Натыйжада, 400 пайда болушу керек. Бирок кодду кабыкка киргизгенде 2-1-сүрөттө көрсөтүлгөндөй, таң калычтуу натыйжага ээ болобуз.



Сүрөт 2-1. *Операторлордун аткарылышына мисал*

Сизге алгач эле эмне Python математикадан начарбы? Кантип 210 деп жооп берсин? деген ой келет. Бул оператордун артыкчылыгынан жана аларды аткаруудагы иерархиядан улам пайда болот. Операторлордун артыкчылык тизмеси эсиңизде болсун. Көбөйтүү биринчи кезекте турат. Ошондуктан, Python *биринчи* көбөйтүүнү *эсептеп*, андан кийин кошууну эсептейт. Бул учурда Python биздин теңдемени мындайча карайт: 20 * 10 + 10.

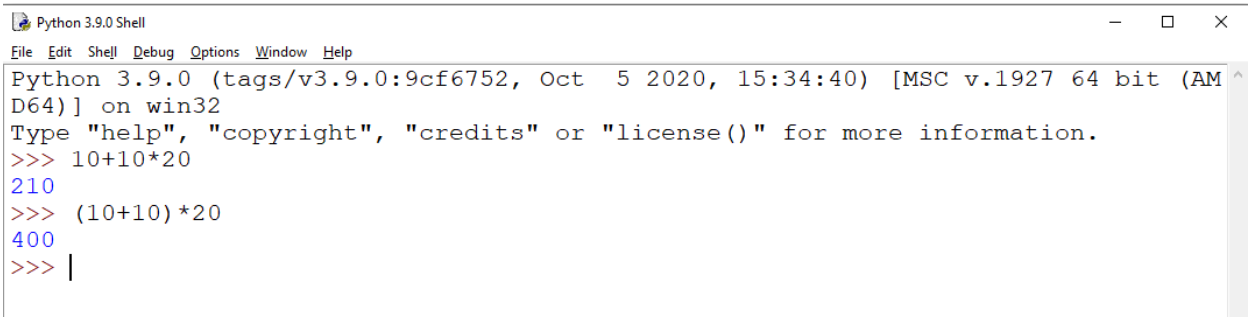
Башыңыз ооруп кеттиби?

Бул бир караганда түшүнүксүз сезилиши мүмкүн. Бирок бактыга жараша, жөнөкөй чечим бар. Кашаанын жардамы менен Python программалоо тилин, баалоо тартибин - эсептөөлөрдү жүргүзүү тартибин колдонууга мажбурлай алабыз. Мунун эки таасири бар: биринчиден, Python биз каалаган эсептөөнү жүргүзөт жана биздин артыкчылыгыбызды адаштырбайт. Экинчиден, ал башка программисттерге сиздин теңдемеңиз, чындыгында, эмнени көздөп жаткандыгын жөнөкөй эле көз караш менен билүүгө мүмкүндүк берет.

Келгиле, аны сынап көрөлү. Төмөнкүлөрдү кабыгыңызга жазыңыз:

```
(10+10)  * 20
```

2-2-сүрөттө көрүнүп тургандай, эми биз каалаган натыйжага ээ болдук. Кашаанын ичине (10 + 10) коюп, биз Python жана башка программистерге теңдеменин ошол бөлүгүн биринчи кезекте аткарылышын көздөгөнбүздү көрсөттүк (2-2-сүрөттү караңыз).



Сүрөт 2-2. *Эсептөө иерархиясында кашаа колдонулган мисал*

Ишти бир аз татаалдаштыруу үчүн биз *уя түзүү* деп аталган нерсени жасап көрөлү. Бул кашаанын *ичине* кашаа жайгаштырганыңызды билдирет, мындан ары кандай тартипте эсептөөлөрдү жүргүзүү керектигин айтат. Ушундай учурда, ички кашаалар, андан кийин тышкы, андан кийин калган теңдемелер бааланат.

Муну карап көрөлү:

((10+5) * 10) / 2

Эгер сиз ушул теңдемени Python кабыгына киргизсеңиз, анда ал эсептөө тартибин төмөнкүдөй тартипте аткарат:

- 10 + 5 барабар 15
- 15 * 10 барабар 150
- 150/2 барабар 75

Бирок, эгер биз кашаа колдонбосок, анда Python аны мындайча окуйт:

10 + 5 * 10 / 2

Же:

• 10/2 барабар 5

• 5 * 5 барабар 25

• 25 + 10 барабар 35

Анткени, Python көбөйтүүнү жана бөлүүнү кошуудан жана кемитүүдөн мурун аткарат.

Андыктан, башаламандыкка жол бербөө үчүн жөнөкөй математикадан башка нерсени жасап жатканда ар дайым кашаанын ичине жазыңыз.

Маалыматтардын түрлөрү: душманыңызды билиңиз

Супер душмандар ар кандай формада жана көлөмдө болот. Сиздин дүйнөнү өлүм нурлары жана генетикалык жактан өзгөртүлгөн гориллалар менен жок кылууга умтулган жаман илимпозуңуз бар. Эч кандай себепсиз эле булчуңдарын титиретип ачууланган жашыл түстөгүсү дагы бири бар. Анан дагы эч ким эч кандай тамаша айтпаса да, жөндөн жөн эле күлө бергендери да бар.

Жаман каршылаштардын миңдеген түрлөрү бар. Жаңыдан калыптанып жаткан супер баатыр катары, сизге алардын бардыгын иретке келтирүү кыйынга турат. Мистер Адаштыруучу супер акылдуубу же ал жөн гана чогуу иштеше албаган түшүнүксүз жаман душманбы? Ал эми табышмактуу Стивен Кинг Конг жөнүндө эмне айтууга болот. Жарымы горилла, жарымы коркунучтуу жазуучу. Бул эмне деген шумдук? Анан кантип ошол чоң горилланыкындай манжалары менен китеп жазат?

Акыл-эсиңизди жоготуп койдуңузбу, анда бул жагы жетиштүү.

Же Мистер Адаштыруучу кайрадан жумуштабы?

Бактыга жараша, ушул кара санатайлардын бардыгын кармаганга мүмкүнчүлүк бар. Алар архетиптер деп аталат.

Бизде Python программалоо тилинде код жазууда да ушундай көйгөй бар. Ар түрдүү маалыматтар айланып жүрөт. Биринчиден, бизде сандар жана текст бар. Ал аз келгенсип, бизде сандардын ар кандай түрү бар. Сизде жөнөкөй сандар, ондук белгилери бар сандар жана убакыт же акча сыяктуу нерселерди чагылдырган сандарыңыз бар.Ал эле эмес, өзүн сөз сыяктуу алып жүргөн сандар да бар.

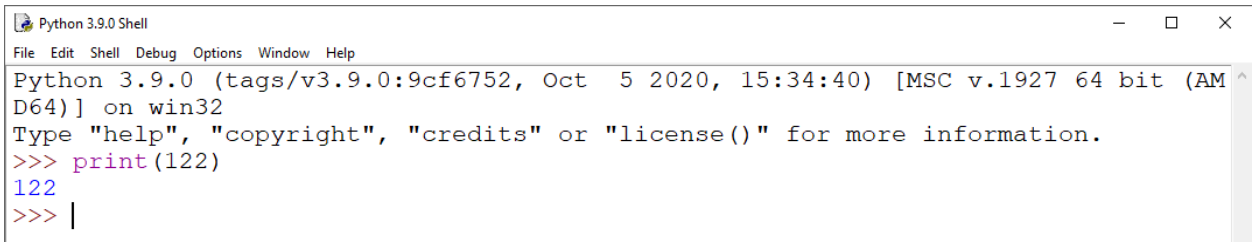
Бактыга жараша, Python программалоо тилинде маалымат түрлөрү деп аталган нерсе бар. Булар программаларыңызга кандай типтеги маалыматтарды киргизип жаткандыгыңызды аныктоонун же классификациялоонун жолдору болуп саналат. Чындыгында, бул жүйөлүү ой болушу мүмкүн. Бирок, негизинен, Python сиз эмнени айтсаңыз ошону гана билет. Ушундай эле нерсе бардык программалоо тилдерине тиешелүү. Чындыгында, бардык программалоо тилдерде Python сыяктуу маалыматтардын түрлөрү бар. Демек, бул типтеги түшүнүктөр башка тилдерди үйрөнүүдө да маанилүү.

Бул китепте биз маалыматтардын бир нече түрүн талкуулайбыз. Бирок бул бөлүмдө бир белгилүү сандар топтомуна токтолобуз.

Жалпысынан, Python сандарды сан катары эле тааныйт. Бирок сиз сандардын бардыгы бирдей эмес экенин түшүнөсүз да. Жөнөкөй болуш үчүн сиз көргөн ар бир сан бүтүн сан же ондук чекити жок сан болуп, бүтүн сан деп аталат. Бүтүн сандарга 0, 2, 5, 10, 100, 1000, 1032 жана башка ушул сыяктуу сандар кирет. Төмөнкү кодду колдонуп көрүңүз:

```
print(122)
```

Сиздин натыйжаңыз төмөндөгүдөй болушу керек (2-3-сүрөттү караңыз):

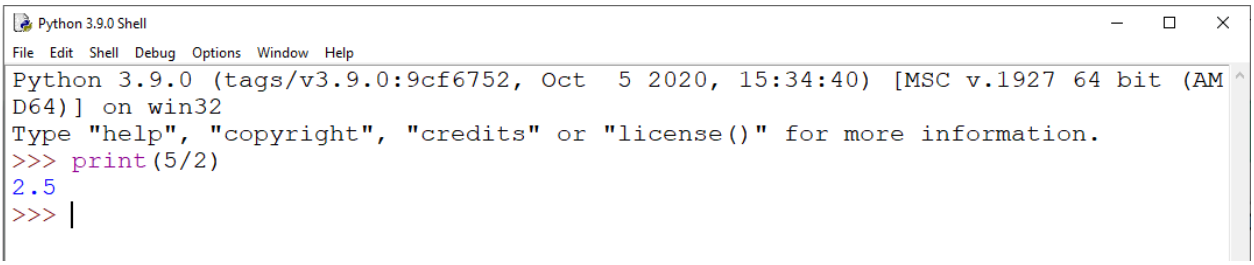


2-3-сүрөт. *Бүтүн санды жазуу*

Мурда көргөнүбүздөй, биз бүтүн сандары колдонуучунун экранына басып чыгаруу менен гана чектелбейбиз. Өзүңүздөр билгендей, эсептөөлөрдү жасай алабыз. Келгиле, төмөнкүлөрдү байкап көрөлү:

```
print(5/2)
```

Бул кодду иштеткенде, 2-4-сүрөттө көрсөтүлгөндөй бир кызыктуу нерсе болот:



Сүрөт 2-4. *Print() функциясынын ичинде эсепти аткаруу*

Эки бүтүн санды көрсөткөнүбүзгө карабастан, берилген сан такыр критерийлерге туура келбейт. Ар бир сан ондук бөлчөккө ээ болгондо, ал маалыматтын бүтүн түрү катары эсептелбейт. Тескерисинче, ал жылышма же жылышма чекиттүү сандар болот.

Биз теңдемелерди бүтүн сандарда кандай аткара алсак, ошондой эле аларды жылышма чекиттүү сандар менен да аткара алабыз. Буга мисал 2-5-сүрөттө келтирилген:

Сүрөт 2-5. *Жылышма маалыматтардын түрү*

Жылышма чекиттүү сандар башка жылышма чекиттүү сандарга кошулганда, натыйжада, дагы жылышма чекиттүү сандар пайда болот. Бул сан бүтүн сан болушу да мүмкүн. Мисалы, мен 2.5 + 2.5 жыйынтыгын сурасам, сиз 5 деп жооп берет элеңиз. Python бул тууралуу эмне деп айтканын карап көрөлү:

2-6-сүрөттө көрүнүп тургандай, Python биз күтпөгөн нерсени жасады. Ал 5.0 деп жылышма чекит санын берди.

```
>>> 2.5+2.5
5.0
>>>
```

Сүрөт 2-6. *Эки жылышма чекиттүү эсепти чыгаруу*

Бул ылайыктуу жыйынтык болсо да, биз өзүбүздүн сандардын түрүн өзгөртүшүбүз керек болгон жагдайга кабылышыбыз мүмкүн. Мисалы, бизде ондуктарды көрсөтпөөнү же сандарды тегеректөөнү каалаган программа болушу мүмкүн. Бул учурда бир вариант катры биздин сандарды өзгөртүү керек болот.

Бул жөнүндө билүүдөн мурун дагы бир нерсени байкап көрөлү. Бүтүн жана жылышма чекиттүү сандар менен эсепти чыгарганда эмне болот? Төмөнкүлөрдү байкап көрүңүз (2-7-сүрөттү караңыз):

```
print(5 - 2.5)
```

Натыйжада:

```
>>> 5-2.5
2.5
>>>
```

Сүрөт 2-7. *Бүтүн сандан жылышма чекиттүү санды кемитүүнүн натыйжасы*

Кайсыл убакта болбосун бүтүн сан же жылышма чекиттүү сан менен эсеп чыгарганда натыйжасы жылышма чекиттүү сан болот.

Сандык маалыматтардын түрлөрүн айландыруу

Эң биринчи үйрөнө турган нерсе - бүтүн санды жылышма чекиттүү санга айландыруу. Мурунку мисалыбызда бүтүн санды жылышма чекиттүү санга айлантуу үчүн жөнөкөй бөлүү ыкмасын колдондук. Ушул эле натыйжага жетүүнүн дагы бир

жолу - бул Python иштөө мейкиндигинде орнотулган float() функцияларынын бирин колдонуу.

Float() колдонууга өтө жөнөкөй, сиз айландыргыңыз келген бүтүн санды кашаанын ичине жайгаштырасыз (). Келгиле, аны жасап көрөлү!

```
float(12)
```

Эгер сиз аны Python Shell кабыгына киргизсеңиз, анда төмөнкүдөй натыйжа болот:

```
>>> float(12)
12.0
>>>
```

Сүрөт 2-8. *Бүтүн санды жылышма чекиттүү сандарга айландыруу*

2-8-сүрөттө көрүнүп тургандай, сиздин натыйжаңыз 12.0 болушу керек (ондук чекитсиз кадимки 12нин ордуна).

Артка кайтуу үчүн жылышма чекиттүү санды бүтүн санга айландырыңыз. Биз Python тилинин дагы бир ыңгайлуу орнотулган функцияларын колдонобуз. Ал - int()!

Int() функциясы float() сыяктуу эле иштейт. Жөн гана кашаанын ичине айландыргыңыз келген санды териңиз. Калганын Python өзү жасайт. Байкап көрүңүз:

```
int(12.0)
```

Бул кайтып келет:

```
>>> int(12.0)
12
>>>
```

Сүрөт 2-9. *Жылышма чекиттүү санды бүтүн санга айландыруу*

2-9-сүрөттө көрсөтүлгөндөй, биз жылышма чекиттүү санды - 12.0дү алып, ондук чекитин алып салуу менен, бүтүн 12 санына айландырдык.

Бизде .0 менен бүтпөгөн жылышма чекиттүү сан болсо эмне болот? Муну жөнөкөй тест аркылуу билип алалы. Аны Python кабыгыңызга киргизиңиз:

```
int(12.6)
```

Enter баскычын басканда, натыйжа 12 болот. Эмне үчүн 13 эмес? Жылышма чекиттүү санды бүтүн санга айландырганда Python ондук чекиттен кийин бардыгын алып салат жана аны көрмөксөн болот. Эгер сиз тегеректөөнү кааласаңыз, анда башка бир функцияны колдонушуңуз керек. Бул жөнүндө кийинчерээк ушул китептен үйрөнөбүз.

Башка маалымат түрлөрүнө айландыра турган дагы көптөгөн маалыматтын түрлөрү бар. Алардын калган бөлүгүн бул китепте чагылдырып беребиз. Азырынча өзүңүзгө кол чаап койсоңуз болот. Сиз арсеналыңызга эки жаңы кубаттуулукту коштуңуз: алар int() жана float() функциялары!

Өзгөрмө деген эмне?

Буга чейин биз маалыматтардын түрлөрүн биринен экинчисине айландыруу үчүн колдонула турган кээ бир негизги математикалык операторлорду жана функцияларды изилдеп чыктык. Бирок, *чыныгы* күчкө ээ болуш үчүн биз *өзгөрмө* деп аталган жашыруун курал жөнүндө билишибиз керек.

Өзгөрмөлөр жөнүндө ой жүгүртүүнүн бир нече жеңил ыкмалары бар. Бул аларды түшүнүүнү жеңилдетет. Алгач аларды сиз бир нерсе сактай турган куту деп эсептесеңиз болот. Биздин учурда аларда сакталган нерсе - бул маалыматтар. Бул маалыматтар сандар болушу мүмкүн, текст болушу мүмкүн, акчалай маани, итиңиздин аты, тексттин абзацы же сиздин жашыруун уяңыздын коопсуздук коду болушу мүмкүн.

Өзгөрмө[1] Python, ошондой эле программалоонун башка тилдеринде көптөгөн функцияларды аткарат. Өзгөрмөнү колдонуунун эң мыкты жолу - бул маалыматты кайра-кайра жазып отурбаш үчүн сактап койгонуңуз. Мисалы, сизде көп колдонгон сандардын узун тизмеси болушу мүмкүн. Ошол узун тизмени ар бир жолу керек болгон сайын терүүнүн ордуна сиз аны өзгөрмөдө сактап, өзгөрмөгө кайрылсаңыз болот.

Өзгөрмөнү колдонуу үчүн ага ат коюп, андан кийин анын маанисин аныктоо гана жетиштүү. Мисалы:

```
a = 8675309
```

Бул код өзгөрмө "a" аталышын жаратат жана андан кийин ага маани берет. Бул учурда анын мааниси 8675309 болуп саналат.

Келгиле, жөнөкөй программа түзүп көрөлү. Ал эки өзгөрмөгө бир аз маалымат берип, андан кийин колдонуучунун экранына басып чыгарат. Биздин биринчи программалоо мисалыбыздагы жаңы Python файлын кантип түзүү керек экенин унутпаңыз. Python Shell кабыгында *File*, андан соң *New File* баскычын басыңыз. Жаңы терезе ачылат. Жаңы терезеге төмөнкү кодду киргизиңиз:

[1] Кыргыз тилдүү окурмандарга, жаш программисттерге программалык командалар жана структуралар менен өзгөрмөлөрдүн айырмасын жана колдонулушун жеткиликтүү түшүндүрүү максатында бул китепте өзгөрмөлөр кыргызча берилген. Чыныгы практикада өзгөрмөлөр латын тамгалары менен белгиленет.

```
a = 500
b = 250
print(a)
print(b)
```

Биринчи *File* басып, андан кийин *Save* баскычын басыңыз. Файлга VariableTest.py деген ат бериңиз. Коддун иш-аракетин көрүү үчүн Run баскычын басып, андан соң Run Module дегенди басыңыз.

Код 2-10-сүрөттө көрсөтүлгөндөй иштейт.

Сүрөт 2-10. *Эки өзгөрмөнүн маанилерин басып чыгаруу*

Ошентип, өзүңүз көрүп тургандай, "a" өзгөрмөсүнө 500, андан кийин "б" өзгөрмөсүнө 250 маанисин койдук. Андан кийин басып чыгаруу функциясын колдонуп, эки өзгөрмөнүн тең маанисин басып чыгардык. Эми көңүл ачалы!

Чынын айтсак, өзгөрмө баалуулуктарын басып чыгаруу кызыксыз. Бирок, басып чыгаруу биздин өзгөрмөлөр менен жасай турган жалгыз эле нерсе эмес. VariableTest.py кодун өзгөртөбүз. Файлга төмөнкүдөй кодду кошуп коюңуз:

```
a = 500
b = 250
print(a)
print(b)
print(a+b)
```

Файлды сактап, андан кийин кайра иштетип, жыйынтыгын көрүңүз. Ал 2-11-сүрөт менен дал келиши керек.

Сүрөт 2-11. *Эки өзгөрмөнү кошуунун жана басып чыгаруунун натыйжаларын көрсөтүү*

Бул жерде биз эки өзгөрмө түзүп, аларга мурдагыдай эле маани бердик. Биз аларды да басып чыгардык. Бирок, бул жолу биз алар менен бир аз математикалык эсеп жүргүзүп, натыйжасын чыгардык. Саптагы код: print(a + b) программага print()

функциясынына кашаанын () ичиндегилердин бардыгын басып чыгарууну айтат. Азыр биз (a) + (b) теңдемесин басып чыгаралы деп жатабыз. Бул 750 болот.

Эскерте кетүүчү нерсе - бул "a" же " b " өзгөрмөсүндөгү маалыматтардын маанилерин өзгөртпөй, аларды колдонуп эсеп жүргүзөт. Өзгөрмө ичиндеги маалыматты өзгөртүү үчүн бизде бир нече варианттар бар. Жаңы файл түзүп, анын атын VariableChange.py деп коёлу. Бул кодду ага киргизиңиз:

```
a=500
b=250
a=a+b
print(a)
```

Натыйжаны көрүү үчүн кодду иштетиңиз (2-12-сүрөттө көрсөтүлгөн):

```
750
>>>
```

Сүрөт - 2-12. *Эки өзгөрмөнүн натыйжасын өзгөрмөгө дайындоо*

Ошентип, бул жерде эмне болду? Алгач биз "a" жана "b" өзгөрмөлөрүнүн маанилерин атадык жана аныктадык. Андан кийин эки өзгөрмө маанисин кошуп, "a" өзгөрмөсүнүн маанисин ушул теңдеменин натыйжасына дал келтирдик. Андан кийин 750 маанисин көрсөтүү үчүн "a" өзгөрмөсүн басып чыгардык.

a = деп тергенде, "a" маанисин барабар (=) белгисинен кийин келген мааниге өзгөртүү керектигин айттык. Кийин Python "a" жана "b" сандарын кошуп, ошол маанини кайра "a" га койду. Барабар белгиси (=) *дайындоо оператору* катары белгилүү.

Эгерде биз "a" өзгөрмөсүнүн маанисин өзгөрткүбүз келбесе, анда биз толугу менен жаңы өзгөрмө түзө алабыз. VariableChange.py кодун төмөнкүлөргө дал келгидей кылып өзгөртөлү:

```
a=500
b=250
c=a+b
print(c)
```

Бул жолу "a" маанисин өзгөртпөстөн, жөн гана жаңы "c" өзгөрмөсүн түзүп, ага "a" + "b" маанисин бердик, андан кийин "c" мазмунун басып чыгардык.

Супер Баатыр генератору - 3000

Эми код менен тажрыйбабыз бар болгондуктан, аны ушул китептин аягына чейин кура турган программанын пайдубалын түптөө үчүн колдонолу. Программа

колдонуучуларга каармандарды (же терс каармандарды) статистика, кокустан пайда болгон ысымдар жана кокустан пайда болгон жөндөмдөр, статистикалар менен толуктоо мүмкүнчүлүгүн берген супер баатыр генератору болот.

Төмөнкү коддордун айрымдары биздин программага текст кошот. Биз аны 3-бөлүмдө кененирээк карайбыз. Азырынча биз бул текстти белги катары гана колдонобуз. Андыктан кодду түшүнүүдө эч кандай көйгөй жаралбашы керек.

Ар бир баатырдын белгилүү бир физикалык жана психикалык касиеттери бар. Эгер сиз буга чейин роль ойногон оюндарды же RPG оюндарын ойногон болсоңуз, анда сиз бул түшүнүк менен таанышсыз. Эгер андай болбосо, кабатыр болбоңуз! Жаныңыздагы адамдарды карап, аларды байкаңыз. Мисалы, сиздин дене тарбия мугалимиңиздин булчуңдары күчтүү жана ал жакшы формада болушу мүмкүн. Демек, ал сиздин информатика мугалимиңизге караганда көбүрөөк *күч-кубатка* ээ жана *чыдамкай* экенин билдирет.

Башка жагынан алганда, сиздин информатика мугалимиңиз сиздин машыктыруучуңузга караганда көбүрөөк акылдуу жана даанышман болсо керек. Айтайын дегеним, анын *акыл-эси* жана *даашымандыгы* бир топ жогору. Алгач ушул төрт сапат же статистика менен баштайлы. Кийинчерээк дагы көп нерсени кошсок болот.

Ар биринин маанисин аныктоо үчүн төмөндөн жогоруга карай диапазон беришибиз керек. Азырынча 0–20 диапазонун колдонсок болот. 0 төмөн, 20 бийик. Демек, эгер биз күчтү талкууласак, анда 0 өтө алсыз, ал эми 20 Геракл болмок. Демек, орто эсеп катары 10ду алабыз.

Ошо сыяктуу эле, акыл үчүн, 0 акылсыз болот деп айтсак, анда 20 Альберт Эйнштейн десек болот. 10 диапазонуна түшкөн адам орточо акылдуу деп эсептелет.

Эми оюнчуларга өздөрүнүн атрибуттук баалуулуктарын коюуга уруксат берсек болот. Бирок андан кийин ар бир адам баалулугун 20га чейин жеткирип, дүйнөдөгү эң күчтүү, эң акылдуу адамга айланарын билебиз. Бул нерсе сиз экөөбүздү толук аныктаганы менен, башка жөнөкөй адамдар ошол жогорку талаптарга жооп бербейт.

Тескерисинче, биз алардын атрибуттарынын маани-маңызын туш келди берүүбүз керек. Python функциясын колдонуп, божомол сандарды оңой эле жаратууга мүмкүн. Бул сиз болжолдогон – random().

random() функциясын колдонуу башка функцияларга караганда бир аз айырмаланып турат. Аны колдонуу үчүн алгач аны Python иштөө мейкиндигине импорттошубуз керек. Биз муну коддун жөнөкөй сабы менен жасайбыз:

```
import random
```

random() функциясы башка функциялардай эле иштейт. Анткени анын кашаасынын ичине параметрлерди берсе болот. RandomGenerator.py деп аталган жаңы Python файлын түзүп, төмөнкү кодду киргизиңиз:

```
import random
күч = random.randint(1,20)
print(күч)
```

Бул коддо биз алгач random() модулун импорттойбуз. Андан кийин "күч" деген өзгөрмө түзөбүз. Өзгөрмөлөр жөнүндө айта кетүүчү бир маанилүү нерсе бар. Программалоо дүйнөсүндө *ат коюу конвенциясы* деген түшүнүк бар. Демек, өзгөрмөлөрдү атоодо белгилүү бир "эрежелер" сакталышы керек. Өзгөрмө менен сиз ар дайым өзүңүзгө же болочок программистке өзгөрмө ичинде кандай маалыматтардын түрү сакталарын билгидей кылып ат берүүнү каалайсыз. Мисалы, “а” өзгөрмөсүнүн аталышы бизге көп маалымат бербейт. Аны "күч" деп атоо, анын ичиндеги маалыматтар эмне үчүн керек экендигин аныктайт.

Эгерде сиздин өзгөрмө атыңызда бирден ашык сөз болсо, анда аларды бир сөз катары сактап, экинчи сөздүн биринчи тамгасын баш тамга менен жазыңыз. Мисалы, эгер биздин өзгөрмөбүз "Баатыр Күч" болсо, анда биз аны “баатырКүч” деп атамакпыз. Эгер ал "Баатыр Күч Статистикасы" болсо, биз “баатырКүчСтатистикасы” деп колдонмокпуз.

Экинчи эреже - аны ар дайым мүмкүн болушунча кыска жана жөнөкөй кылып сактоо. Эске салсак, өзгөрмөлөр кодду терүүдө убакытты үнөмдөө үчүн колдонулат. Андыктан узак аталыштар максатты жокко чыгарат.

Кодго кайтып келиңиз ....

Биздин “күч” аталышындагы өзгөрүлмөнү жараткандан кийин ага маани беришибиз керек. Коддун кийинки бөлүгү random() модулун чакырат жана randint атрибутун колдонот. Randint random() курамына кирет жана кокустук санды гана жаратпастан, туш келди бүтүн санды түзүүнү айтат. Кашаанын ичине киргизген мааниси - бул кокустук санынын диапазону. Эсиңизде болсун, биздин статистика 1ден 20га чейинки аралыкта болушун каалайбыз. Андыктан биз киргизген маани (1 - 20) болот.

Кодду RandomGenerator.py сайтынан бир нече жолу иштетип көрүңүз. Ар бир жолу кокустук санды алышыңыз керек:

Эми кокустук сандар генератору иштеп, аны кантип колдонууну түшүнгөндөн кийин дагы бир нече статистиканы кошуп көрөлү:

```
import random
күч = random.randint(1,20)
интеллект = random.randint(1,20)
чыдамдуулук = random.randint(1,20)
акылмандык = random.randint(1,20)
```

Андан кийин биз бул баалуулуктарды текшерүү үчүн экранга басып чыгарышыбыз керек. Ал үчүн бир нече текстти белги катары колдонуп, андан кийин ар бир өзгөрмөнүн маанисин тиешелүү белгиден кийин басып чыгарабыз.

Бул кодду өзгөрмөлөрүңүздөн кийин кошуңуз:

```
print("Сиздин каармаңыздын статистикасы: ")
print("Күч: ", күч)
print("Интеллект: ", интеллект)
print("Чыдамдуулук: ", чыдамдуулук)
print("Акылмандык: ", акылмандык)
```

Бул жерде print() функциясынын башкача колдонулушуна туш болобуз. Буга чейин биз сандарды жана өзгөрмөлөрдү басып чыгаруу үчүн print() колдонуп келгенбиз. Ошентсе да, азыр ***сап*** катары белгилүү болгон жаңы маалымат түрүн колдонуп жатабыз. ***Сап*** - бул жөн гана текст. Ал каалаган тамгаларды, өзгөчө белгилерди (!, @, #, $,%, ^, &, *, -, +, = ж.б.) жана каалаган санды камтышы мүмкүн. Бирок, ал текст катары каралышы үчүн тырмакчалардын "" арасына коюлушу керек. Эгер андай болбосо, Python аны башка нерсе катары чечмелейт. Азыр буга көп кабатыр болбоңуз. Аны 3-бөлүмдө кененирээк карап чыгабыз. Азырынча коддун бир сабын карап көрөлү:

```
print("Сиздин каармаңыздын статистикасы: ")
```

Бул код компьютерге түзмө-түз экранга "Сиздин каармаңыздын статистикасы:" деп басып чыгарууну айтат.

Кийинки көрсөтмө бир аз башкача:

```
print("Күч:",күч)
```

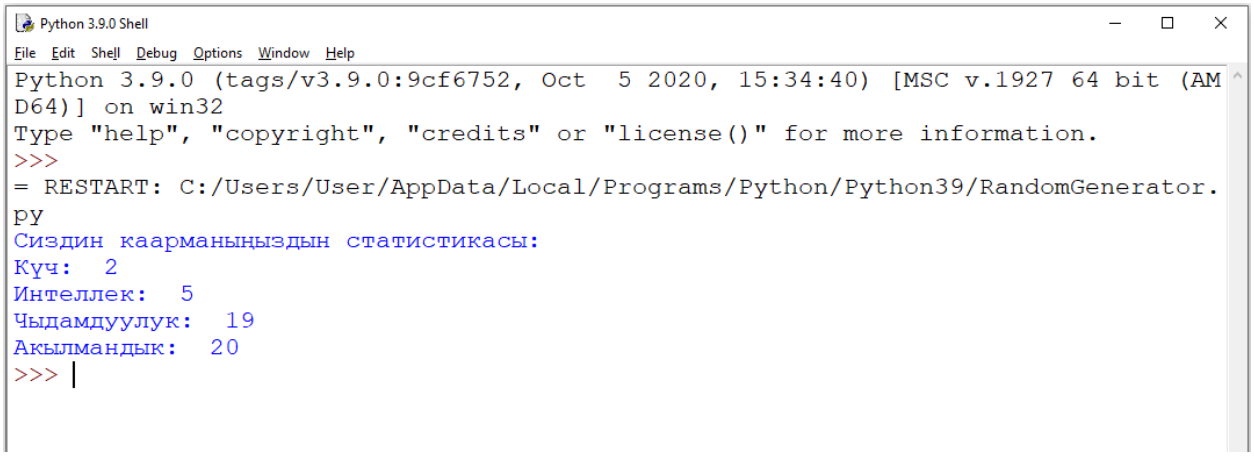
Бул print() функциясы эки нерсени аткарат. Биринчиден, текстти - “*Күч:*”-кашаанын ортосуна басып чыгарыңыз дейт. Андан кийин print() функциясы боюнча дагы нускамалар бар экендигин билдирген үтүрдү (,) кошобуз. Андан кийин биз мазмунун басып чыгарууну каалаган өзгөрмөнүн атын киргизебиз. Бул учурда өзгөрмө - *күч*. Өзгөрмө тырмакчага алынбагандыгын эске алыңыз. Эгер тырмакчада болсо, анда ал "*күч* " деген сөздү күч деп аталган өзгөрмөнүн мазмунунун каршысына басып чыгармак.

Эми RandomGenerator.py файлыңыз мындай болушу керек:

```
import random
күч = random.randint(1,20)
интеллект = random.randint(1,20)
чыдамдуулук = random.randint(1,20)
акылмандык = random.randint(1,20)
print("Сиздин каармаңыздын статистикасы: ")
print("Күч: ", күч)
print("Интеллект: ", интеллект)
```

```
print("Чыдамдуулук: ", чыдамдуулук)
print("Акылмандык: ", акылмандык)
```

Кодду бир нече жолу иштетип көрүңүз. Биздин программа кокустан пайда болгон сандарды жаратарын унутпаңыз. Ошондуктан кодду аткарган сайын натыйжалар ар башкача болот. 2-13-сүрөттө анын кандай болушу керектиги жөнүндө мисал келтирилген:



Сүрөт 2-13. *Кокустук статистикасын түзүү*

Куттуктайбыз! Сиз Супер Баатыр генератору - 3000 тиркемесинин баштапкы колдонмосун түздүңүз!

Бул эпизоддо

Бул кызыктуу эпизоддо биз көп нерселерди камтыдык. Сиз жаш шакирт болуп баштагансыз. Бирок сиздин күч-кубатыңыз барган сайын туруктуу өсүп жатат! Жакында сиз Керемет Баладан Керемет Жигитке айланасыз? Билбейм, атыңыздын үстүнөн дагы иштейбиз. Баарынан маанилүүсү - программалоо боюнча алгачкы кадамдарды супер баатырдай жасаганыңыз болду.

Алдыда кандай коркунучтар бар? Кийинки бөлүмдө биз текст менен иштөөнү карап чыгып, Супер Баатыр генератору - 3000 тиркемебиздин үстүнөн иштөөнү улантабыз. Ошондой эле, биз өз ишибизди документтештирип, комментарий бере баштайбыз, эгерде сиз качандыр бир кезде улуу адамдардын катарына кирем деп үмүттөнсөңүз, анда сизге милдеттүү түрдө программалоо практикасы керек!

Алга жылуудан мурун ушул бөлүктөн эмнелерди үйрөнгөнүбүздү карап көрөлү:

- Маалымат түрлөрү: Маалымат түрлөрү бардык программалоо тилдеринде бар жана программа иштеп жаткан маалыматтардын түрүн аныктоого жардам берет. *Integer же int* - бул бүтүн сандар үчүн маалымат түрү жана *float* - бул жылышма чекиттүү сандар же ондук сандар үчүн маалымат түрү.

- Операторлордун иерархиясы: Теңдемелерди иштетүүдө кээ бир операторлор артыкчылыкка ээ же башка операторлордон биринчи иштелет.
- Операторлор: Жалпы операторлорго + (кошуу), - (кемитүү), * (көбөйтүү) жана / (бөлүү) кирет.
- Дайындоо оператору: Барабар белгиси (=), дайындоо оператору катары белгилүү. Бул сизге өзгөрмөгө маани берүүгө мүмкүнчүлүк берет.
- Иштетүү тартиби: Математикалык амалдарды аткаруу тартиби операциялардын тартиби деп аталат. Кайсы математикалык теңдемеде биринчи аткарылышы керек амалды билиш үчүн ал бөлүгүн кашаанын ичине киргизүү менен көрсөтсөк болот. Мисалы: (1 + 1) * 10 көбөйтүүнүн кошууга караганда артыкчылыкка ээ экендигине карабастан, 1 + 1 көбөйтүүдөн мурун аткарылышын камсыз кылат.
- Маалымат түрлөрүн которуу: int() жана float() бизге жылышма чекиттүү санды бүтүнгө, ал эми бүтүн санды жылышма чекиттүү санга которууга мүмкүнчүлүк берет.
- Өзгөрмөлөр: Өзгөрмөлөр - бул маалыматтарды сактоо бирдиги. Ошондой эле, аларды маалыматтардын жайгашкан жерин көрсөткөн белгилер деп эсептесеңиз болот. Бирок аларды маалыматтын бир бөлүгү камтылган кутуча деп эсептөө оңоюраак болушу мүмкүн. Дайындоо операторунун жардамы менен аларга ат коюу жана аларга маани берүү аркылуу өзгөрмө түзөбүз. Мисалы: a = 12.
- Ат берүү конвенциясы. Конвенциялар - коддоо эрежелерин жеңилдетүүгө жардам берет. Бул сиз үчүн жана келечектеги программисттер үчүн сиздин кодуңуз болуп саналат. Аларды "мыкты тажрыйба" деп эсептеңиз. Мисалы, өзгөрмөнү атоодо ар дайым биринчи сөз үчүн кичине тамгаларды, андан кийинки сөздөр үчүн баш тамгаларды колдонуңуз. Бир нече сөздү бир сөзгө топтоңуз. Мисалы: socialSecurity туура. Social Security туура эмес жана SyntaxError алып келет. Ошондой эле, алардагы маалыматтар эмне үчүн колдонулгандыгын сүрөттөгөн кыска аталыштар менен өзгөрмөлөр менен атоого аракет кылыңыз.
- random() жана randint(): random() - кокустук сандарды жаратууга мүмкүнчүлүк берген модуль. Аны import random коду аркылуу иштөө мейкиндигиңизге импорттошуңуз керек.
- Берилген сандар диапазону менен бүтүн санды кокустан түзүү үчүн random.randint(1,20) же random.randint(5,100) деп киргизиңиз. Ошондой эле эгер сиз сандарды туш келди түрдө 1ден 20га чейин же 5тен 100гө чейин түзгүңүз келсе да, ошондой эле кыласыз.
- Эгер сиз 0дон 20га чейинки сандарды жаратууну кааласаңыз, анда аны коддо көрсөтүшүңүз керек. Мисалы: random.randint(0,20).

3-БӨЛҮМ

НЕРСЕЛЕРДИ САПКА ТИЗҮҮ

Супер баатыр, кайрадан кош келиңиз! Сиз супер баатырлар жана терс каармандар, айрыкча, шумпайлар жөнүндө билишиңиз керек. Анткени алар улам сиздин тынчыңызды алып турушат. Бактыга жараша, бул бөлүмдө кеп сиздин жөндөмдүүлүктөрүңүздү жогорулатууга жана текстке байланыштуу бардык маселелерди чечүүгө жаңы супер күч берүү тууралуу болмокчу!

Биз текст менен иштөөнүн жана башкаруунун негиздерин, анын ичинде тексттин жалпы функцияларын, ошондой эле тексттик маалыматтардын түрү жөнүндө маалыматтарды изилдейбиз. Ошондой эле текстти форматтоону жана тексттин ар кандай маалымат түрлөрүнө өткөрүлүшүн камтыйбыз. Акыр-аягы, биз сизди жана болочок программисттерди бир топ баш оорудан сактап калуу үчүн документтештирүүнүн маанилүүлүгүн жана кодду кантип комментарийлөөнү карап чыгабыз.

Андыктан ачык жашыл түстөгү шым кийип, кызгылт сары Дей-гло маскасын кийип алыңыз. Wonder Boy (же Girl) логотибиндеги кетчуп такты тазалап, манжаларыңызды сулуу жана ийкемдүү кылыңыз.

Код жазууга даярданыңыз!

Пикирлериңизди сыртка калтырыңыз

Программалоо тилине сүңгүп кирүүдөн мурун биз айткан, бирок азырынча оолактап жүргөн темага токтолуп кетүү маанилүү. Туура ат коюу конвенциялары сыяктуу эле, кодуңузга комментарий жазуу же документтештирүү чеберчилиги - мыкты программист дайыма колдонуп жүргөн тажрыйбалардын бири. Бул үйдөн чыкканга чейин чапаныңызды үтүктөгөн сыяктуу. Албетте, муну өткөрүп жиберсе болот. Бирок анда сизде душманыңызга шылдың болуу коркунучу жаралат.

Кодуңузга комментарий берүүгө бир нече себеп бар. Биринчиден, программисттер көбүнчө кодду биринчи программалаганга караганда аны кийинчерээк дагы карап турушат. Ал күндөр, жумалар, айлар, алтургай, жылдар өткөндө болушу мүмкүн. Миңдеген коддорду кайрадан карап чыгуу тозок эле болот. Айрыкча, ар бир бөлүмдүн эмне кыларын аныкташыңыз керек болсо, бир топ ишти кайра карап чыгууга туура келет. Эгер сиздин бөлүмдөрүңүз белгиленип, кыскача сүрөттөлсө, анда кийинчерээк сапта жаңыртууга туура келген көйгөйлүү жерлерди же бөлүктөрдү табуу оңой болот.

Кодду документтештирүүнүн дагы бир себеби, башка программисттер аны кайсы бир убакта кайра карап чыгышы керек болушу мүмкүн. Мындай

программисттер сиздин кожоюнуңуз, жумуштагы кесиптештериңиз, же болбосо, жумушка орношо электе жазганыңызга өзгөртүүлөрдү киргизиши керек болгон болочоктогу программист болушу мүмкүн.

Акыркысы, бир программанын кодун экинчи программада кайталап колдоно турган учурлар болот. Биз аны натыйжалуулук деп атайбыз (эгер, албетте, буга сиздин компания уруксат берсе!). Бул учурларда, эгерде сиз өз ишиңизге комментарий берген болсоңуз, аны документтештирген болсоңуз, анда сиз издеп жаткан коддун бир бөлүгүн табуу тезирээк ишке ашат.

Программисттерде комментарий калтыруунун ар кандай жолдору бар. Ар бир адамдын өзүнүн стили болот. Айрым компаниялар кодуңузду өзгөчө, формattалган стилде документтештирүүнү талап кылышы мүмкүн. Ал эми башкалары сизге калтырат.

Дагы бир нерсе: комментарий сиздин кодуңузга жазылып жатканда, котормочу же компилятор аларды этибарга ала бербейт. Демек, эгер сиз аларды туура эмес синтаксисти колдонуп киргизбесеңиз, алар сиздин кодуңузга таптакыр таасир этпейт.

Комментарий берүү үчүн сиз хештег же # белгисин колдоносуз. Ошол саптын калган бөлүгүндө # пайда болгондон кийин ал комментарий деп эсептелет.

Бул жерде бир комментарийге мисал келтирели:

 # Коддун бул блогу баатырдын статистикасын туш келди эсептеп чыгат

Эгер сиз ушул кодду иштетсеңиз, анда дагы эч нерсе болбойт. Анткени Python комментарийлерге көңүл бурбайт. Алар компьютерлердин керектөөсү үчүн эмес, адамдар жана программисттер үчүн гана жазылган.

Келгиле, комментарий коддун жанында кандайча көрүнөрүн карап көрөлү. Биздин мурдагы бөлүмдөн үйрөнгөн RandomGenerator.py эсиңизде барбы? Аны ачып, ага төмөнкү текстти кошуңуз:

```
 import random
# Коддун бул блогу баатырдын статистикасын туш келди эсептеп чыгат
  күч = random.randint(1,20)
  интеллект = random.randint(1,20)
  чыдамдуулук = random.randint(1,20)
  акылмандык = random.randint(1,20)
  print("Сиздин каармаңыздын статистикасы: ")
  print("Күч: ", күч)
  print("Интеллект: ", интеллект)
  print("Чыдамдуулук: ", чыдамдуулук)
  print("Акылмандык: ", акылмандык)
```

Көрүнүп тургандай, бул коддун так ушул бөлүгү түшүнүүнү жеңилдетет. Коддун үзүндүсүнүн аягына дагы бир комментарий кошуп, мындан да түшүнүктүү кылышыбыз мүмкүн:

```
import random

# Коддун бул блогу баатырдын статистикасын туш келди эсептеп чыгат.

import random
күч = random.randint(1,20)
интеллект = random.randint(1,20)
чыдамдуулук = random.randint(1,20)
акылмандык = random.randint(1,20)

# Туш келди сандарды жаратуу кодунун аягы

print("Сиздин каармаңыздын статистикасы: ")
print("Күч: ", күч)
print("Интеллект: ", интеллект)
print("Чыдамдуулук: ", чыдамдуулук)
print("Акылмандык: ", акылмандык)
```

Бул жерде идея башкача иш кылган коддун ар бир бөлүмүнүн аяктоо жана баштоо чекитин белгилөө болуп саналат. Сиз элестеткендей, бул документтерди алып кетүү оңой, бирок анын өз артыкчылыктары бар. Канча ирет комментарий бересиз, ал сиздин жумушуңуз. Бирок, адатта, документтештирбей койгондон көрө, документтештиргениңиз жакшы.

Блоктук комментарийлер

Кадимки комментарийлерден тышкары, комментарийлердин *блоктук комментарийлер* деп аталган формасы дагы бар. Комментарийлердин бул түрү коддун бир эле бөлүгүн түшүндүрүп берүү үчүн бир саптан ашык талап кылынганда колдонулат. Ошондой эле, сиз кодду жазган күн, аны ким жазган жана башка ушул сыяктуу нерселерди документтештирүү керек болгондо колдонсо болот. Блоктук комментарийин көрсөткөн төмөнкү кодду кара:

```
# Random функциясын импорттоо
import random

# Бул код китептин автору тарабынан жазылган
# Өспүрүмдөр үчүн басылып чыгарылган Python китеби
# Коддун бул блогу баатырдын статистикасын туш келди эсептеп чыгат.
```

```
күч = random.randint(1,20)
интеллект = random.randint(1,20)
чыдамдуулук = random.randint(1,20)
акылмандык = random.randint(1,20)
# Туш келди сандарды жаратуу кодунун аягы
# Каармандын статистикасын басып чыгарат
print("Сиздин каарманыңыздын статистикасы: ")
print("Күч: ", күч)
print("Интеллект: ", интеллект)
print("Чыдамдуулук: ", чыдамдуулук)
print("Акылмандык: ", акылмандык)
```

Сиз көрүп тургандай, блоктук комментарий үчүн ар бир саптын башына жазыла турган белгини (#) кошуп коюу жетиштүү.

Саптагы комментарий

Комментарий берүүнүн дагы бир жолу *саптагы комментарий* деп аталат. Бул сиздин кодуңуз менен бир эле сапка комментарий калтыргандыгыңызды билдирет. Алар комментарий берүүнүн башка түрлөрү сыяктуу кеңири тараган эмес. Бирок, коддун белгилүү бир сабынын аткарган иштерин документтештирүү керек болсо, пайдалуу болушу мүмкүн. Мисалы, RandomGenerator.py файлында биз кокустуктарды импорттой баштайбыз. Сиздин кодуңузду карап жаткан программистке коддун ушул сабы айдан ачык көрүнүп турса да, биз аны түшүндүрүп берүү үчүн ичине комментарий калтырып койсок болот.

Ал мындайча көрүнөт:

```
import random      # random модулун импорттоо
```

Жалпы эреже катары, бир саптагы коддун эмне кыларын түшүндүрүп берүүнү туура көрбөсөңүз, ички комментарийлерди колдонуудан оолак болуңуз.

Комментарий берүүнүн башка жолдору

Кодуңузга комментарий калтыруунун акыркы жолу - каталарды табуу. Бул адаттан тышкары угулушу мүмкүн. Бирок, чындыгында, иш жүзүндө өтө ыңгайлуу болот. Кээде сиздин кодуңуз ката кетириши мүмкүн, анда сиз коддун туура эмес болгон бөлүгүн жокко чыгарышыңыз керек. Кодду толугу менен жок кылуунун ордуна, ар дайым бөлүмдөргө комментарий берсеңиз болот. Python # белгисин көргөндө ошол саптан кийинки белгилерге көңүл бурбай тургандыгын унутпаңыз.

Эгерде биз төмөнкү кодду комментарийлей турган болсок, анда ал мурункуга караганда башкача иштейт:

```
import random
күч = random.randint(1,20)
интеллект = random.randint(1,20)
чыдамдуулук = random.randint(1,20)
акылмандык = random.randint(1,20)
print("Сиздин каармаңыздын статистикасы: ")
#print("Күч: ", күч)
#print("Интеллект: ", интеллект)
print("Чыдамдуулук: ", чыдамдуулук)
print("Акылмандык: ", акылмандык)
```

Бул код менен биз каармандын күчүн же интеллектин экранга басылып чыккандыгын көрө албайбыз. Анткени биз коддун ошол бөлүгүн түшүндүрүп бергенбиз. Анын ордуна чыдамдуулук жана акылмандык гана көрсөтүлөт.

Программаны кадимки абалына кайтаруу үчүн биз # белгисин алып салабыз. Сиздин кодуңузга комментарийлерди кошуп, кодуңуздун бөлүктөрүн комментарийлеп, анын программаңызга кандай таасир этерин карап көрсөңүз болот.

Телефону жок билдирүүлөр

Эми биздин кодду документтештирүү үчүн комментарийлердин маанилүүлүгүн жана аларды кантип түзүү керектигин түшүнгөндөн кийин маалыматтын кийинки түрү - *саптар* менен иштөөгө өтсөк болот.

Сап маалыматтарынын түрү сиз тере ала турган ар кандай белгилерден турат. Болгону ал тырмакчанын "" ичинде болсо болду. Чындыгында, бул - каалаган тамга, сан же атайын белги. Бул бир тамга, сүйлөм же тамгалардын, сандардын жана атайын белгилердин аралашмасы болушу мүмкүн.

Келгиле, LearningText.py аттуу жаңы файл түзөлү. Ага төмөнкү текстти кошуңуз:

```
# Текстти ушундай жол менен басып чыгарасыз
print("Holy smokes, it's the Grill Master!")
print ('Holy smokes, it's the Grill Master!')
```

Эгер сиз коддун экинчи версиясын иштетсеңиз, анда жараксыз синтаксистик ката - Invalid SyntaxError пайда болот. Мунун эмне үчүн болуп жаткандыгын түшүнө аласызбы? Келгиле, кодду дагы бир жолу карап чыгалы. Print() функциясы

тырмакчалардын ичинде камтылган нерселерди басып чыгарарын билебиз. Биздин сүйлөм бир тырмакча менен аяктап жана башталганы менен, жакшылап карасаңыз, үчүнчү тырмакчаны көрө аласыз. Ал "it's" деген сөздөн турат.

Басып чыгаруу функциясында бир тырмакчаны колдонууда этият болушубуз керек. Анткени Python колдонулган тырмакча менен кыскартууда колдонулган апострофту айырмалай албайт. *Holy* сөзүнөн мурун биринчи тырмакчаны көргөндө параметрди баштайт. Андан кийин сөздө апостроф кездешкенде котормочу башы маң болуп, аны акыркы сөз катары эсептейт. Акырында ал үчүнчү тырмакчага туш болуп, ката көрсөтөт.

Маселенин мындай түрүнөн качуунун бир нече жолу бар. Эреже боюнча биринчиден, ар дайым кош тырмакчаларды колдонуу керек. Экинчиден, бир тырмакча керек болгон же колдонууну каалаган учурларда, *чыгуу – escape-* белгиси көйгөйүңүздү чечиши мүмкүн.

Чыгуу баскычы, негизинен, кайырма слэш (\) белгиси болуп саналат. Ал бир тырмакчаны кадимки элемент катары кароону сунуш кылат. Аны колдонуу үчүн аны Python жөнөкөй текст катары кабыл алыш керек болгон белгиден мурун кошосуз. Сиз аны мындай коддосоңуз болот:

```
# Текстти ушундай жол менен басып чыгарасыз
Print('Holy smokes, it\'s the Grill Master!') # \ белгисинин колдонулушуна көңүл буруңуз
```

Эми сиз кодду иштетсеңиз, анда 3-1-сүрөттө көрсөтүлгөн натыйжага ээ болосуз:

```
Holy smokes, it's the Grill Master!
>>>
```

3-1 сүрөт. *Басып чыгаруу билдирүүлөрүн форматтоо үчүн escape белгисин колдонуу*

Жөнөкөйлүк үчүн азыр коддогу кош тырмакчаларды колдонууга кайтып келели. Алга жана ошол өзгөрүүнү жасаңыз. Мен ушул жерде күтүп отурам.

Бүттүңүзбү? Абдан жакшы. Дагы бир нече сап текст кошуп коёлу:

```
# Текстти ушундай жол менен басып чыгарасыз
print("Holy smokes, it's the Grill Master!")
print("Анын эти аябай жакшы, туруштук бере албайсын!")
print("Тез, керемет бала! Керемет нанды алып, мага сендвич даярда!")
print("Генийдин сөзүнөн:'Адам нан жана суу менен гана жашай албайт!'")
```

Бул коддун максаты эки тараптуу. Биринчиден, ал тексттин бир нече саптарын кантип басып чыгарууну көрсөтөт. Экинчиден, эки жана бир

тырмакчаларын бири-биринин ордуна колдонсо болот. Туура грамматиканы колдонгондо адамдын цитатасын бир эле тырмакча аркылуу келтирсе болот.

Бул учурда бир тырмакчадан качуунун кажети жок. Себеби, биз print() функциясын кош тырмакча менен *баштадык*. Биз бир тырмакча менен баштаганда гана функцияны аягына чыгарууну көздөбөгөн башка бир тырмакчадан качып кетүүдөн чочулашыбыз керек.

Саптар жана өзгөрүлмөлөр менен иштөө

Биз сандарды сактаган сыяктуу эле, саптар да өзгөрмөлөрдө сакталышы мүмкүн. Ыкмасы санды сактоого окшош, бирок бир аз башкача:

```
аты = "Гриллдин мастери"
print(аты)
```

Алгач биз "name" деп аталган өзгөрүлмө түзүп, андан кийин ага текст киргизебиз. Белгилей кетчү нерсе, бир сандан айырмаланып, биз маанини тырмакчага алдык. Бул биз өзгөрүлмөбүзгө сап кошуп жатабыз дегенди билдирет. Андан кийин print() функциясын колдонуп, колдонуучунун экранына өзгөрүлмөбүздү басып чыгарабыз.

Бул жерде бардыгы кызыктуу боло баштайт. Жаңы файл түзүп, төмөнкү кодду байкап көрүңүз:

```
жашы = "42"
окуунуБүтүргөн = 27
print(жашы + окуунуБүтүргөн)
```

Эгер сиз ушул кодду колдонуп көрүп, иштей турган болсоңуз, анда ката жөнүндө билдирүү аласыз. Эмне үчүн? Мунун себеби жөнөкөй: " *жашы* " деп аталган өзгөрүлмөбүздү жазганда, ага "42" маанисин бердик. Бирок, биз маанини тырмакчага камтыгандыктан, Python ал маалыматтарды сап түрүндөгү маалымат катары чечмелеген. Ал ортодо "*окуунуБүтүргөн*" өзгөрүлмөсүнө сандык маалыматтардын түрү берилген. Математиканы эки өзгөрүлмө менен жасоого аракет кылганда, натыйжа чыкпай койду. Анткени сиз сапты эсептей жасай албайсыз.

Кызыгы, сиз айрым математика операторлорун саптарда колдоно аласыз. Python жана башка тилдерде бириктирүү деп аталган нерсе бар. Бир сапты экинчисине кошкондо же аларды бириктиргенде бириктирүү пайда болот. Биз саптарды бириктирүүдө кошуу операторун колдонобуз. Бул жерде ал код:

```
print("Керемет" + "Бала")
```

Коддун ушул бөлүгүн текшергенде, натыйжаңыз төмөнкүчө болот:

```
КереметБала
```

Эгер + операторун саптар камтылган эки өзгөрүлмө үчүн колдонсоңуз, анда ушундай эле натыйжа болот:

```
аты = "Керемет"
атасынынАты = "Бала"
print(аты + атасынынАты)

Натыйжасы:
КереметБала
```

Маанилүү эскертүү: эгерде сиз эки сапты бириктиргиңиз келсе, анда алардын ортосунда бошток колдонууну ойлонуп көрүшүңүз мүмкүн. Муну сиз кошуп жаткан биринчи саптын аягына бош орун кошуу менен жасоого болот:

```
print("Керемет  " + "Бала")
```

же экинчи сапка чейин бошток кошуу менен, сиз кошулуп жатасыз:

```
print("Керемет" + "  Бала")
```

Албетте, бош орунду камтыган үчүнчү сапты киргизүүгө эч нерсе тоскоол боло албайт:

```
print("Керемет" + " " + "Бала")
```

Бул иштейт, анткени бош орун да сап же символ болуп эсептелет.

Сиз саптарда колдоно турган дагы бир математикалык оператор бул көбөйтүү (*) оператору же текст менен иштөөдө айтылгандай, *сапты кайталоо оператору*. Бул код менен иштеп көрүңүз:

```
print("КереметБала" * 20)
```

Натыйжа 3-2-сүрөттө көрсөтүлгөндөй болот:

```
>>> print("КереметБала" * 20)
КереметБалаКереметБалаКереметБалаКереметБалаКереметБалаКереметБалаКереметБалаКер
еметБалаКереметБалаКереметБалаКереметБалаКереметБалаКереметБалаКереметБалаКереме
тБалаКереметБалаКереметБалаКереметБалаКереметБалаКереметБала
>>>
```

3-2-сүрөт. *Сапты көчүрүүнүн натыйжасынын мисалдары*

Эгерде сиз ушул код менен файл түзсөңүз, анда 3-3-сүрөттө көрүнүп тургандай, сапты камтыган өзгөрүлмөдө сап кайталанышын жасасаңыз, анда сиз окшош натыйжага ээ болосуз:

```
дос="КереметБала"
```

```
print("Грилл мастеринин барбекюсун буздуң!")
print("Эл сенин атыңды айтып жатат!")
print(дос *20)
```

```
Грилл мастеринин барбекюсун буздуңуз!
Эл сенин атыңды айтып жатат!
КереметБалаКереметБалаКереметБалаКереметБалаКереметБалаКереметБалаКереметБалаКер
еметБалаКереметБалаКереметБалаКереметБалаКереметБалаКереметБалаКереметБалаКереме
тБалаКереметБалаКереметБалаКереметБалаКереметБалаКеремет
>>>
```

3-3-сүрөт. *Өзгөрмө үчүн сап кайталанышын жасоо*

Узунураак саптар

Бир сап менен же бир сөз менен гана чектелип калса, саптар өтө күчтүү болмок эмес. Жогоруда айтылгандай, саптар бир бүтүн сүйлөм болушу мүмкүн жана биз аларды бир сөз сыяктуу эле, өзгөрүлмө менен белгилейбиз:

```
тамаша = "Эмне себептен Жөргөмүш киши эжеси Мей менен таарынышты?"
себеби = "Ал интернетке өтө көп убакыт коротуп жаткан."
print(тамаша)
print(себеби)
```

Көп саптуу саптар

Кээде текстти кандайдыр бир жол менен терип же аны ырдын тексти сыяктуу түзгүңүз келиши мүмкүн. Бул учурда сиз көп саптуу сап түзө аласыз. Бул үчүн үч кош тырмакчаны (“”” “””) же үч жалгыз тырмакчаны (''') колдонуу жетиштүү. Бул жерде коддо кандай көрүнүш болору жөнүндө бир мисал келтирилген. Жаңы файл түзүп, байкап көрүңүз. Сиз 3-4-сүрөттө көрсөтүлгөн натыйжаны көрүшүңүз керек:

```
print("""Менин атым Грилл Мастери
жана менин табитим бар
даамын татып көрүү үчүн.
Эң сонун болгон!""")
```

```
Менин атым Грилл Мастери
 жана менин табитим бар
даамын татып көрүү үчүн.
Эң сонун болгон!
>>>
```

3-4-сүрөт. *Көп саптуу саптык басылма операторлорун түзүү*

Ушул эле натыйжага үч тырмакчаны колдонуп жетишүүгө болот:

```
print('''Менин атым Грилл Мастери
 жана менин табитим бар
даамын татып көрүү үчүн.
Эң сонун болгон!''')
```

Саптарды форматтоо

Көп саптуу саптарды колдонсоңуз, текстти жана саптарды форматтоого жардам берет. Балким, сизде андан башка да жакшы ыкмалар бар. Балким, сиз кызды же баланы ыраазычылык балына адаттан башкача чакырып, аларды таң калтыргыңыз келиши мүмкүн же жаңы темадагы ырыңыздын текстинин үстүнөн иштеп жаткандырсыз. Кандай болгон күндө да, ылайыктуу *сап форматтоочусу* жок текстиңиз жупуну жана кызыксыз болот.

Ал эми эргибөө - баатырга керек болгон акыркы нерсе.

Биз escape белгисин (\) мурун талкуулаганбыз. Python апострофту кандай кабыл аларын print() функциясынын аягында салыштырып, аны кантип колдонуу керек экенин билдик. Чындыгында, бир нече ар кандай escape белгилери бар, алардын ар бири текстти белгилүү бир форматта өзгөртө алат. Алар төмөнкүлөр:

- \ - Көп саптуу саптарда жаңы сап түзүүгө мүмкүнчүлүк берет
- \\ - Тескери кыйгач сызыкты форматтоо үчүн колдонулат
- \ n - Сап тыныгуусун жаратат (жаңы сапка өтөт)
- \ t - Абзац же чегинүү жаратат
- \ ’ же \" - Бир же эки жолу тырмакча үчүн колдонулат

Берилген escape белгилеринин колдонулушун жакшыраак түшүнүү үчүн “\n” белгисин карап чыгалы. Бул escape белгиси кандайдыр бир текстке киргизген сайын жаңы сап түзүүгө мүмкүнчүлүк берет.

Жаңы Python файлын түзүп, анын атын WonderBoyTheme.py деп коюңуз. Бул кодду файлга киргизиңиз:

```
print("Менин атым\nКеремет Бала\nжана бул абдан сонун\nБул шым мага чак экен!")
```

Бир караганда бул код абдан маанисиз жана чаташканга окшойт. Программаны иштеткенибизде \n кандай иштээрин так көрө алабыз (3-5-сүрөт).

```
Менин атым
Керемет Бала
жана бул абдан сонун
Бул шым мага чак экен!
>>>
```

3-5-сүрөт. *Бирдиктүү print() функциясында саптарды форматтоо*

Адатта, ушул код тилкесин көргөндө print() функциясынын ичиндегилердин бардыгы бир сапка чыгып калат деп күтсөк болот. Бирок, \n escape сапты үзгүлтүккө учуратат жана анын ордуна текст өзүнчө саптарда пайда болот.

\t escape дагы окшош ыкма менен иштейт. Болгону ал жаңы сызык жаратпайт. Тескерисинче, тексттеги чегинүү же өтмөктү түзөт. WonderBoyTheme.py файлыбызга дагы бир нече текст киргизели:

```
print("Менин атым\nКеремет Бала\nжана бул абдан сонун\nБул шым мага чак экен!")
print("Ал жерде жолдор \ttap")
print("tap \ttap \tабдан tap!")
```

Эгер сиз ушул кодду иштетсеңиз, анда 3-6-сүрөттө көрсөтүлгөн натыйжалар кайтып келет:

```
Менин атым
Керемет Бала
жана бул абдан сонун
Бул шым мага чак экен!
Ал жерде жолдор         tap
tap     tap     абдан tap!
>>>
```

Сүрөт 3-6. *Escape белгисин колдонуунун дагы бир мисалы*

Жөргөмүш киши ушундай темадагы ырга ээ *болгусу келген*!

Сүрөттүн мисалына көңүл буруңуз! "Tap, tap, tap, абдан tap" деген сөздөрдүн бардыгы боштуктарга кандайча чегинген? Мунун бардыгы \t аркылуу болду.

Акырында \" жана \' белгилерин кайра карап чыгалы. Мурда белгиленгендей, кээде тырмакчаларды экранда басып чыгарган тексттин бир бөлүгү катары колдонгуңуз келет. Бул көйгөй жаратат. Анткени Python сиз тырмакчалар менен эмнени айткыңыз *келгенин* сиз *ага айтмайынча* айырмалай албайт.

Программага сиздин тырмакча белгилерди грамматикалык мааниде, программалык мааниде колдонууну каалаганыңызды билдирүү үчүн сиз жөн эле escape колдоносуз. WonderBoyTheme.py файлына дагы бир нече текст кошолу. Сиздин текст меники менен дал келгенин текшериңиз:

```
print("Менин атым\nКеремет Бала\nжана бул абдан сонун\nБул шым мага чак экен!")
print("Ал жерде жолдор \ttap")
```

```
print("тар \ттар \табдан тар!")
print("\n")
print("Эл мени көргөндө, баары айтып, макул болуп жатышат:")
print("\"Бала, алдагы шым сага чак!\"")
```

Бул программаны иштетип, 3-7-сүрөттө көрүнүп тургандай, натыйжага көз чаптырыңыз:

```
Менин атым
Керемет Бала
жана бул абдан сонун
Бул шым мага чак экен!
Ал жерде жолдор         тар
тар     тар     абдан тар!


Эл мени көргөндө, баары айтып, макул болуп жатышат:
"Бала, алдыгы шым сага чак!"
>>>
```

Сүрөт 3-7. *\t белгисин чегинүү табуляциясын түзүү үчүн колдонуу*

Коддун ушул бөлүгүнө өзгөчө көңүл буруңуз:

```
print("\"Бала, алдагы шым сага чак!\"")
```

Биринчи кош тырмакча (”) андан кийинки нерселерди экранга басып чыгаруу керектигин айтат. Андан кийин Python дагы бир тескери кыйгач сызыкка (\) туш болуп, анын артындагы белгини жөнөкөй текст катары карайт. Акыры ал акыркы кош тырмакчага туш келип, анын алдында эч кандай чыгуу белгиси же тескери кыйгач сызык болбогондуктан, ал сиз басып чыгарууну каалаган тексттин аягына чыгууну ниет кылып жатканыңызды билет.

Эгерде биз эки тырмакчанын бардыгын бир тырмакчага (’) алмаштырсак, анда ушул код так ушундай иштээрин эске алыңыз.

Сиздин арсеналыңыздагы жаңы куралды тааныштыруу: тизмелер

Чынын айтканда, кылмыштуулукка каршы күрөшүү - оор. Супер баатыр (же шакирт, жаңы баштоочу!) кээде кайраттуулукка, тематикалык ырларга жана аларга мүнөздүү супер күчүнөн дагы башка нерсеге таянууга туура келет. Ар бир баатырдын колунан эч нерсе келбей калганда кайрыла турган кандайдыр бир супер курал же гаджет болот. Бул үчүн биз сизди дагы бир нече шаймандар менен жабдып башташыбыз керек. Пайдасы тие турган кур менен сизди жабдыйбыз (ал Бэтмандын куруна окшош. Сиз аны жайма базардан сатып алгансыз).

Биздин эң биринчи гаджеттерибиздин бири *тизмелер* болот. Өзгөрмө маалыматтардын түзүлүшү болгондой эле, *тизмелер* да бар. Бирок, өзгөрмөлөрдөн айырмаланып, тизмелер бир эмес, бир нече маалыматты камтышы

мүмкүн. Ал эми өзгөрмөнү белги же кутуча деп эсептесе болот. Бирок тизмелер кутучалардын тобу менен толтурулган шкафка окшош.

Биз тизмелерди толтура алабыз. Анда ошол эле маалыматтардын түрлөрү, өзгөрмө, анын ичинде саптар, сандар, бүтүн сандар, жылышма чекиттүү сандар жана башкалар бар. Тизмеге маани берүү үчүн, эки квадрат кашаанын ортосуна [] жайгаштырып, үтүр (,) менен бөлүп алабыз.

Кана эмесе тизме түзөлү:

```
суперКүчтөр = ['учуу', 'сонун мүйүз', '20/20 көрүнүш', 'Коддоо көндүмдөрү']
```

Бул коддо биз *суперКүчтөр* деген тизме түзүп, ага өзүнчө төрт маалымат бердик. Бул учурда саптар мааниси: учуу, сонун мүйүз, 20/20 көрүнүш жана Коддоо көндүмдөрү. Эгерде биз бул тизмени басып чыгаргыбыз келсе, анда өзүбүздүн ыңгайлуу print() функциясын колдонушубуз керек:

```
print(суперКүчтөр)
```

Бул тизмени басып чыгарганыбызда бир кызыктуу окуя болот. Биз күткөндөй эле, тизменин мазмунун басып чыгаруунун ордуна ал бүтүндөй структураны басып чыгарат (3-8-сүрөттү караңыз):

```
['учуу', 'сонун мүйүз', '20/20 көрүнүш', 'Коддоо көндүмдөрү']
>>>
```

Сүрөт 3-8. *Тизме басып чыгаруу*

Эсиңизде болсун! Тизме - бул өзүнчө сакталган элементтердин тобу. Биздин тизмедеги ар бир пункт индекс номери катары белгилүү болгон нерсеге дал келет. Бардык тизмелер 0 индексинен башталат жана андан кийин ырааттуу улантылат. Демек, биздин тизмедеги "учуу" 0, “сонун мүйүз” 1, "20/20 көрүнүш" 2 индексинде ж.б.у.с.

Эгерде биз, мисалы, 3 индексинде жайгашкан элементти жөн эле басып чыгаргыбыз келсе, анда мындай код жазышыбыз мүмкүн:

```
суперКүчтөр = ['учуу', 'сонун мүйүз', '20/20 көрүнүш', 'Коддоо көндүмдөрү']
print(суперКүчтөр[3])
```

Натыйжада, Коддоо көндүмдөрү экранга басылып чыгат, анткени ал биздин тизменин 3-индексинде жайгашкан (эсиңизде болсун, тизмелер 0 индексинен башталат). Жакшыраак түшүнүү үчүн тизмебиздеги ар бир элементти өзүнчө басып чыгаралы:

```
суперКүчтөр = ['учуу', 'сонун мүйүз', '20/20 көрүнүш', 'Коддоо көндүмдөрү']
```

```
print(суперКүчтөр[0])
print(суперКүчтөр[1])
print(суперКүчтөр[2])
print(суперКүчтөр[3])
```

3-9-сүрөт бизге натыйжаларды көрсөтөт:

```
учуу
сонун мүйүз
20/20 көрүнүш
Коддоо көндүмдөрү
>>>
```

Сүрөт 3-9. *Тизмедеги маанилерди басып чыгаруу*

Же болбосо, print() функциясын бир сапта төмөнкүдөй код менен жазсаңыз болот:

```
суперКүчтөр = ['учуу', 'сонун мүйүз', '20/20 көрүнүш', 'Коддоо көндүмдөрү']
print(суперКүчтөр[0], суперКүчтөр[1], суперКүчтөр[2], суперКүчтөр[3])
```

Бул учурда ошол эле натыйжага жетишебиз. Келгиле, файл түзүп, анын атын ListExample.py деп коёлу. Ага төмөнкү кодду кошуп, андан соң программаны иштетиңиз (натыйжалар 3-10-сүрөттө көрсөтүлгөн):

```
суперКүчтөр = ['учуу', 'сонун мүйүз', '20/20 көрүнүш', 'Коддоо көндүмдөрү']
print(суперКүчтөр[0], "0 индексинде жайгашкан")
print(суперКүчтөр[1], "1 индексинде жайгашкан")
print(суперКүчтөр[2], "2 индексинде жайгашкан")
print(суперКүчтөр[3], "3 индексинде жайгашкан")
```

```
учуу 0 индексинде жайгашкан
сонун мүйүз 1 индексинде жайгашкан
20/20 көрүнүш 2 индексинде жайгашкан
Коддоо көндүмдөрү 3 индексинде жайгашкан
>>>
```

Сүрөт 3-10. *Тизмедеги маанилерди басып чыгаруунун дагы бир ыкмасы*

Бул мисалда, биз print() функциясынын аягына бир аз текст кошобуз. Биринчи басуу параметрин үтүр менен бөлүп, андан кийин print() функциябыздын экинчи бөлүгүн аныктай тургандыгыбызга көңүл буруңуз. Эгерде биз тизмедеги мааниге (тексттерге) чейин бир нече текстти басып чыгарууну кааласак, анда ушул ыкманы колдонсок болот:

```
print("0 индексинде жайгашкан элемент бул ", суперКүчтөр[0])
```

Бул бизге төмөнкүнү берет: 0 индексинде жайгашкан *элемент* - учуу. Акыры, тизмедеги бардык элементтерди басып чыгаруунун оңой жана натыйжалуу жолу бар. Келгиле, дагы бир файл түзүп, аны PowersWeaknesses.py деп атайлы. Ага төмөнкү кодду кошуңуз:

```
суперКүчтөр = ['учуу', 'сонун мүйүз', '20/20 көрүнүш', 'Коддоо көндүмдөрү']
суперАлсыз = ['болоня', 'лактозага болгон чыдамсыздык', 'коомдук эрежелер', 'тар шым']
print("Мына биздин жаңыдан пайда болгон баатыр/тарапташ, \"Керемет Бала!")
print("Анын супер күчтөрү: ", * суперКүчтөр)
print("Жана анын алсыз жактары: ", * суперАлсыз)
```

Тизменин атын * белгиси менен колдонуу программага тизмени толугу менен колдонушун айтат. Мисалы, эгер сиз print(**суперКүчтөр*) деп терсеңиз, анда программа *суперКүчтөр* тизмесиндеги бардык нерселерди басып чыгарат. Биздин мурунку коддун натыйжасы 3-11-сүрөттө көрсөтүлгөн:

```
Мына биздин жаңыдан пайда болгон баатыр/тарапташ, "Керемет Бала!
Анын супер күчтөрү:  учуу сонун мүйүз 20/20 көрүнүш Коддоо көндүмдөрү
Жана анын алсыз жактары:  болоня лактозага болгон чыдамсыздык коомдук эрежелер т
ар шым
>>>
```

Сүрөт 3-11. *Тизменин бардык мазмунун басып чыгаруу*

Тизмелерди өзгөртүү

Тизмелер өзгөрмө сыяктуу өзгөрүшү мүмкүн. Аларга башка маалыматтарды кошуп, алып салабыз, кайрадан жайгаштырабыз. Мисалы, PowersandWeaknesses.py файлында бизде супер алсыз жактардын тизмеси бар, алардын бири "лактозага болгон чыдамсыздык" (сүт азыктарын иче албай же балмуздак жей албайбыз - жок!). Бактыга жараша, бул алсыздыкты жоюунун жолу бар: аларда сүттөгү ферменттерди сиңирүүгө жардам берген дары-дармектер бар. Ошентип, эми дагы бир жолу оозуңузду балмуздакка толтура аласыз. Ураа!

Бул билимди алганда, биз ушул алсыз жакты супер алсыздыктар тизмесинен алып салгыбыз келет. Бул үчүн del билдирүүсүн колдонобуз.

```
суперАлсыз = ['болоня', 'лактозага болгон чыдамсыздык', 'коомдук эрежелер', 'тар шым']
del суперАлсыз[1]
print(*суперАлсыз)
```

Бул 1-индексте жайгашкан элементти алып салат. Биздин учурда бул "лактозага болгон чыдамсыздык". *суперАлсыз* тизмесинин мазмунун басып чыгарганда буларды көрөбүз: болоня, коомдук эрежелер, тар шым.

Биздин тизмеден бир маанини жок кылуу үчүн ошондой эле алып салуу ыкмасын колдонсок болот. Программага элементтин ордун айтуунун ордуна аны жөн гана мааниси менен камсыз кылабыз:

```
суперАлсыз = ['болоня', 'лактозага болгон чыдамсыздык', 'коомдук эрежелер', 'тар шым']
суперАлсыз.remove('лактозага болгон чыдамсыздык')
```

Бул del билдирүүсүн колдонгондой эле натыйжа берет. Тизмеден элементтерди өчүрүүдөн тышкары, аларды кошо алабыз. Бул үчүн бир нече ыкмалар бар. Биринчиси - append командасын колдонуу. Бул ыкма элементти тизменин аягына улайт же кошот:

```
суперАлсыз = ['болоня', 'лактозага болгон чыдамсыздык', 'коомдук эрежелер', 'тар шым']
del суперАлсыз[1]
суперАлсыз.append('эти')
print(*суперАлсыз)
```

Бул мисалда, биз адегенде *суперАлсыз* тизмебизди түзөбүз. Андан кийин del командасын колдонуп, 1-индекстеги элементти алып салабыз (тизмелер 0-индекстен башталаарын эске алып). Андан кийин биздин жаңы душманыбыз "эт" бар экени аныкталды. Ошондуктан аны тизмеге кошуу үчүн append билдирүүсүн колдонобуз. Жыйынтыктарды басып чыгарганда төмөндөгүчө болот:

болоня коомдук эрежелер тар шым эт

Андан тышкары, бир нерсени биздин тизмеге киргизсек болот. Insert ыкмасы appendтен бир аз башкача иштейт. Бул нерсени тизмедеги каалаган позицияга кошууга мүмкүнчүлүк берет. Ал жерде append аны аягында жайгаштырат. Бул колдонууда кандай көрүнөт:

```
суперАлсыз = ['болоня', 'лактозага болгон чыдамсыздык', 'коомдук эрежелер', 'тар шым']
del суперАлсыз[1]
суперАлсыз.insert(1,'эт')
print(*суперАлсыз)
```

Insert ыкмасы эки аргументти же параметрди колдонот. Биринчиси - тизмеге кошуп жаткан нерсеңизди программага кайда жайгаштырууну, башкача айтканда, кайсы индекстеги абалда болушун кааларын айтат. Экинчи аргумент тизмеге

кандай маани кошууну каалаганын айтат. Жогорудагы кодду иштетүү төмөнкүдөй басылып чыгат:

```
болоня эт коомдук эрежелер тар шым
```

Башка тизме методдору

Жалпысынан алганда, алардын саны он бир. Алардын ар бири тизмеде сакталган маалыматтар боюнча сыйкырдуу операцияларды жасоого мүмкүнчүлүк берет. Алардын көпчүлүгүн карап чыкканыбыз менен, бул бөлүмдө алардын бардыгын чагылдырууга мүмкүнчүлүгүбүз жок. Бирок, төмөндө ар кандай тизме ыкмалары жана аларды эмне үчүн колдонулгандыгы жөнүндө маалыматты таба аласыз. Аларды өзүңүз көнүгүү катары колдонсоңуз болот.

- list.pop(): pop ыкмасы тизмедеги маанини кайтарып берүүгө же басып чыгарууга жана андан кийин алып салууга мүмкүндүк берет. Бул туура элементти алып салып жатканыңызды ырастоого мүмкүндүк берет. Мисалы:

  ```
  print(суперАлсыз.pop(1))
  print(*суперАлсыз)
  ```

- list.reverse(): Тизмедеги нерселерди иреттөөгө болот. Мунун бир жолу - тескери ыкманы колдонуу. Бул сиздин тизмедеги элементтериңиздин иретин өзгөртүп, биринчи элементти аягына чейин, ал эми акыркы элементти алдыга жылдырып ж.б.у.с. алардын ордун өзгөртүүгө негиз берет. Мисалы:

  ```
  суперАлсыз.reverse()
  print(*суперАлсыз)
  ```

- list.sort: Элементтердин иреттүүлүгүн өзгөртүүнүн дагы бир жолу - бул иреттөө ыкмасын колдонуу. Бул ыкма сиздин тизмеңиздеги элементтерди алфавиттик тартипте жөн эле иреттейт. Мисалы:

  ```
  суперАлсыз.sort()
  print(*суперАлсыз)
  ```

- list.count(): Бул ыкма берилген маани тизмеде канча жолу кездешерин эсептөө үчүн колдонулат. Мисалы, сиз "эт" тизмеде канча жолу кездешерин билгиңиз келиши мүмкүн. Биз муну count ыкмасынын жардамы менен аныктасак болот. Мисалы:

  ```
  print(суперАлсыз.count('эт'))
  ```

  Бул "эт" биздин тизмеде канча жолу пайда болгонун

  кайтарып берет. Бул учурда бир эле жолу пайда болду.

- list.extend(): Бул ыкманы колдонуу жөнөкөй. Ал бир тизмени экинчи тизмеге бириктирүү үчүн колдонулат. Мисалы, биздин каармандарыбызды жеңе турган дагы көп элементтерди камтыган *аябайСуперАлсыз* деген тизмеңиз болсо, анда аны эски тизмебиздеги

супер алсыздыктар менен бириктирсек болот. Ушундай жол менен сизде бир гана тизме болот. Мисалы:

```
суперАлсыз.extend(аябайСуперАлсыз)
```

- list.copy(): Айрым учурларда тизмени көчүрүп, анын көчүрмөсү колуңузда болушун каалайсыз. Балким, бул тестирлөө максатында болушу мүмкүн же сизде ушундай эле аталыштагы маалыматтар бар болсо, анда текстти кайра жазгандан көрө, аны оңдоо оңой болмок. Кандай болгон күндө да, copy ыкмасын колдонуу менен тизмени көчүрүп алсаңыз болот. Мисалы:

  ```
  СуперАлсыз2 = СуперАлсыз.copy()
  ```

  Эскертүү: list.copy Python 3.3 жана андан жогору версияларында гана жеткиликтүү. Мурунку версиясын колдонууда AttributeError чыгат.

- List.index(): Көбүнчө кайсы бир нерсенин тизмедеги ордун билиш керек. Керек болгондо аны чакыра алабыз. Кодуңузду издөөнүн ордуна, индекстин жардамы менен тизмедеги маанини аныктай аласыз.

  Мисалы:

  ```
  print(СуперАлсыз.index('эт'))
  ```

- 3 санын кайтарып берет. Анткени "эт" биздин тизмеде 3-орунда турат. (Эскертүү: эгерде сиз ушул ыкмалар менен, айрыкча сорттоо методдору менен тажрыйба жүргүзүп көргөн болсоңуз, анда "эт" сиз үчүн башкача абалда болушу мүмкүн).

- list.clear(): Акыркы ыкма - бул тазалоо - clear ыкмасы. Биз аны эң артына калтырдык. Анткени аны колдонууну үйрөнсөңүз, ал өзү угулуп аткандай бардык нерсени тазалайт. Тизмеңиздеги бардык маалыматтарды тазалап салат. Айрым учурларда тизмедеги маалыматтардын бардыгын өчүрүп салууга туура келиши мүмкүн. Бул ыкма дал ошол үчүн керек. Мисалы:

  ```
  СуперАлсыз.clear()
  ```

Бул эпизоддо

Бул кызыктуу эпизоддо бир топ жерди басып өттүк. Азыркы учурда сиздин күч-кубатыңыз барган сайын көбөйүүдө. Мен аны айтууга батына албайм. Секириктер жана чектер! Жакында сиз өзүңүздүн Керемет Кайык, балким, Керемет Унаа аттуу супер унааңызда иштеп каласыз!

Башкача айтканда, сизде айдоочунун күбөлүгү болсо, унааңызды айдайсыз. Эгер андай болбосо, анда сиз Керемет Велосипед же Керемет Роликтер менен эрмек кыласыз.

Келгиле, ушул бөлүмдөн үйрөнгөнүбүздү кайталайлы, макулбу?

- Комментарийлер биздин кодду биз же башка программист тарабынан келечекте маалымдама үчүн документтештирүүгө мүмкүндүк берет.
- Хэш же # жана бош орунду колдонуу менен комментарийлерди түзөбүз. Андан кийинки бардык тексттерди Python этибарга албайт.
- Эгер кошумча тактоо керек деп эсептесек, анда белгилүү бир коддон кийин комментарий калтыра алабыз. Муну сарамжалдуулук менен пайдаланыңыз.
- Кодду комментарийлегенде учурдагы кодду өчүрбөстөн, каталарды табууга болот. Биз көйгөй эместигин билгенден кийин кодду жөн гана жокко чыгарабыз.
- Escape белгилери, адатта, текст катары эсептелбеген атайын белгилерди басып чыгарууга мүмкүндүк берет. Ошондой эле, алар саптарыңызды форматтоого мүмкүндүк берет.
- Escape белгилери \t, \n, \', \" жана \\ кирет.
- Саптар - тамгалардан, сандардан же белгилерден турган маалыматтардын түрү.
- Кошуу (+) белгисин колдонуп, бир же бир нече сапты бириктирүүгө болот. Ал бириктирүү деп аталат.
- Белгисин колдонуп, саптарды кайталай аласыз. Алардын бир нече нускасын жасай аласыз.
- Тизмелер - бул кутучаларга толгон шкаф сыяктуу иштеген сактоочу бирдиктер. Аларда бир гана нерсени сактасаңыз болот (өзгөрмө болгондо). Сиз аларды мындай жол менен аныктайсыз: суперКүчтөр = [‘учуу’ , ‘20/20 көрүнүш’] ж.б.
- Тизмелер индекстелген элементтерди - маалыматтардын бөлүктөрүн камтыйт. Нерселер 0 индексинен башталып, ырааттуу жүргүзүлөт.
- Тизмелерди бир нече жол менен басып чыгара аласыз. Анын ичинде басып чыгаруу print(суперКүчтөр[1]- бирдиктүү маани үчүн басып чыгаруу же эгерде сиз бардык тизмени басып чыгаргыңыз келсе (* суперКүчтөр).
- del оператору тизмеден бир нерсени жок кылууга мүмкүндүк берет.
- Анын ичинде 11 тизме ыкмасы бар insert(), append(), pop(), reverse(), sort(), count(), extend(), copy(), index(), clear(), and remove()

4 - БӨЛҮМ

ЧЕЧИМ КАБЫЛ АЛУУ

Кылмыштуулукка каршы күрөшүү жөнүндө сөз болгондо, биз, супер баатырлар, көп учурда чечим кабыл алууга мажбур болгон жагдайга туш келебиз. Имараттын капталынан ыргытылып жаткан кызды сактап калабызбы же кылмышкерди кылмыш ордунда кармап калуу үчүн анын жерге кулап түшүшүнө жол беребизби? Бүгүн биз коргоочу кийимибизди жууйбузбу же дагы бир күн жыт менен жүрө беребизби?

Акыр-аягы, кылмыштуулукка каршы күрөшүү жана ал үчүн программалоо - бир нерсеге байланыштуу болот. Ал - чечим кабыл алуу.

"Ар бир иш-аракетке реакция бар" деген сөздөрдү укkан чыгарсыз. Ооба, бул өзгөчө, программалоодо туура. Ойлонуп көрсөңүз: компьютериңизди колдонгон сайын аны чечим кабыл алууга мажбурлайсыз. Чычканыңызды жылдырганда, баскычты басканда, кодуңуз иштебей жатканда бир саат бою экранды башыңыз менен ургулаганда (макул, балким акыркысы жок) - ушунун бардыгын чечмелеп, аткарып беришин сиз компьютерден талап кыласыз жана күтөсүз.

Бул жерде жеңил мисал: “а” тамгасын же “к” тамгасын бассаңыз, компьютер эки сценарий үчүн эмне кылуу керектигин билиши керек. Эгер сиз текст иштетүүчү тиркеме колдонуп жаткан болсоңуз, анда жөнөкөйү - ушул сценарийдеги ошол эки тамганын бирин экранга басып чыгарасыз.

Көбүнчө, компьютердик программалоого байланыштуу чечим кабыл алууну талкуулаганда, биз аны көп жооптуу поп-викторинанын контекстинде түшүндүрөбүз. Программа колдонуучуга бир нече варианттарды сунуштайт - мисалы, A, B, же C тандап, андан кийин кайсы вариант тандалгандыгына жараша реакция жасайт.

Бардык программалоонун эң күчтүү функцияларынын бирин чындап түшүнүү үчүн келгиле, супер баатырдын коргоочу кийимдерин кийип, супер мээбизди иштетип, кийинки өнүгүп жаткан супер күчкө - чечимдерди кабыл алууга умтулалы.

Чечим кабыл алуу

Жашооңузду компьютердик программа катары элестетип көрүңүз. Бул түшкү тамак мезгили жана жаңыдан канат байлаган шакирт - келечектеги баатыр. Ал булчуңдарын чыңдоо үчүн түшкү тамакты талап кылат. Сиздин алдыңызда бир топ

нерселер турат: эки кесим нан, эки куту жаңгак майы жана үч куту шире. Муну жакшыраак көрө алышыбыз үчүн тизмеге киргизели!

- Нан (эки кесим);
- Кыртылдаган жер жаңгак майы;
- Каймактуу жер жаңгак майы;
- Алма шireси;
- Жүзүм шireси;
- Кулпунайдан жасалган кыям.

Көрүнүп тургандай, түшкү тамакты жей электе чечим кабыл алышыңыз керек. Бизде нан бар экенин билебиз. Бирок, жаңгак майынын кайсы түрүн колдонобуз? Ширелер жөнүндө эмне айта аласыз?

Бул сценарий чечим кабыл алуу, андан дагы жакшысы - *шарттуу оператор* катары белгилүү. Башкача айтканда, белгилүү шарттардын аткарылышын биз программа катары кандай кабыл алабыз? Муну программалык жол менен карап көрүү үчүн *псевдокод* деп аталган нерсеге кайрылалы.

Жок, псевдокод сиздин ата-энеңиз 1980-жылдары кызыгып угуп жүргөн Фил Коллинздин эски ыры эмес. Бул - тилди колдонуп, кодду пландаштыруу ыкмасы. Бул код сыяктуу угулганы менен, *код эмес*. Эгерде биз кайрыла турган болсок, же болбосо сэндвич сценарийин псевдокоддой турган болсок, анда мындай көрүнүш пайда болот:

эгерде сендвич жасагың келсе, нан ал.

анда жаңгак майынын түрүн танда;

эгерде жаңгак майынын түрү = "Каймак" болсо,

басып чыгар "Бул кычкыл. Кычкыл жакпайт."

андай болбосо басып чыгар "Кыртылдаган жер жаңгак майы - бул туура тандоо! Сиздин табитиңиз жакшы!"

Андан кийин шire түрүн тандаңыз;

эгерде шire түрү = "Жүзүм шireси" болсо,

басып чыгар "Сизге бул шire түрү жакпайбы?"

эгерде андай болбосо шire түрү = "Кулпунайдан жасалган кыям"

басып чыгар "Албетте, жаңгак майыңызды бузгуңуз келсе, анда алга."

андай болбосо басып чыгар "Алма шireси - бул жалгыз шire! Сиз ушунчалык акылдуусуз!"

Кийин жаңгак майын жана ширени нанга сүйкөп, кытыраган жерин калтырып, тамактаныңыз.

Эгер сиз бул кодду программага киргизсеңиз, анда сиз бир топ ката кетиресиз. Анткени эсиңизде болсун, бул иштеген код эмес. Бул жасалма же шылдың дегенди туюндурган псевдокод. Псевдокодду колдонуп, кээде реалдуу коддоодон мурун программаларыбызды түзүп, маанилүү бөлүмдөрүн картага түшүрө алабыз. Бул биздин программалоо логикабызга жардам берет жана биздин коддогу каталардан алыс болууга мүмкүндүк берет. Бул сиздин досуңуз комикстер дүкөнүнө кантип барууга боло тургандыгын жазып берген көрсөтмөлөрдүн топтому сыяктуу болушу мүмкүн. Бул сулуу эмес көрүнүшү мүмкүн (бирок кээ бир псевдокоддор сулуу келип, диаграммаларга жана графиктерге толгон). Бирок ал сизге кайда баруу керектиги тууралуу түшүнүк берет.

Шарттуу операторлор

Терминдердин эң негизгиси боюнча, шарттуу билдирүүлөр - бул коддун бир бөлүгү иштейби же жокпу аныктай турган коддун үзүндүлөрү - шарт аткарылгандыгына же аткарылбагандыгына байланыштуу болот. Программалоо көз карашынан алганда, шарттуу операторлорду жөнөкөй мисалдарда колдонсо болот. Мисалы:

Эгерде колдонуучу супер баатыр болууну тандаса, анда аны "баатыр" категориясына киргизиңиз. Болбосо, аларды душман категориясына киргизиңиз.

- Эгерде супер баатыр уулуу калдыктарга тийип, алардын супер күчүнө ээ болсо, анда аларды "мутацияланган" деп бөлүп көрсөтүңүз. Эгер андай болбосо, анда аларды "супер державалар" деп бөлүңүз.
- Эгер супер баатыр трагедиялуу жагдайга ээ болсо, анда алардын мүнөзүн "Кара жана түрү суук" кылыңыз. Болбосо, алардын мүнөзүн "Ыкчам жана күлкүлүү" кылыңыз.

Бул шарттуу операторлордун эң жөнөкөй колдонулушу. Чыныгы дүйнөдө программанын белгилүү бир бөлүгү аткарылышы үчүн бир нече шарттар болушу мүмкүн (же болбойт). Жакын арада биз өнүккөн типтерге киребиз. Бирок биз азыр алардын бардыгынын эң негизги шарттуу операторун карап көрөлү. Бул - **If оператору.**

IF оператору

Жогорку мисалдардын бардыгы - if оператору деп аталган нерсенин бир бөлүгү. **If** операторду*о эгерде* кандайдыр бир нерсе болот десе, анда *аны* жаса. Мындан тышкары, эгер кандайдыр *бир нерсе* ишке ашпай калса, программа өзү бир нерсе жасайт деп ойлосоң, жаңылышкан болсуң. Программа такыр эч нерсе жасабайт.

Маселени түшүнүктүүрөөк кылуу үчүн бир аз кодго байкоо жүргүзүп көрөлү. ConditionalStataments.py деп аталган жаңы Python файлын түзүп, ушул кодго киргизиңиз:

```
суперБаатырТүрү="Маанайы суз жана Капалуу"
```

```
print("Бул жерде кандай баатыр бар экенине көз чаптыралы...")
if суперБаатырТүрү =="Маанайы суз жана Капалуу":
    print("Аха, бул жерде сен 'Маанайы суз жана Капалуу' экенсиң.")
    print("Башында трагедиялуу окуя болгон деп ойлоп жатам!")
    print("Сенин үнүң өтө эле орой угулат ...")
    print("Мына, жөтөлдү азайт. Же экөөнү тең.")
```

Бул коддо бир нече нерсени белгилей кетүү керек. Жаңы баштоочулар үчүн биз *суперБаатырТүрү* деген сап өзгөрмөсүн түзүп, аны бир нече текст менен толтурдук. Бул өзгөрмө текстте өзүбүздүн **If** операторубуздун эмнеге шарттуу экенин текшеребиз.

Текстти басып чыгаргандан кийин **if** билдирүүсүн төмөнкү сап менен баштайбыз:

```
if суперБаатырТүрү =="Маанайы суз жана Капалуу":
```

Программа ушул код сабын көргөндө, шарттуу операторун киргизип, анын **ЧЫН** же **ЖАЛГАН** экенин текшерет. Эгер шарт аткарылса, башкача айтканда, натыйжа чын болсо, программа *чегинген* коддун *калган* бөлүгүн (демек, анын бир бөлүгүн) - **if** операторун иштетет.

Бул учурда шарт аткарылат: с**уперБаатырТүрү** тексти **"Маанайы суз жана Капалуу"** менен *барабар*. Ошондуктан программа **if** операторунун курамына кирген print() функцияларын басып чыгарат. Ушундай болгондуктан, программанын натыйжалары төмөндөгүчө болот (4-1-сүрөттү караңыз):

```
Бул жерде кандай баатыр бар экенине көз чаптыралы...
Аха, бул жерде сен 'Маанайы суз жана Капалуу' экенсиң.
Башыңдан трагедиялуу окуя болгон деп ойлоп жатам!
Сенин үнүң өтө эле орой угулат ...
Мына, жөтөлдү азайт. Же экөөнү тең.
>>>
```

Сүрөт 4-1. *Шарттуу оператор менен иштөө*

Дагы бир белгилей кетүүчү нерсе: == белгиси *салыштыруу оператору* катары белгилүү. Бул учурда салыштырылып жаткан маани мурунку тырмакчадагыга ("") дал келиши керек дегенди билдирет. Текстти жана сандарды баалоодо, салыштырууда == белгисин колдонобуз.

Бирок биздин шарт аткарылбаса эмне болот? Эгер *суперБаатырТүрү* мааниси “Маанайы суз жана Капалуу” менен дал келбесе эмне болот? Муну билүү үчүн, биздин кодду оңдоп, анын маанисин өзгөртүү жана программаны кайра иштетүү керек.

```
суперБаатырТүрү = "Ыкчам жана күлкүлүү"
print("Бул жерде кандай баатыр бар экенине көз чаптыралы...")
```

```
    if суперБаатырТүрү =="Маанайы суз жана Капалуу":
        print("Аха, бул жерде сен 'Маанайы суз жана Капалуу' экенсиң.")
        print("Башында трагедиялуу окуя болгон деп ойлоп жатам!")
        print("Сенин үнүң өтө эле орой угулат ...")
        print("Мына, жөтөлдү азайт. Же экөөнү тең.")
```

Эми биз өзүбүздүн кодду иштеткенибизде, биринчи print() функциясы гана кайтарылат (4-2-сүрөттү караңыз):

```
Бул жерде кандай баатыр бар экенине көз чаптыралы...
>>>
```

Сүрөт 4-2. *if операторунун натыйжалары*

Эмне үчүн мындай болуп жатат? Биздин *суперБаатырТүрү* өзгөрмөсүндөгү маанини өзгөрткөндүктөн, Python **if** операторубузга жооп бергенде ал шартты текшерип, анын иштебей калганын жана жалган болуп калганын аныктайт. Шарт аткарылбагандыктан, Python *if* операторунун калган бөлүгүн өткөрүп жиберип, программанын кийинки бөлүгүнө өтөт.

Программанын кийинки бөлүгү жок болгондуктан, программа аяктайт. Биз, албетте, программабызда бир нече **if** операторлорун түзө алмакпыз. Эгер андай кылсак, анда Python ошол шарттардын аткарылышын камсыз кылып, ошол билдирүүлөрдүн ар бирин баалап, коддор блогун аткармак. Келгиле, ConditionalStatements.py файлын ачып, кодду төмөнкүлөргө дал келгидей кылып өзгөртөбүз:

```
суперБаатырТүрү ="Ыкчам жана күлкүлүү"
print("Бул жерде кандай баатыр бар экенине көз чаптыралы...")
if суперБаатырТүрү =="Маанайы суз жана Капалуу":
    print("Аха, бул жерде сен 'Маанайы суз жана Капалуу' экенсиң.")
    print("Башында трагедиялуу окуя болгон деп ойлоп жатам!")
    print("Сенин үнүң өтө эле орой угулат ...")
    print("Мына, жөтөлдү азайт. Же экөөнү тең.")
if суперБаатырТүрү =="Аябай сылык":
    print("Бул жерде сени 'Аябай сылык' деп жатат")
    print("Эшикти ачкан бойдон туруп калсаңыз кылмышкерди кантип
    кармайсыз?")
    print("Андан кечирим сураба – ал жаман адам!")
if суперБаатырТүрү =="Ыкчам жана күлкүлүү":
    print("Оо жигит. Сенин Ыкчам жана күлкүлүү экениңди окуп жатам.")
```

```
print("Менин сага тамашам бар:")
print("Кимде 8 манжа жана 2 бармак бар, кызыктуу эмес бекен?")
print("Сенде!")
```

Бул модификацияланган код менен биз программабызга бир эмес, үч билдирүү кошобуз. Программа текстти басып чыгаруудан башталат. Андан кийин *суперБаатырТүрү* маанисинин **"Маанайы суз жана Капалуу"** экендигин текшерген биринчи if операторуна жолугат. Бул андай эмес болгондуктан, биздин программа ушул чегинген блоктун калган бөлүгүн четке кагат.

Кесилген тексттик кодуңуз бар болгондо Python чегинген код ошол коддун белгилүү бир тобуна таандык экендигин билет. Чегинген код бүткөндөн кийин, белгилүү бир блоктун бүткөндүгүн билип, кийинки топтомго өтөт.

Андан кийин Python биздин экинчи **if** операторуна өтүп, критерийлерди дагы бир жолу текшерет: *суперБаатырТүрү* **"Аябай сылык"** менен барабарбы? Ал дагы барабар эмес, андыктан кийинки код блогуна өтөт. Бул биздин үчүнчү **if** операторубуз болуп калат.

Ушул үчүнчү **if** операторунда биз *суперБаатырТүрү* маанисинин **"Ыкчам жана Күлкүлүү"** менен барабар экендигин текшеребиз. Бул жолу натыйжа туура, ошондуктан программа коддун ушул блогунун бөлүгү болуп саналган **print**() функцияларын аткарат. Натыйжаны 4-3-сүрөттөн көрүүгө болот:

```
Бул жерде кандай баатыр бар экенине көз чаптыралы...
Оо жигит. Сенин Ыкчам жана күлкүлүү экенинди окуп жатам.
Менин сага тамашам бар:
Кимде 8 манжа жана 2 бармак бар, кызыктуу эмес бекен?
Сенде!
>>>
```

Сүрөт 4-3. *Чындык катары бааланган if операторунун мисалы*

Логикалык оператор жана салыштыруу операторлору

Шарттуу операторлорго кирүүдөн мурун көңүлдүү сөздөрдү аныктап алышыбыз керек. Бул күлкүлүү сөздөрдү досторуңузга жана үй-бүлөңүзгө кайталап айтып берүү жагымдуу нерсе гана эмес, ошондой эле алар - сиздин куруңуздагы дагы бир ыңгайлуу курал.

Биринчи сөз - **Boolean**. Келиңиз, үнүңүздү бийик чыгарып айтып, өз тутумуңузду күлкүдөн тазалаңыз. Андан кийин үйдү бир нече жолу айланып чуркап чыгып, маектериңизде Boolean деген сөздү канча жолу колдоноруңузду санап көрүңүз. Мен ушул жерде күтөм.

Boolean маалыматтардын дагы бир түрү болуп саналат. Ошондой эле сиз ConditionalStatements.py кодун буга чейин түшүндүрүп бергенден болжолдонгондой,

бул маалыматтын түрү **True** жана **False** деген эки башка мааниге ээ болушу мүмкүн.

Биз **if** шарттуу операторлор менен иштегенде, белгилүү бир шарттын чын же жалган экендигин сурап жатабыз. Ошол шарттарды же критерийлерди канчалык сөз кылбайлы, күндүн аягында, жооп ушул эки тандоонун бирөөсү гана болушу мүмкүн.

Албетте, биз жөн гана чындыктын оюнун ойной албайбыз же компьютерди тобокелчиликке сала албайбыз. Андыктан Python жана башка тилдерде маалыматтарды салыштырып, акыркы натыйжанын чын же жалган экендигин аныктоого жардам берүү үчүн салыштыруу операторлору деп аталган нерсени колдонобуз.

Салыштыруу операторлорунун бирин талкууладык, ал == барабар оператору. Мындан сырткары, бизде дагы беш салыштыруу оператору бар. Алар төмөнкүлөр:

- == Мааниси башка мааниге барабар экендигин билүү үчүн колдонулат;
- ! = Маанинин башка бир мааниге барабар болбогондугун билүү үчүн колдонулат;
- < Маанинин башка мааниден аз экендигин аныктоо үчүн колдонулат;
- > Маанинин башка мааниден чоң экендигин аныктоо үчүн колдонулат;
- <= Кичине же барабар маани үчүн колдонулат;
- > = чоңураак же барабар маани үчүн колдонулат.

Азырынча биз шарттуу операторубуз үчүн сап менен иштедик. Жаңы куралдарыбызды, салыштыруу операторлорун жакшыраак түшүнүү үчүн анын ордуна сандар менен иштөөгө өтөлү. Баштоо үчүн, MathIsHard.py аттуу жаңы файл түзүңүз. Ага төмөнкү кодду киргизиңиз:

```
кереметБала = 10

жаңыКийим = 20

print("Бул жаңы коргоочу кийим жылтырак болуш керек. Сенин алганга мүмкүнчүлүгүң жетеби деп ойлонуп жатам...")

if кереметБала > жаңыКийим:

    print("Куттуктайбыз! Жаңы коргоочу кийимди сатып алууга акчаңыз жетиштүү!")

if кереметБала < жаңыКийим:

    print("Сиз ошол сүлгүңүздү коргоочу кийим кылып кийип жүрүшүңүз керек окшойт...")

    print("Балким сиз Керемет Атаңыздан жакшылап сурасаңыз акча берип калаар...")
```

Келгиле, ушул кодду кененирээк карап чыгалы, макулбу? Эки өзгөрмө түзүүдөн баштайбыз: *кереметБала* жана *жаңыКийим*. Андан кийин “Бул жаңы

коргоочу кийим жылтырак болуш керек. Сенин алганга мүмкүнчүлүгүң жетеби деп ойлонуп жатам ... "

Керемет Бала чындыгында эле ошол жаңы коргоочу кийимди алар-албасын билиш үчүн, *кереметБала* (сиздин акчаңызды билдирет) менен *жаңыКийим* (ошол жалтырак жаңы коргоочу кийимдин баасын чагылдырган) менен салыштыруу керек.

Биздин биринчи **if** билдирүүбүз *кереметБала* өзгөрмөсүнүн мааниси менен *жаңыКийим* өзгөрмөсүнүн маанисинен чоң же барабар экендигин текшерет. Эгер ошондой болсо, анда ал текстти басып чыгарат: «Куттуктайбыз! Жаңы коргоочу кийимди сатып алууга акчаңыз жетиштүү!» Бирок, акча жаңы коргоочу кийимдин наркынан көп болбогондуктан, анын мааниси **чын** экендигин билүү үчүн программа кийинки билдирүүгө өтөт.

Бирок, акча жаңы коргоочу кийимдин наркынан көп болбогондуктан, анын мааниси **чын** экендигин көрүү үчүн программа кийинки билдирүүгө өтөт.

Бул шарт аткарылып, true маанисин бергендиктен, **if** операторунун калган бөлүгүн аткарышат, натыйжада (4-4-сүрөттү караңыз):

```
Бул жаңы коргоочу кийим жылтырак болуш керек. Сенин алганга мүмкүнчүлүгүң жетеби деп
ойлонуп жатам...
Сиз ошол сулгуңузду коргоочу кийим кылып кийип жүрүшүңүз керек окшойт...
Балким сиз Керемет Атаңыздан жакшылап сурасаңыз акча берип калаар...
>>>
```

Сүрөт 4-4. *If билдирүүлөрүн баалоо*

Boolean оператору чындыгында кандайча иштээрин көрүү үчүн, жаңы Python файлын түзүп, анын атын **BooleanExamples.py** деп коюңуз. Бул кодду киргизиңиз:

```
# Ар кандай мааниси бар эки өзгөрмө түзүү
a = 10
b = 20
#Ар кандай салыштыруу операторлорун колдонуп, маанилерин
#салыштырыңыз
print("a  мааниси b маанисине БАРАБАРБЫ?", a == b)
print("a мааниси b маанисине БАРАБАР ЭМЕСПИ?", a!= b)
print("b маанисине  караганда a мааниси  ЧОҢБУ? ", a> b)
print(" a мааниси b маанисинен КИЧИНЕБИ ? ", a <b)
print("a мааниси b маанисинен ЧОҢ же БАРАБАРБЫ? ", a>= b)
print(" a мааниси b маанисинен КИЧИНЕ же  БАРАБАРБЫ? ", a <= b)
```

Бул программаны иштетүү менен кайсы салыштыруу чын, кайсынысы жалган экендигин көрсөтөт. Чындыктын мааниси салыштыруунун туура экендигин билдирет, ал эми **жалган** туура эмес дегенди билдирет.

ELSE билдирүүсү

Эми **if** операторун жана салыштыруу операторлорун түшүнгөндөн кийин, шарттуу операторду башка түрүнө өтсөк болот: ал - **else** оператору. Азырынча биз берилген шарт аткарылганда гана коддордун жыйындысын аткарган шарттуу операторлорду колдонуп келебиз. Бирок, натыйжасы чын болсо, жыйынтыгы чын болсо, калганы башка болсо, эмне болот?

Техникалык жактан бул натыйжага бир нече **if** билдирүүлөрүн колдонуп жетишсек дагы, программаңызды жазуунун мыкты жана натыйжалуу жолу бар. Келгиле, **MathIsHard.py** файлынын кодун түзөтүп, аны төмөнкүлөргө дал келгидей кылып өзгөртөбүз:

```
кереметБала = 10

жаңыКийим = 20

print("Бул жаңы коргоочу кийим жылтырак болуш керек. Сенин алганга мүмкүнчүлүгүң жетеби деп ойлонуп жатам...")

if кереметБала > жаңыКийим:

    print("Куттуктайбыз! Жаңы коргоочу кийимди сатып алууга акчаңыз жетиштүү!")

else:

    print("Сиз ошол сүлгүңүздү коргоочу кийим кылып кийип жүрүшүңүз керек окшойт...")

    print("Балким сиз Керемет Атаңыздан жакшылап сурасаңыз акча берип калаар...")
```

Бул версияда **if** билдирүүсүн экинчи **else** билдирүүсүнө алмаштырдык. **Else** оператору **if** оператору өзүнүн шартын аткара албаса гана иштей баштайт. Негизинен, сиз программага: “Эгер ушундай боло турган болсо, анда муну жаса, антпесе, тигини жаса!” - деп жатасыз.

Бул программанын натыйжасы мурдагыдай эле; бирок азыр код азыраак болуп калды, ошондуктан, экинчи **if** деген сөз жок болгондуктан, башка салыштыруу жүргүзүүнүн кажети жок. Бул эсептөө кубатын жана иштетүүнү үнөмдөйт. Бул жерде анчалык деле чоң иш эместей көрүнгөнү менен, сиз он миңдеген саптары жана жүздөгөн **if** операторлору бар программада канча үнөмдөп калаарыңызды элестетсеңиз болот.

Дагы бир эскертүү: **else** операторун колдонгондо, Python ар дайым сиздин if блогуңузду же else блогуңузду аткарат. Сиздин программаңыз ушул эки жолдун бирин аткармайынча эч качан бүтпөйт.

Бирок сизге **if** жана **else** варианттарынан көп нерсе керек болсо эмне болот? Үч вариант болсо кандай болот эле? Же төртөөбү? Же төрт миллиардбы? Бул үчүн бизге тырмактай if жана else билдирүүсүнөн күчтүү нерсе керек болот.

Ошол ыйгарым укуктарды дагы бир жолу жаңыртууга даярсызбы? Андай болсо, үйрөнүүгө даяр болуңуз. Алдыда **else if** келе жатат!

ELSE IF билдирүүсү

Сиздин оюңузду билем. Else if бул чыныгы фраза эмес го деп турасыз. Чындыгында, бул күн бою уй саап, чарчаганынан чекесинен терин аарчып, эмгектенип жүргөн дыйкандын аты ушундай болсо керек. Аны else if байке деп атаңыз!

Ооба, мен сизге жаңылык тараткандыжаман көрөм. Бирок бул жерде else if чыныгы сөз айкашы жана ал шарттуу оператор үй-бүлөсүнүн чыныгы мүчөсү болот. Бул өтө ар тараптуу, натыйжалуу жана сизге программист катары, эң жакын досторуңуздун бири болот. Анын жардамы менен сиз кадимки if же else операторлорунун кызыксыз аралашмасы менен жөнөкөй бир же эки сценарийдин ордуна, каалаган сандагы шарттуу сценарийлерди түзсөңүз болот.

Адаттагыдай эле, бул жаңы күчтү үйрөнүүнүн мыкты жолу – колдонуп көрүү, аны сынап көрүү. Ошентип, жогорудагыны эске алып, UncleElseIf.py деп аталган жаңы файлды түзүңүз жана ушул кодду жазыңыз:

```
# Биздин акчабызды жана жаңы коргоочу кийимдин баасын чагылдырган
#өзгөрмөлөрүңүздү түзүңүз

кереметБала = 20

жаңыКийим = 20

print("Бул жаңы коргоочу кийим жылтырак болуш керек. Сенин алганга
мүмкүнчүлүгүң жетеби деп ойлонуп жатам...")

if кереметБала > жаңыКийим:

      print("Куттуктайбыз! Жаңы коргоочу кийимди сатып алууга акчаңыз
      жетиштүү!")

      print("Сизде бир аз акча ашып калгансып турат.")

      print("Чач тегиздетүү боюнча эмне дейсиз? Чачыңыз маскаңызды жаап
      турат!")

elif кереметБала == жаңыКийим:

      print("Жаңы коргоочу кийимди сатып алууга гана акчаңыз жетиштүү!")

      print("Айырмасы жок!")

      print("Мм...жана мага эч кандай башка кеңеш жок окшойт...")

else:

      print("Сиз ошол сүлгүңүздү коргоочу кийим кылып кийип жүрүшүңүз
      керек окшойт...")

      print("Балким сиз Керемет Атаңыздан жакшылап сурасаңыз акча берип
      калаар...")
```

Бул код тааныш көрүнүшү мүмкүн, анткени ал - MathIsHard.py файлынын өзгөртүлгөн версиясы. Жаңы баштоочулар үчүн биз *кереметБала* маанисин 20га өзгөрттүк (көбөйгөнү менен куттуктайбыз!). Эмне үчүн экенин бир нече мүнөттөн кийин түшүнөсүз. Андан кийин, биз print() кириш операторубузду жазабыз, андан

кийин биринчи if блогубузду кошобуз. Бул биринчи кезекте биздин *if* акчанын жаңы коргоочу кийимдин наркынан *жогору* экендигин текшерет. Бул салыштыруу *жалган* жыйынтык бергендиктен, программа print() функциясын өткөрүп жиберип, кийинки блокко өтөт.

Бир секундага токтоңуз! Кийинки блок if же else эмес. Чынында, бул else-if дагы эмес – анда эмне? else-if операторун колдонуп, гибриддүү else жана if – elif түзүлгөн.

Python elif кодуна келгенде, ал дагы бир жолу салыштырып көрөт - бул учурда биздин акча жаңы коргоочу кийимдин наркы менен *бирдей экендигин* текшерет. Бул эки өзгөрмө 20 маанисине ээ болгондуктан, калган башка функциялар аткарылып, чегинилген print() функциялары өз сыйкырын жасайт.

Else-if true деп баалангандыктан, программа мындан ары издөөнүн кажети жок экендигин билип, коддун ушул блогунан чыгып кетет. Биздин if, else, and else- if блогунан кийин код жок болгондуктан, программа аяктайт.

Бул жерде бардыгы кызыктуу боло баштайт. If, else жана else-if тердин өз блоктору бар деп эсептегени менен, чындыгында, алардын бардыгы бир блоктун бөлүгү. Ойлонуп көрсөңүз болот: сиз else-ifти, else же if жок колдоно албайсыз, туурабы? Балким, колдонушуңуз мүмкүн, бирок анда сиздин программа сиз ойлогондой иштебей калышы ыктымал.

Жогоруда айтылгандай, else-if билдирүүлөр каалаган санда варианттарды түзүүгө мүмкүндүк берет. Келгиле, дагы бир нече elif билдирүүсүн кодубузга кошуп, натыйжаларын карап көрөлү. UncleElseIf.py текстин төмөнкүлөргө дал келтирүү үчүн өзгөртүңүз:

```
# Биздин акчабызды жана жаңы коргоочу кийимдин баасын чагылдырган
#өзгөрмөлөрүңүздү түзүңүз

кереметБала = 20

жаңыКийим = 20

print("Бул жаңы коргоочу кийим жылтырак болуш керек. Сенин алганга
мүмкүнчүлүгүң жетеби деп ойлонуп жатам...")

# Акча жаңы коргоочу кийимдин наркынан жогору экендигин текшериңиз

if кереметБала > жаңыКийим:

      print("Куттуктайбыз! Жаңы коргоочу кийимди сатып алууга акчаңыз
      жетиштүү!")

      print("Сизде бир аз акча ашып калгансып турат.")

      print("Чач тегиздетүү боюнча эмне дейсиз? Чачыңыз маскаңызды жаап
      турат!")

# Акчабыз жаңы коргоочу кийим менен бирдей экендигин билип алыңыз

elif кереметБала == жаңыКийим:

      print("Жаңы коргоочу кийимди сатып алууга гана акчаңыз жетиштүү!")
```

```
    print("Айырмасы жок!")
    print("Мм...жана мага эч кандай башка кеңеш жок окшойт...")

# Акчабыз нөл экендигин текшерип көрүңүз
elif кереметБала == 0:
    print("Оо жигит, сен сындың!")
    print("Балким, коргоочу кийимди илип, алжапкычты алууга убакыт
    келди!")
    print("Кайрадан келүүгө убакыт келди... Жигит!")

# Эгерде башка шарттардын бардыгы аткарылбаса, анда else дагы башка
#шарттарды жаратат
else:
    print("Сиз ошол сулгуңузду коргоочу кийим кылып кийип жүрүшүңүз
    керек окшойт...")
    print("Балким сиз Керемет Атаңыздан жакшылап сурасаңыз, акча берип
    калаар...")
```

Коддун ушул версиясында биз коддун үзүндүлөрүн айкыныраак кылуу үчүн ар бир бөлүмгө комментарийлерди (#) коштук. Ошондой эле, шарттуу блокко экинчи elif билдирүүсүн коштук. Анда *кереметБала* өзгөрмөсүнүн мааниси 0 экендигин текшерип, эгер ошондой болсо, жаңы жумушка орношууну сунуш кылган текстти басып чыгарат.

Теория жүзүндө, эгерде, биз шартты аткарышыбыз керек болсо, анда биз шарттуу блокко каалаганча *elif* кошо алабыз,. Мисалы, *кереметБала* өзгөрмөсүнүн маанисин 20га жеткенге чейин бирден текшерип чыксак болот. Мисалы, анын кандай болорун карап көрөлү:

```
# Биздин акчабызды жана жаңы коргоочу кийимдин баасын чагылдырган
#өзгөрмөлөрүңүздү түзүңүз
кереметБала = 20
жаңыКийим = 20
print("Бул жаңы коргоочу кийим жылтырак болуш керек. Сенин алганга
мүмкүнчүлүгүң жетеби деп ойлонуп жатам...")
if кереметБала == 0:
    print("Жок. Сизге 20 сом керек.")
elif кереметБала == 1:
    print("Жок. Сизге дагы 19 сом керек.")
elif кереметБала == 2:
    print("Жок. Сизге дагы 18 сом керек.")
elif кереметБала == 3:
```

```
     print("Жок. Сизге дагы 17 сом керек.")
elif кереметБала == 4:
     print("Жок. Сизге дагы 16 сом керек.")
elif кереметБала == 5:
     print("Жок. Сизге дагы 15 сом керек.")
# 19га жеткенге чейин elif кошуп чыгыңыз
# Андан кийин, 20 же андан жогорку маани үчүн else жазыңыз
else:
     print("Жетиштүү болду!")
```

Бул коддун үлгүсүндө биз баанын алгачкы 5 сомун жабуу үчүн дагы 5 elseif коштук. Мен 19 elseif кошо алмакмын. Бирок, ал китептин бир нече бетин ээлемек. Андан көрө, бош орундарды өзүңүз толтуруп, программаны сынап көрүңүз. *кереметБала* же *жаңыКийим* маанисин бир нече жолу алмаштырыңыз. Биздин шарттарды текшерип жаткан өзгөрмөлөрдөгү мааниге жараша натыйжалар кандайча өзгөрөрүн көрө аласыз.

Логикалык операторлор

Elif шарты канчалык күчтүү болсо дагы, *шарттуу операторлордун ... ратордун... тордун ... дун дасыккан адиси* (бул жерде жаңырыкты уктуңузбу*?")* болууга үйрөнүшүңүз керек болгон дагы бир жөндөм бар. Бул мүмкүнчүлүк *логикалык операторлор* деп аталат.

Азырынча биз бир катар операторлордун түрлөрүн, анын ичинде ушул бөлүмдө салыштырма операторлорун камтыдык. Логикалык үч гана оператор бар. Бирок, алар программаларыңызга жаңы деңгээлдеги күч берет.

Салыштыруу операторлору сыяктуу эле, логикалык операторлордун бир гана максаты бар. Алар өзгөрмөлөрдүн маанилерин бири-бирине салыштырууга жардам берет. Салыштыруу операторлору сыяктуу эле, логикалык операторлор Boolean - логикалык жоопту табышат: true же false. Алар биринчи кезекте эки же андан көп салыштыруунун туура же туура эмес экендигин аныктоо үчүн колдонулат. Башка операторлордон айырмаланып, логикалык операторлор атайын белгилерден же символдордон турбайт. Тескерисинче, алар - баарыбызга түшүнүктүү болгон and – жана, not - эмес, or- же сыяктуу чыныгы сөздөр.

Булардын биринчиси – **AND**. Аны түшүнүү эң оңой болсо керек. Ал жөн гана билдирүүнү карап, "бул **ЖАНА** тигил" экөө тең туура экендигин аныктоого аракет кылат. Эгерде экөө тең болсо, анда ал туура деп баалайт; эгер бир же бир нече шарт аткарылбаса, анда ал жалган деп баалайт.

Келгиле, аны коддо бир аз жакыныраак карап көрөлү. **LogicalOperatorsExample.py** деп аталган жаңы файл түзүп, төмөнкү коддун үзүндүлөрүн киргизиңиз:

```
# Баалоо үчүн бир нече өзгөрмө түзүңүз
кереметБала = 20
жаңыКийим = 20
эскиКийим = 'сасык'

# Акча жаңы коргоочу кийимдин баасына барабар экендигин текшериңиз
#AND
# эски коргоочу кийим "сасып" жатат
if кереметБала >= жаңыКийим and эскиКийим == 'сасык':
      print("Оо... сиз коргоочу кийимди сатып алууга мүмкүнчүлүгүңүз
      бар!")
      print("Ал эми эски коргоочу кийимиңиз чындыгында сасык!")
      print("Эмнеге өзүңүздү жаңы коргоочу кийим менен сыйлабайсыз?")

# Эгерде if аткарылбаса, else дагы башка нерсени берет
else:
      print("Кечиресин балам, азырынча жаңы коргоочу кийимдин убагы
      эмес.")
```

Керемет бала жаңы коргоочу кийимди сатып алуудан мурун эки шарт аткарылышы керек. Биринчиден, анын чыгымдарды жабуу үчүн 20 сому болушу керек. Экинчиден, анын эски коргоочу кийими сасык жыттанып кетиши керек. Ал өзүнүн өмүр бою топтогон акчасын жаңы коргоочу кийимге сарптоонун бирден-бир жолу болот!

Текшерилүүгө тийиш болгон өзгөрмөлөрүбүздү аныктагандан кийин биз if билдирүүсү менен *кереметБала* мааниси *жаңыКийим* маанисинен чоңбу же барабарбы (> =) текшеребиз. Бул учурда ал бирдей болот. Ошондуктан программа андан ары уланып, **and** операторун көрөт да, текшерүүнүн кийинки бөлүгү дагы *чын* болушу керек экендигин билип, бардык if операторлорун толугу менен туура деп баалайт. *эскиКийим* маанисинин "сасык" экендигин текшерет. Демек, дал келет жана эки шарты тең **туура** болгондуктан, if операторунун калган бөлүгүн басып чыгарууга өтөт.

Эгерде шарттардын экөө тең туура эмес болсо, анда else оператору ишке киргизилмек. Мына жыйынтык (4-5-сүрөт):

```
Оо... сиз коргоочу кийимди сатып алууга мүмкүнчүлүгүңүз бар!
Ал эми эски коргоочу кийимиңиз чындыгында сасык!
Эмнеге өзүңүздү жаңы коргоочу кийим менен сыйлабайсыз?
>>>
```

Сүрөт - 4-5. *Else билдирүүсүн колдонуу*

Логикалык операторлорубуздун кийинкиси: or. Шарттуу операторлор үчүн колдонулганда **or** оператору жок дегенде бир шарттын чын деп бааланышын талап кылат. Башка шарт (шарттар) *жалган* болушу мүмкүн, бирок бирөө *чын* болсо, бүтүндөй билдирүүнү туура деп баалайт.

or операторун колдонууга мисал:

```
# Текшерилүүчү өзгөрмөлөр
кереметБала = 20
жаңыКийим = 20
жаңыБутКийим = 50

# Жаңы коргоочу кийимди сатып алууга мүмкүнчүлүгүңүз барбы же жаңы бут
#кийим ала аласызбы, же жокпу текшерет
if кереметБала >= жаңыКийим or кереметБала >= жаңыБутКийим:
     print("Жаңы коргоочу кийим же жаңы бут кийим сатып алууга
     мүмкүнчүлүгүңүз бар окшойт.")
     print("Бул жакшы, анткени алардын бири чындыгында сасык!")

# Эгерде эки шарт тең аткарылбаса, анда төмөндөгү else иштейт
# Эгерде шарттардын бири дагы туура болбосо, else иштебейт
else:
     print("Бул уят, анткени алардын бири чындап эле сасык!")
```

Бул мисал программа бир же эки шарттын туура же туура эмес экендигин текшерүү үчүн иштелип чыккан. Эгерде экөө тең туура болсо, анда биздин print() функциябыз иштей берет. Бир гана шарт туура болсо - дагы деле сонун; биздин print() функцияларыбыз иштейт. Эсиңизде болсун: **or** оператору үчүн бир гана чын шарты талап кылынат.

Эки шарт тең *аткарылбас*а гана программа else операторун иштетет.

Кылдат байкоочу ошол программалардагы кичинекей көйгөйдү байкашы мүмкүн. Керемет бала бир бут кийим же жаңы кийим сатып ала аларын билсек да, кайсынысын тандай тургандыгын билбейбиз. Андан сырткары, ал экөөнү тең чогуу ала аларын билбейбиз. Экөөнүн тең алууга мүмкүнчүлүгү барбы деп гана текшердик.

Бул көйгөйдү чечүүнүн жана программабызды кеңейтүүнүн бир нече жолу бар.

Бир нерсени иретке келтирүү үчүн дагы бир нече if операторлорун кошумчаласак болот. Ошентсе да, азыр уя салуу деп аталган нерсени талкуулоо үчүн ылайыктуу учур болмок. Жок - анын канаттууларга эч кандай тиешеси жок!

Уялоо канаттууларга гана тиешелүү эмес

Кээде бир блокто бир шартты (же экөөнү) чын экендигин текшерүү жетишсиз. Мисалы, эгерде биринчи шарт чын деп бааланса, экинчи же үчүнчү (же төртүнчүсү ж.б.) шарттын аткарылышын текшерип көрүшүбүз мүмкүн. Керемет бала жаңы коргоочу кийим жана бут кийим сатып ала алабы же жокпу аныктай турган биздин кодду карап көрүңүз. Керемет бала ошол буюмдардын бирин сатып ала аларын билебиз. Бирок анын экөөнү тең сатып ала алабы, анын акчасы бар-жок экенин билбейбиз. Ошондой эле, ага коргоочу кийим керекпи же бут кийимби, кайсынысы көбүрөөк керек экендигин да билбейбиз.

Ушул суроолордун айрымдарына код менен жооп бере алабыз. Башкача айтканда, жоопторду алуу үчүн алгоритм колдонсок болот. Биз бир нече билдирүүлөрдү бир if билдирүүсүнүн ичинде текшерсек, анда аны **уя салуу** деп атайбыз.

if билдирүүсүн колдонгонубузда код кантип автоматтык түрдө чегинип кетерин байкадыңыз. Биз кош чекит (:) коюп, enter баскычын басканыбызда, иштөө мейкиндиги кийинки сапка түшүрүп, андан кийин сегиз боштук чегиндирет. Бул бизге, программист катары, көрүнүп тургандай, чегинген код анын жогорудагы if билдирүүсүнүн бөлүгү болуп саналарын көрсөтөт. Ошондой эле программага дагы ушул эле нерсе айтылат. Бул **код иерархиясы** деп аталат, анда (1) бул кодду ошол коддон мурун аткарат жана (2) бул чегинген код жогорудагы кодго таандык.

Уялоо кандайча иштээрин жакшыраак түшүнүү үчүн мурунку мисалыбызды кайрадан иштеп чыгалы:

```
# Текшерилүүчү өзгөрмөлөр
кереметБала = 20
жаңыКийим = 20
жаңыБутКийим = 50

# Жаңы коргоочу кийимди сатып алууга мүмкүнчүлүгүңүз барбы же жокпу
#текшерилет
if кереметБала >= жаңыКийим:
      print("Сиз жаңы коргоочу кийимди сатып ала аласыз.")
      print("Бирок, жаңы бит кийимчи?")

# Сиздин мүмкүнчүлүгүңүз бар-жогун текшергенде
     if кереметБала >= жаңыБутКийим:
         print("Жаңы бут кийим сатып алууга да мүмкүнчүлүгүз бар
          окшойт.")
         print("Бул жакшы, анткени алардын бири чындыгында сасык!")
         print("Бирок, сиз бир учурда экөөнү тен ала аласызбы?")
```

```
# Эгерде сиз бут кийимди ала албасаңыз, бирок коргоочу кийимди сатып алсаңыз, анда ал муну дагы жасайт:

    else:
        print("Кечирексиз, болгону жаңы коргоочу кийим гана сатып ала аласыз.")

# Эгерде эки шарт тең аткарылбаса, анда төмөндөгү else башталат
# Эгерде шарттардын жок дегенде бири чын болсо, анда бул else иштебейт
else:
    print("Бул уят, анткени алардын бири чындап эле сасык!")
```

Жаңыртылган мисалда байкалчу биринчи нерсе - эгерде деп аталган биздин **if** билдирүүбүздүн чегинүүсү. Биринчи if Керемет Бала жаңы коргоочу кийим сатып ала аларын текшерип көрөт.

Ал жасай алгандыктан (*кереметБала* өзгөрмөсүнүн мааниси *жаңыКийим* өзгөрмөсүнүн маанисинен чоң же барабар дегенди билдирет), программа чегинген - же уяланган - **if** шартына өтөт. Программа дагы бир жолу if уялардын шарттарынын чын экендигин текшерет (бул *кереметБаланын мааниси жаңыБутКийим*дин маанисинен чоң же ага барабар дегенди билдирет). Эгер ошондой болсо, анда ал чегинген print() функцияларын аткарат.

Эскертүү. Уялаган if билдирүүсүнөдөгү print() функциялары да чегингендигин эске алыңыз.

Бул учурда уядагы **if** билдирүүсү жалган, ошондуктан уядагы чегинген **else** оператору иштейт.

If билдирүүсүнүн түп нускасы – false-жалганды кайтарганда гана, төмөнкү бөлүгүндөгү else билдирүүсү иштейт. Бул программанын натыйжасы кандай?

```
Сиз жаңы коргоочу кийимди сатып ала аласыз.
Бирок, жаңы бит кийимчи?
Кечирексиз, болгону жаңы коргоочу кийим гана сатып ала аласыз.
```

Эгерде сизде экиден ашык **if** болсо, эмне болмок? Мындай болгондо, ар бир кошумча **if** үчүн **elif** колдонуш керек. Келгиле, **if** билдирүүсүнүн уялаган күчүн чындап көрсөтүү үчүн математиканын жөнөкөй мисалын колдонолу. SuperHeroQuiz.py аттуу жаңы файл түзүп, ушул кодду киргизиңиз:

```
# КереметБаланын тест жыйынтыгын чагылдырган өзгөрмө
кереметБалаБаллы = 100
```

```
# Кириш текст

print("Супер-баатырдын интеллектуалдык викторинасын/Акылга сыярлык тестти аякташыңыз менен куттуктайм!")

print("Же, кыскартканда С.Б.И.В.А.С.Т.")

print("Сынактан өткөнүңдү же кулаганыңды көрөлү!")

print("Өтө турган балл дос болууга мүмкүндүк берет.")

# Керемет Бала өзүнүн С.Б.И.В.А.С.Т. тапшыргандыгын текшерүү үчүн
#салыштыруу кутучасы. Сынак.

if кереметБалаБаллы > 60:
      print("Бул сенин сынагыңдын жыйынтыгы: ")

      if кереметБалаБаллы > 60 and кереметБалаБаллы < 70:

         print("Мм...,сен араң өттүң!")

      elif кереметБалаБаллы >= 70 and кереметБалаБаллы < 80:

         print("Өттүң... орточо балл начар эмес. Жакшы аракет кылып
      окуйсуң деп ойлойм.")

      elif кереметБалаБаллы >= 80 and кереметБалаБаллы < 90:

          print("Оо, жакшы! 4 деген баага татыдың!")

      elif кереметБалаБаллы >= 90:

         print("Карачы! Классындагы эң жогорку балл. Мен ушуга чейин
         көргөн эң жаш С.Б.И.В.А.С.Т. тапшыруучусу болдуң!")

else:
      print("Аябай жакшы аракет болду, бирок, кечирип коюңуз, өтпөй
      калдыңыз.")
      print("Блитз Бургерине күзөтчү керек деп уккам – курал
      колдонгонду билесизби?")
```

Бул сценарийде Керемет Баланын баллы жетиштүү эмес. Жетиштүү болуу үчүн ал С.Б.И.В.А.С.Т тапшырыш керек. Сынакта 60тан жогору балл алгандар гана өтө алат.

Керемет Бала сынактан өткөн-өтпөгөнүн билүүдөн тышкары, ага тесттин жыйынтыгы боюнча бир аз пикирибизди билдиргибиз келет. Ар бир 10 баллдык диапазон үчүн **if / elif** билдирүүсүн түздүк. Анда Керемет Баланын алган баллына жараша айрым тексттер басылып чыгат.

Керемет Бала сынактан өтпөй калса (ал 60тан төмөн балл алып калса), анда if / elif билдирүүлөрүнүн бардыгы иштебей, анын ордуна else ишке киргизилет.

Маанилүү эскертүү: **if** билдирүүнүн биринчи шарты аткарылбаса, башка шарттардын бири дагы каралбайт; анын ордуна, программа автоматтык түрдө **else** билдирүүсүнө өтөт. Программа биринчи **if** шартына туш болгондо, *кереметБалаБаллын* текшерип, анын 60тан чоң экендигин сурайт. Эгерде андай болбосо, программа аяктап, **else** билдирүүсүн аткарат.

Программанын натыйжасы

```
Супер-баатырдын интеллектуалдык викторинасын/Акылга сыярлык тестти
аякташыңыз менен куттуктайм!
Же, кыскартканда С.Б.И.В.А.С.Т.
Сынактан өткөнүңдү же кулаганыңды көрөлү!
Өтө турган балл дос болууга мүмкүндүк берет.
Бул сенин сынагыңдын жыйынтыгы:
Карачы! Классыңдагы эң жогорку балл. Мен ушуга чейин көргөн эң жаш
С.Б.И.В.А.С.Т. тапшыруучусу болдуң!
```

кереметБалаБаллы маанисин бир нече жолу өзгөртүп, программаны кайра иштетип, натыйжалардын кандай өзгөрөрүн билип алыңыз.

Бул эпизоддо

Бул кызыктуу эпизод жаңы маалыматтарга толуп кетти. Ушул кезге чейин бардык бөлүмдөрдүн ичинен бул бөлүм сиздин күчүңүздү эң жогорку деңгээлге арттырды деп айткым келет! Сиздин супер мээңиз, кыраакы түшүнүгүңүз жана супер баатырды окуган жөндөмүңүз менен тамашалар аркылуу ушул убакка чейин бардык маалыматтарды сиңирип алдыңыз деп толук ишенем.

Сиз аны жакшылыкка же жамандыкка колдоносузбу? Убакыт көрсөтөт! Ата-энеңиз укмуштуудай жазуучу Джеймс Пейн жазган бул кереметтүү китептин өзгөчөлүгү эмнеде жана эмне үчүн аны окубай коё албайсың деп сураганда, бул тууралуу кыскача айтып берсеңиз болот!

- Чечим кабыл алуу - бул белгилүү бир критерийлердин негизинде программанын тигил же бул жолду улантуу жөнүндө чечим кабыл алышы керек болгон процесс.
- Псевдокод - программанын бөлүмдөрүн сүрөттөө үчүн колдонулган жасалма тил. Бул - макет жана программанын ар кандай бөлүктөрүн жакшыраак түшүнүү үчүн программаларды түзүүнүн стенографиясы.

- Шарттуу операторлор, эгер белгилүү бир шарттар аткарылса же аткарылбаса, программаңыздын тигил же бул бөлүгү боюнча иш алып барууга мүмкүнчүлүк берет. Алар: **if, else** жана **elif**.
- **if** шарты программаңызда чечим чыгарууга мүмкүндүк берет. Мисалы, **if** "x" чыкса, сиз программаны коддун фрагментин аткарууга мажбурлай аласыз.

  Мисалы:

```
if 10 < 20:
     print("Ооба, 10 саны 20 санынан кичине")
```

- **Else** пунктун кошуу менен **if** билдирүү блогу дагы да татаалдашат. Мисалы, **if** "x" болуп калса, сиз программанын кодунун үзүндүсүн аткара аласыз же эгерде if "x" *чыкпай калса*, анда сиз аны башка код блогунда аткара аласыз.

  Мисалы:

```
if 10 < 20:
     print("Ооба, 10 саны 20 санынан кичине")
 else:
     print("Математика өтө татаал! Сандар мээге оор келет!")
```

- **Else if / elif** сиздин кодуңузга кошумча шартты кошуу үчүн колдонулат.

  Мисалы:

```
if 10 < 20:
     print("Ооба, 10 саны 20 санынан кичине")
elif 10 == 20:
     print("10 саны 20 санына барабар болбошу керек, бирок, сен
     айтып  жатсаң!")
else:
     print("Биздин түшүнүгүбүздө 10 саны 20 санынан чоң!")
```

- Салыштыруу операторлору өзгөрмөлөрдүн маанилерин салыштырууга мүмкүндүк берет. Алар төмөнкүчө: (==) барабар, (!=) га тең эмес, (<) кичине, (>)чоң, (<=) кичине же барабар жана (>=) чоң же барабар.
- Логикалык операторлор бир нече шарттарды текшерүүгө мүмкүнчүлүк берет. Алар: **and** - жана, **not** - эмес, **or** - же.

5 - БӨЛҮМ

ЦИКЛДЕР ЖАНА ЛОГИКА

Кээде кылмыштуулукка каршы күрөшүүдө сиз өзүңүздү кайра-кайра эле бир жол менен айланып чуркап жүргөндөй сезишиңиз мүмкүн. Күнү-түнү окшош эле көйгөй менен алпурушуп жаткандай болушуңуз мүмкүн. Бул жердеги банк каракчысы, тигил жакта бактын коңулуна кептелип калган мышык, ааламды басып алууга аракет кылган каардуу гений ж.б. менен алпурушуп жүргөндөйсүз. Сиз баш-отуңуз менен дээрлик бир циклге кептелип калгандайсыз.

“Суур Күнү” фильминдеги Билл Мюррейдин узак убакыт бою кайталана берген узун күнү (ата-энеңизден сураңыз) сизге жакпашы мүмкүн, бирок компьютериңиздин программаларында циклдерди колдонуу сиз үчүн *эң сонун* нерсе болушу мүмкүн. Компьютердик программалардын эң негизги максаттарынын бири - күн сайын кайталана турган тапшырмаларды кайталап туруу. Өз программаларыбызды кулдай иштетип, аларды ушул түйшүктүү тапшырмаларды аткарышыбыз үчүн колдонуучу ыкмалардын бири *цикл* деп аталат.

Циклдер сиз буга чейин болжолдогондой, белгилүү бир шарт туура болсо, коддун үзүндүсүн улам-улам кайталап турушат. Шарттуу операторлор сыяктуу (4-бөлүмдө камтылган) циклдер - программисттин муктаждыгына жараша аткарылышы же аткарылбашы үчүн шарт. Ал аткарылышы керек же аткарылбашы керек.

Циклдердинн бир нече түрлөрү бар жана алардын ар бир түрүн ушул укмуштуу окуялуу бөлүмдө чагылдырабыз. Ошентип, биз циклдешүүгө даярданып жаткандай эле, өзүңүздү жаш баатыр болууга даярдаңыз!

Циклдер деген эмне?

Программист катары биздин жалпы максаттарыбыздын бири - натыйжалуу код жазуу. Жасаган аракеттеребиздин баары колдонуучунун натыйжалуу колдонуусун камсыз кылууга, процессордук ресурстарды азайтууга жана программаларды эң аз код менен түзүүгө багытталышы керек.

Буга жетишүүнүн бир жолу - бул Python программалоо тилиндеги эки түрдүү циклдерди колдонуу. Ушул бөлүмдүн кириш сөзүндө айтылгандай, циклдер - бул биз аныктаган шарт аткарылганга чейин бир нече жолу коддун бир бөлүгүн кайталоого мүмкүндүк берген сыйкырдуу жандыктар.

Программалоодо цикл кандайча иштей тургандыгы жөнүндө бир мисал - сиз колдонмо түзгөндө сиз ойлогон санды кимдир бирөө болжолдой ала тургандыгы. Код колдонуучудан цифраны туура божомолдогонго чейин улам сурай берет.

Болжолдонгон цифра туура болгондо, циклдан чыгып программанын калган бөлүгү аткарылат.

Адаттагыдай эле, *Үрөй учурган бөлмөгө* бет алалы жана жаңы кодуңузду сынап көрөлү. Биринчиден, SinisterLoop.py файлын түзүп, төмөнкү кодду кошуңуз:

```
# Бош өзгөрмө түзүңүз. Ал жерге маалыматтарды кийинчерээк сактайбыз.
болжолдонгонСан = ' '

# колдонуучу 42 санын киргизгенге чейин улана турган while циклин түзүңүз
while болжолдонгонСан != '42':
    print("Синистер Луп сиздин алдыңызда турат!")
    print("Мен ойлогон санды тапсаңыз, мени кармоого уруксат берем!")
    print("0 менен 4түн ортосундагы сандарды колдонуп каалагандай сан
    жазыңыз:")
    болжолдонгонСан = input()
    # Колдонуучунун терген санын болжолдонгонСан өзгөрмөсүнө сактайт
print("Синистер Луп таң калып кыйкырат!")
print("Болжолдонгон сан " + болжолдонгонСан + " экенин кантип таптыңыз?")
```

Бул коддун сценарийинде каардуу Синистер Луп биздин каарман Керемет Бала менен бетме-бет келип, аны каардуу адамдын оюндагы каардуу санды айтууга мажбурлайт. Керемет Бала ийгиликке жетишсе, Синистер Луп өзүн колго түшүрүүгө мүмкүнчүлүк берет. Эгерде, ал санды айта албасачы? Эгер айта албаса, анда ал сизден улам-улам бир номерди киргизүүнү сурана берет. Кызыктуу бекен?

Бул коддук мисалдан бир нече жаңы нерселерди үйрөндүк. Биз буга чейин жасай элек нерсени жасап баштайбыз - *болжолдонгонСан* деген бош өзгөрмөнү түздүк. Бул өзгөрмөнү бош калтырып жатабыз, анткени кийинчерээк колдонуучу аны маалымат менен толтурушу керек.

Андан кийин, while loop - while цикли деп аталган коддордун блогун түзөбүз. Сап:

```
while болжолдонгонСан != '42':
```

программага *болжолдонгонСан* өзгөрмө мааниси - "42" ге *барабар эмес*, же != болгон учурда иштешин айтат. Андан кийин биз бир нече сап текстти басып чыгарып, колдонуучудан сан киргизүүнү суранабыз. Санды топтой турган чыныгы код сабы:

```
болжолдонгонСан = input()
```

Input() функциясы print() функциясына окшош. Ал болгону колдонуучудан алынган киргизүү же маалыматтарды кабыл алат. Бул киргизүү колдонуучунун клавиатурасындагы баскычтарды басуу менен жүргүзүлөт. Программага

киргизилген маалыматтар (=) операторунун сол жагындагы өзгөрмөдө сакталат - бул учурда ал *болжолдонгонСан*.

Программанын иштешин көрүү үчүн, кодду иштетип, 42 санын терүүдөн мурун ар кандай сандарды бир нече жолу терип көрүңүз.

Баарынын башын айландырып бүтүштүбү? Жакшы*.*

Кыскача эскертүү: бул мисалда while критерийлери үчүн биз барабар эмес (! =), операторун колдондук. Сиз эмне үчүн анын ордуна барабар же == операторун колдонгон жокпуз деп сурап жаткан чыгарсыз. Себеби, биз бир нерсе туура эмес болуп жаткан учурда программадагы циклдин иштешин каалап атабыз. Маани 42ге барабар болгондо, эгер, анын ордуна == операторун колдонуп, программада цикл колдонсок, биз олуттуу циклдик логикалык ката кетирген болот элек.

Бул жерде циклдер кооптуу болуп калышы мүмкүн. Эгер, өзгөрмөнүн мааниси 42ге барабар болгон учурда, биз программага цикл деп атай турган болсок, анда программа эч качан биздин циклди аткара алмак эмес.

Эмне үчүн? *болжолдонгонСан* 42 болгондо биз аны циклдеп койгонбуз. Бирок, биз эч качан *болжолдонгонСан* маанисин койбогондугубуз эсиңизде болсун. Демек, Python мааниси 42 экендигин текшерүүгө барганда, ал циклде эместигин аныктайт жана ал циклдан чыгат. Анткени маани 42 болсо гана цикл иштейт!

Эгер сиз бул нерсени татаал деп эсептесеңиз, анда муну карап көрүңүз: эгер *болжолдонгонСан* маанисин 42 деп дайындап коюп, while циклинин шартын == 42 кылып койсок эмне болмок?

Бул сценарийде программа түбөлүккө иштей бермек. Эмнеге? Себеби биз муну "*болжолдонгонСан*дын мааниси 42 болсо, ушул кодду карап чыгыңыз" деп айтып жатабыз."Бул коркунучтуу чексиз цикл деп аталган нерсе жана ал - ар бир программисттин жашоосундагы азабы. Кызыктуу болуш үчүн InfiniteLoop.py аттуу жаңы файлды түзүп, төмөнкү кодду киргизели:

Эскертүү. Бул программаны иштеткенде чексиз цикл пайда болот. Андан чыгуу үчүн IDLE терезеңизди жаап, IDLE ды өчүрүп, кайрадан күйгүзүшүңүз керек болот.

```
# Мааниси 42 болгон өзгөрмө түзүңүз.

болжолдонгонСан = 42

print("Синистер Луп сиздин алдыңызда турат!")
print("Менин чексиз циклимди кара!")

# болжолдонгонСандын мааниси 42 болсо, улана турган while циклин түзүңүз.
while болжолдонгонСан == 42:
      print("Луп!")
```

Эмне болуп жатканын көрүү үчүн кодду иштетиңиз. Куттуктайбыз, сиз биринчи чексиз циклиңизди жараттыңыз! Эми, экинчи мындай кылбаңыз!

Бир аз башкача аракет жасап көрөлү. Биздин баалуулук үчүн сандын ордуна текстти колдонолу. WonderBoyPassword.py деген дагы бир жаңы файл түзүп, ушул кодду териңиз:

```
# Керемет Баланын сыр сөзүн сактоо үчүн өзгөрмө түзүңүз
password = ''
print("Атанын гаджеттерди сактаган жайына кош келдиңиз!")

while сырСөз != "кереметбала2018":
    print("Сураныч, кээ бир кызыктуу технологияларга жетүү үчүн сыр сөз киргизиңиз!")
    сырСөз = input()
print("Сиз туура сыр сөз тердиңиз!")
print("Сизге канча гаджет керек болсо ошончо ала бериңиз!")
print("Canon гаджетине тийбеңиз - бул Атага таандык!")
```

Бул код сиз күткөндөй иштейт. 42 санын колдонгон мисалыбыз сыяктуу эле, бул программа бош өзгөрмөнү түзүп, кириш текстин басып чыгарат. Андан кийин биз сыр создүн маанисин ("кереметбала2018") кирмейинче иштей бере турган while циклин түзөбүз. Колдонуучу ошол өзгөчө маанини киргизгенден кийин программа циклден чыгып, басып чыгаруу print билдирүүсүнө өтөт.

Бирок, бул жерде бир аз айырмачылык бар. Берилиштердин түрлөрүн санабай, текст менен иштегендиктен, киргизилген маани шарт менен бирдей болушу керек. Башкача айтканда, сыр сөз сөзсүз "кереметбала2021" болушу керек. Чоң тамгалар чоң тамга менен кичине тамгалар кичине тамга менен жазылышы керек.

Эмне үчүн? Өтө кызыксыз деталдарга кирип кетпестен, программаңыздагы ар бир каарманга берилген өзгөчө маани бар экендигин билиңиз. Эсиңизде болсун, компьютер текстти көрбөйт, тескерисинче, машина тилине которулган 1 жана 0 сериялярын көрөт. Ушундан улам, компьютерлер "H" жана "h" эки бөлөк нерсе деп эсептешет.

Программаны иштетип, "Кереметбала2018" же "КЕРЕМЕТБАЛА2018" деп терип көрүңүз да, эмне болуп жаткандыгын байкап көрүңүз. Көрүнүп тургандай, программа while циклин улантып, сыр сөздү сурай берет. Сиз "кереметбала2018" сыр сөзүн киргизгенде гана программа циклден чыгат.

Ушул түрдөгү "цикл логикасы" менен циклдерди коддоо таптакыр кадыресе көрүнүш. Чындыгында, сыр сөздөр же коопсуз маалымат жөнүндө сөз болгондо, программа дал ушундай жол менен жүрүшү керек. Бирок сиз баш тамга менен жазууга маани бербесеңиз, кандай болуп калышы ыктымал? Киргизүүнү баш тамга менен жазылуусуна карабастан иштешин кааласаңызчы?

Буга жетишүүнүн бир нече жолу бар. Алардын бири текстти кичинекей тамгаларга которууну камтыйт. Бул үчүн сиз str.lower() деп аталган жаңы сап функциясын колдоносуз. WonderBoyPassword.py кодун төмөнкүдөй өзгөртүңүз:

```
# Керемет Баланын сыр сөзүн сактоо үчүн өзгөрмө түзүңүз
сырСөз = ''
print("Атанын гаджеттерди сактаган жайына кош келдиңиз!")
while сырСөз != "кереметбала2018":
    print("Сураныч, кээ бир кызыктуу технологияларга жетүү үчүн сыр
    сөз киргизиңиз!")
    сырСөз = input()
    сырСөз = сырСөз.lower()
print("Сиз туура сыр сөз тердиңиз!")
print("Сизге канча гаджет керек болсо ошончо ала бериңиз!")
print("Canon гаджетине тийбеңиз - бул Атага таандык!")
```

```
Бул кодубузда мурунку үзүндүбүзгө жаңы сапты коштук:

сырСөз = сырСөз.lower()
```

Бул код тилкеси *сырСөз* өзгөрмөсүнө берилген маалыматтарды алып, кичине тамгага айлантат. Ошентип, цикл сыр сөздүн туура же туура эместигин текшергенде, колдонуучу тамгалардын бирин баш тамга менен жаздыбы же жокпу деп тынчсыздануунун кажети жок.

Эскертүү. Бардык *сап* түрүндөгү маалыматтарды кичине тамга менен белгилөө үчүн, str.lower() колдоносуз. Мисалы: сап.lower(). Сапты чоң тамга менен жазуу үчүн, str.upper() колдоносуз. Мисалы: *сырСөз*.upper()

Циклдерди чектөө

Биздин циклдердин чексиз иштешине жол берсек да, алардын иштөө убактысынын санын чектеп койгубуз келет. Мисалы, биздин WonderBoyPassword.py кодубузда колдонуучу сыр сөздү канча жолу кааласа, ошончо жолу табууга аракеттене алат. Программа туура сыр сөз берилгенден кийин гана циклден чыгат. Бирок, бул программа жазуунун эң мыкты ыкмасы эмес.

Сыр сөздөр менен иштөөдө же циклдин канча жолу кайталанышын чектөө керек болгондо - көрсөтүлгөн критерий аткарылса, циклди токтото турган шарт түзө аласыз.

Колдонууну үзгүлтүккө учурашын көрүү үчүн WonderBoyPassword.py дарегиндеги кодду төмөнкүлөргө дал келтирүү үчүн түзөтүңүз:

```
# Керемет Баланын сыр сөзүн сактоо үчүн өзгөрмө түзүңүз
сырСөз = ''
сырСөзСаны = 0
print("Атанын гаджеттерди сактаган жайына кош келдиңиз!")
while сырСөз != "кереметбала2018":
      print("Сураныч, кээ бир кызыктуу технологияларга жетүү үчүн сыр
      сөз киргизиңиз!")
      сырСөз = input()
      сырСөз = сырСөз.lower())
      сырСөзСаны = сырСөзСаны + 1
      if сырСөз == "кереметбала2018":
          print("Сиз туура сыр сөз тердиңиз!")
          print("Сизге канча гаджет керек болсо ошончо ала бериңиз!")
          print("Canon гаджетине тийбеңиз - бул Атага таандык!")
      elif сырСөзСаны == 3:
          print("Кечиресиз, аракетиңиз туура болгон жок!")
          break
```

WonderBoyPassword.рунун бул версиясында биз бир нече жаңы коддорду коштук. Жаңы баштап жаткандар үчүн биз *сырСөзСаны* деген жаңы өзгөрмөнү аныктадык жана ага 0 маанисин бердик. Бул өзгөрмө сыр сөздү табууга жасалган аракеттердин санын байкоо үчүн колдонулат. Колдонуучу туура эмес божомолдогон сайын ушул коддун жардамы менен цикл кайталанат:

```
сырСөзСаны = сырСөзСаны + 1
```

сырСөзСаны маанисине 1ди коштук. Андан кийин 2ни коштук. Биринчиси, колдонуучу туура сыр сөздү тапса, айрым тексттерди басып чыгарат. Elif оператору *сырСөзСаны* мааниси 3кө барабар болгондо (үч жолу аракет кылгандан кийин) башталат. Кийинчерээк ал кечирим сураган текстти басып чыгарып, андан кийин while циклинен чыгуу үчүн break операторун колдонот.

Кодду бир нече жолу байкап көрүңүз, сыр сөздү жок дегенде үч жолу туура эмес киргизип, жок дегенде бир жолу так тапсаңыз болот.

For цикли

Циклдин бир нече жолу кайталангандыгына тастыктоонун дагы бир жолу - for циклин колдонуу. Циклдин мындай түрү кандайдыр коддун үзүндүсүн канча жолу кайталоо керектигин чектөөдө колдонулат.

For циклин киргизүү үчүн белгилүү ыкма - бул сандар тизмеси боюнча эсептөө программасын түзүү. Мисалы, 1-10 арасы. Бирок, биз катардагы программисттер эмеспиз да, биз кокустан код түзгөн супер баатырларбыз. Ошондуктан, бизге программанын өзгөчө түрү керек. Count10.py мисалы!

```
print("Синистер Луп, ал жакта экениңизди билем!")

print("Эгерде мен 10го чейин санагыча кафетериядагы мыздаткычтан чыкпасаңыз...")

print("Сиз ошол даамдуу стоп белгиси түрүндөгү пиццаны ала албайсыз!")

for x in range(1,11):

    print(x)

print("Мен сага эскерткем! Азыр пиццанын бардыгы Керемет Балага таандык!")
```

Бул коддун маанилүү бөлүгү ушул жерде:

```
for x in range(1,11):

    print(x)
```

For циклды баштайт. x өзгөрмөсү (бул өзгөрмө каалагандай аталышы мүмкүн, адатта, программисттер аны 'i' же 'x' деп аташат) кайталануу санынын маанисине ээ болот. Чындыгында, ушул жол колдонулганда ал *кайталануучу өзгөрмө* катары белгилүү. Андан кийин колдонула турган *ырааттуулукту* айтып берүү үчүн range функциясын колдонобуз. Ырааттуулук сан аралыгында же тексттин жардамы менен түзүлүшү мүмкүн (кененирээк төмөндө).

Кашаанын ичиндеги сандар функциянын башталыш жана токтоо параметрлери болуп саналат. Мисал келтирүү үчүн биз 10го чейин эсептегибиз келет. Андыктан баштоону 1ге, ал эми токтогонду 11ге коёбуз. range 10до токтошун кааласак да, мында акыркы сан катары 11ди жазышыбыз керек. Анткени range биздин акыркы чекитке чейинки санда токтойт. Эгерде биз Синистер Лупко ошол муздаткычтан чыгып келишине узак убакыт бергибиз келсе, анда баштоо 0до, токтогондо 10до, ал эми башталышы 12де, токтошу 1.000.000де болушу мүмкүн болмок (бирок, анда ал тоңуп өлүп калмак!).

Соңунда кайталоонун канча жолу болгонун басып чыгаруу үчүн x өзгөрмөсүн басып чыгарабыз. 10го жеткенден кийин, программа for циклинен чыгып, коддун кийинки бөлүгүнө өтүп кетет да, ал акыркы басылма болуп калат. Эгер сиз программаны иштетсеңиз, анда төмөнкүдөй жыйынтыкка жетесиз:

```
Синистер Луп, ал жакта экениңизди билем!
Эгерде мен 10го чейин санагыча кафетериядагы мыздаткычтан чыкпасаңыз...
Сиз ошол даамдуу стоп белгиси түрүндөгү пиццаны ала албайсыз!
1
```

```
2
3
4
5
6
7
8
9
10
Мен сага эскерткем! Азыр пиццанын бардыгы Керемет Балага таандык!
```

Эгер биз кодду бир аз татаалдаткыбыз келсе, анда for циклинин курамына кирген текстти print билдирүүсүнө кошуп алсак болот, мисалы:

```
print("Синистер Луп, ал жакта экениңизди билем!")

print("Эгерде мен 10го чейин санагыча кафетериядагы муздаткычтан чыкпасаңыз...")

print("Сиз ошол даамдуу стоп белгиси түрүндөгү пиццаны ала албайсыз!")

for x in range(1,11):
      print(x, "Миссисипи")
print("Мен сага эскерткем! Азыр пиццанын бардыгы Керемет Балага таандык!")
```

Бул жерде болгон өзгөрүү:

print(x) коду print(x, "Миссисипи") менен алмаштырылды

Бул бизге мындай жаңы натыйжаны берет:

```
Синистер Луп, ал жакта экениңизди билем!
Эгерде мен 10го чейин санагыча кафетериядагы мыздаткычтан чыкпасаңыз...
Сиз ошол даамдуу стоп белгиси түрүндөгү пиццаны ала албайсыз!
1 Миссисипи
2 Миссисипи
3 Миссисипи
4 Миссисипи
5 Миссисипи
6 Миссисипи
7 Миссисипи
```

```
8 Миссисипи
9 Миссисипи
10 Миссисипи
Мен сага эскерткем! Азыр пиццанын бардыгы Керемет Балага таандык!
```

Бул биз жерде бар болгону "Миссисипи" деген сөздү басып чыгаруу үчүн коштук.

Өсүү ырааттуулугу менен эсептөөдөн тышкары, range терс ырааттулук менен эсептөө мүмкүнчүлүгүнө да ээ. Ал үчүн биз step деп аталган түшүнүктү колдонушубуз керек. Step - бул range() функциясынын милдеттүү эмес параметри жана сандарды өсүү же кемүү ырааттуулугу үчүн колдонулат. Мисалы, биз 10дон 1ге чейин эсептеп көргүбүз келсе, for циклин төмөнкүдөй жазмакпыз:

```
for x in range(10,0, -1):
     print(x)
```

Циклдин -1 бөлүгү step болуп саналат жана негизинен программага ар бир жолу бирден кемитүүнү айтат. Эгер сиз бул кодду иштеткен болсоңуз, анда төмөнкүлөргө алып келиши мүмкүн:

```
10
9
8
7
6
5
4
3
2
1
```

Эгерде биз -2 step жасасак, ар бир жолу 2ден кемитүү менен, эсептөө кемүү ырааттулугунда болмок.

```
for x in range (10,1 -2):
   print(x)
```

Анда жыйынтык:

```
10
8
6
4
2
```

Эгерде биз 2 step менен өсүү ырааттулугунда эсептегибиз келсе, анда + белгисин кошуунун кажети жок - биз 2ден кошууну төмөнкүдөй жасайбыз:

```
For x in range(1,10,2):
      print(x)
```

Анда жыйынтык:

```
1
3
5
7
9
```

For циклы менен иштөө мындан да кызыктуу

Албетте, for циклы сандарды басып чыгаруу үчүн гана колдонулбайт. Жогоруда айтылгандай, биз кодду бир нече жолу кайталоону кааласак, for циклы биздин эң мыкты тандообуз бойдон кала берет.

Мисалы, биз тажатма, бир эле текстти экранда бир нече жолу басып чыгаргыбыз келет. Биздин досубуз for циклы буга жардам бере алат!

```
for x in range(1,10):
      print("Керемет")
print("Бала!")
```

Бул кичинекей үзүндү, циклден чыгып кетерден мурда "Керемет" деген сөздү тогуз жолу басып чыгарып, аягында “Бала” деп бүтүрөт. Эгер сиз буга сонун темадагы музыканы койсоңуз, анда сиз өзүңүздүн телесериалдарыңызды тартууга даяр болмоксуз!

Цикл үчүн колдоно турган дагы бир жол - бул тизмелерди кайталоо. Бизде шумпай, каардуу адамдардын тизмеси бар. Келгиле, алардын аттарын басып чыгарып көрөлү. Бул үчүн биз төмөндөгүдөй кодду колдонсок болот:

```
кара Ниеттүүлөр = ['Синистер Луп', 'Жазалоочу', 'Джек Хаммер', 'Тоңдургуч', 'Жалтылдак Түш']
print("Мына сизге алып келген бүгүнкү каардуу адамдардын тизмеси:")
print("Баатырдык Курулуш Компаниясы. Сен шаарды жок кыласың, биз аны дээрлик жаңы түргө айлантабыз.")
print("Эң Жаман Кара ниеттер:")

for шумпай in кара Ниеттүүлөр:
    print(шумпай)
```

Бул жерде биз тизмени түзөбүз (биз алгач 3-бөлүмдө тизмелерди талкуулаганыбыз эсиңизде болсо керек). Андан кийин биз тизмени жаман адамдардын ысымдары менен толтурдук. Кийин биз бир нече текстти басып чыгарып, андан соң for циклин жасадык:

```
for шумпай in кара Ниеттүүлөр:
    print(шумпай)
```

Бул жолу, for шумпайдын аты менен өзгөрмө түзүүдөн баштайт, анын жумушу биздин тизмедеги ар бир маанинин маанилерин (убактылуу) кармап туруу болуп саналат.

Биз range функциясын аныктай элек болгондугубуздан улам, цикл *кара Ниеттүүлөр* тизмесиндеги ар бир маани үчүн бир гана жолу иштейт. Ал аркылуу ар бир жолу, *шумпай* өзгөрмөсүнүн маанисине башкача маани берилет. Мисалы, биринчи жолу 'Синистер Луп' *шумпай* адамдарга өтүп, андан кийин басылып чыгат. Андан кийин цикл экинчи жолу уланып, 'Жазалоочу' *шумпай* адамдарга өткөрүлүп берилет. Ал дагы бир жолу басылып чыгарылат. Бул цикл тизмедеги бардык маанилер аркылуу өткөнгө чейин уланат. *Шумпай тизмесинин* акыркы мааниси "Жалтылдак түш" болот. Сиз маалыматты өзгөртмөйүнчө анын мааниси ошол бойдон калат.

Эгер биз ушул кодду иштете турган болсок, анда натыйжасы мындай болот:

```
Мына сизге алып келген бүгүнкү каардуу адамдардын тизмеси:
Баатырдык Курулуш Компаниясы. Сен шаарды жок кыласың, биз аны дээрлик жаңы түргө айлантабыз.
Эң Жаман Кара ниеттер:
Синистер Луп
Жазалоочу
Джек Хаммер
Тоңдургуч
Жалтылдак Түш
```

Break, continue, жана pass билдирүүлөрү

Циклдер коддун бөлүктөрү аркылуу кайталоо үчүн колдонулганына карабастан, кээде циклди эрте бүтүрүп, циклдин бир бөлүгүн өткөрүп жиберүү же циклдерибизге кирбеген айрым маалыматтарды иштетүү ыкмасы керек деп ойлошубуз мүмкүн. Мындай аракеттерде бизге жардам бере турган үч оператор бар: break - бузуу, continue - улантуу жана pass - өткөрүп жиберүү.

Break жөнүндө сөз кыла турган болсок, бул оператор менен белгилүү бир шарттарда циклден эртерээк чыгууга мүмкүнчүлүк алабыз. Мисалы, WonderBoyPassword.py программабызда үч жолу сыр сөз болжолдоо аракети көрүлгөндөн кийин программадан чыгуу үчүн break колдондук. Бул билдирүүнү мурда чагылдырганбыз. Анда эмесе, келгиле, кийинкисине өтөлү.

Continue билдирүүсү циклдин бир бөлүгүн, чындыгында, андан толугу менен чыкпай, break билдирүүсү сыяктуу өткөрүп жиберүүгө мүмкүндүк берет. Карап көрөлү: эгерде сизде 10дон баштап эсептеген программа болсо, бирок программага текст да камтыгыңыз келсе, анда сиз continue менен аны жасай аласыз.

DoomsdayClock.py аттуу жаңы файл түзөлү. Бул программада Синистер Луп артка саноо таймерин ишке киргизди. Бирок, шумпайлар ар дайым ушунчалык тажатма болушат дейсиң, андыктан анын санак учурунда кандайдыр бир нерсени айта турганына таң калбаңыз!

Бул кодду файлыңызга киргизиңиз:

```
print("Жаман Синистер Луп сенин астында турат, ач көздүк менен колдорун ушалап!")
print("Анын колу рычагда, ал эми бетинде чоң митаамдык бар.")
print("Синистер Луп айтып жатат:")
print("'Азыр сен жеңилдиң, Керемет Бала!'")
print("'Сенде жашоого он секунд бар! Убакытты артка эсептеп жатканда ук!'")

for x in range(10,0,-1):
     print(x, "Миссисипи!")

# X 5ке барабар болгондо, бир аз текстти басып чыгарып, кайра саноону #улантыңыз.

     if x==5:
        print("'Акыркы сөзүң барбы, Керемет Бала?!?'")
        continue
print("Менин санагым 0 болгондо сөзсүз жеңилесиң...")
print("Бирок, эч нерсе болбойт!")
print("'Кайрадан үзгүлтүк!' – деп Синистер Луп кыйкырат.")
```

Мисалды иштетип, жыйынтыгын байкаңыз, ал мындай болушу керек:

```
Жаман Синистер Луп сенин астында турат, ач көздүк менен колдорун ушалап!
Анын колу рычагда, ал эми бетинде чоң митаамдык бар.
Синистер Луп айтып жатат:
'Азыр сен жеңилдиң, Керемет Бала!'
'Сенде жашоого он секунд бар! Убакытты артка эсептеп жатканда ук!'
10 Миссисипи!
9 Миссисипи!
8 Миссисипи!
7 Миссисипи!
6 Миссисипи!
'Акыркы сөзүң барбы, Керемет Бала?!?'
4 Миссисипи!
3 Миссисипи!
2 Миссисипи!
1 Миссисипи!
Менин санагым 0 болгондо сөзсүз жеңилесиң...
Бирок, эч нерсе болбойт!
'Кайрадан үзгүлтүк!' – деп Синистер Луп кыйкырат.
```

Айрым учурларды эске албаганда, бул код биздин циклдердей эле иштейт. Программа if шартына туш болору менен, x 5ке барабар экендигин текшерет. Range 10дон 0ге чейин кемүү тартибинде санап жаткандыктан, коддун бешинчи жолу кайталанышында x 5ке барабар болуп каларын билебиз. Ошондой болгондо биздин шарт аткарылат жана биз циклди натыйжалуу бир саамга тындыруу менен, циклди дагы бир жолу иштнтүүдөн мурун “Акыркы сөзүң барбы, Керемет Бала?!?”деген текстти басып чыгарабыз (биз текстти басып чыгаруу үчүн кайталоону өткөрүп жиберебиз).

Continue билдирүүсүнөн кийин программа кадимки циклди бүтүрүп, андан кийин кадимкидей чыгат.

Жыйынтыгын карасаңыз, '5 Миссисипи' сабы басылбай тургандыгын байкаңыз. Эсиңизде болсун, мунун себеби, биз циклди аттатып өткөрүп жибердик, андыктан ал сапты таптакыр басып чыгарган жок.

Азырынча, эгер, кандайдыр бир шарт аткарылса, циклден кантип чыгууну же цикл аркылуу кайталоону кантип өткөрүп жиберүүнү карадык (break жана continue). Биз үйрөнгөн кийинки бидирүү анчалык деле пайдалуу көрүнбөйт, бирок чындыгында, аны нерселердин чоң схемасы менен иштөөдө колдонуу максатка ылайык.

Pass ***класстар*** менен иштөөдө өзгөчө пайдалуу - бул теманы биз 8-бөлүмдө талкуулайбыз. Циклдер менен иштөөдө pass негизинен толтуруучу катары

колдонулат. Коддун бир бөлүгүн пландаштырып жатканыңызда, бирок сиздин критерийлер кандай болорун так билбегениңизде, бул - мыкты курал.

Мисалы, биздин DoomsdayClock.py программабызда, биздин өзгөрмө 5 саны болгондон кийин, бир нече текстти аткаруу үчүн if билдирүүсүн циклибизге киргиздик. Бирок, ал текстти эмне үчүн каалагандыгыбызды же эсептөөнүн кайсы жеринде текстти басып чыгарууну каалаарыбызды билбесек кандай болот? Балким, биз кесиптешибизден жооп күтүп, коддун ошол бөлүмүнө кайтып барышыбыз керектир.

Pass кодду бүтүрбөгөндүктөн, эгер ал аткарылса, анда эмне болуп кетерин так көрсөтпөстөн, каталар жөнүндө ойлонбостон, бизге шарттуу операторду жайгаштырууга мүмкүндүк берет. Ошентип, кийинчерээк, циклдин ушул бөлүгүндө эмне кылышыбыз керек экендигин билгенден кийин, калган кодду кийинчерээк толтура алабыз.

Эгер биз аны DoomsdayClock. py программасына киргизсек, анда pass төмөндөгүдөй иштейт:

```
print("Жаман Синистер Луп сенин астында турат, ач көздүк менен колдорун ушалап!")
print("Анын колу рычагда, ал эми бетинде чоң митаамдык бар.")
print("Синистер Луп айтып жатат:")
print("'Азыр сен жеңилдиң, Керемет Бала!'")
print("'Сенде жашоого он секунд бар! Убакытты артка эсептеп жатканда ук!'")

for x in range(10,0,-1):
     print(x, "Миссисипи!")
# X 5ке барабар болгондо, бир аз текстти басып чыгарып, кайра саноону улантыңыз.
     if x==5:
        pass
print("Менин санагым 0 болгондо сөзсүз жеңилесиң...")
print("Бирок, эч нерсе болбойт!")
print("'Кайрадан үзгүлтүк!' – деп Синистер Луп кыйкырат.")
```

Эгерде сиз ушул кодду иштетсеңиз, анда if шартына жеткенде, эч нерсе болбой тургандыгын көрө аласыз - программа кадимкидей эле иштейт. Бирок, эгерде сиз pass билдирүүсүн алып салсаңыз, анда сиз ката кетиришиңиз мүмкүн, анткени Python if шартын толтуруу үчүн көбүрөөк кодду күтүп жатат. Байкап көрүңүз, өзүңүз көрүңүз!

Бул эпизоддо

Бул бөлүмдө сиздер менен бир топ маанилүү нерселер тууралуу талкууладык. Эгер сиз өзүңүздү бир аз ыңгайсыз сезсеңиз, анда мен сизди күнөөлөбөйм.. <жымжырт>. Бул бөлүмдү түшүнүү кыйынга турса да, жакшы жаңылык - циклдерди колдонууну өздөштүрүп бүткөндөн кийин, сизде татыктуу, чыныгы программаларды түзүү үчүн керектүү шаймандардын бардыгы бар экендиги кубандырат.

Албетте, сиз бир кеңсеге кирип, дүйнөдөгү эң мыкты программист катары жогорку акы төлөнүүчү конвертти ала албай калышыңыз мүмкүн. Бирок, эй, сиз дагы деле өспүрүмсүз! Сиз ансыз деле келечектеги жеткилеңсиз, атаандаштыктын сызыгынан алда канча алдыда баратасыз. Андан тышкары, достор, алдыда дагы тогуз бөлүм бар экенин унутпайлы!

Китептин ушул жеринде сиз өзүңүздү-өзүңүз эрмектеп, өзүңүздүн код үзүндүлөрүн жана мини программаларыңызды түзүп алсаңыз болот. Жашоодогу бардык нерселер сыяктуу эле, практика чеберчиликти жаратат. Андыктан жаңы супер күчтү тез-тез сынап туруңуз. Канчалык көп код жазсаңыз, программалоону ошончолук жакшы түшүнөсүз.

Айта кете турган болсок, кийинки бөлүмдө, биз биринчи толук кандуу программабызды түзүп жатканда ушул кезге чейин үйрөнгөндөрдүн бардыгын пайдалуу колдонууга жумшайбыз! Ал Супер Баатыр Генератору 3000 деп аталып, ага циклдер, өзгөрмөлөр, if-else, сап жана математикалык функциялар жана башка көптөгөн нерселер кирет. Досторуңузду чакырыңыз, ошондо сиз компаниянын сыймыктуу кишисине айланасыз!

Ал эми бул бөлүмдү жыйынтыктап, бул эпизод боюнча ыкчам шилтеме баракчасына эмнелерди үйрөнгөнүбүздү кайталайлы!

- Циклдер - итерация, эгерде белгиленген шарттар аткарылса же аткарылбаса, кодуңуздун бөлүктөрүн кайталоо дегенди билдирет.
- For циклы шарт аткарылганга чейин же range функциясын колдонуп кайталоо үчүн колдонсо болот. Мисалы үчүн:

```
for x in range(1,10):
      print("Керемет Бала бул жерде!")
```

- Range() - цикл аркылуу бир нече жолу кайталоого мүмкүндүк берген функция. Ал үч параметрге ээ жана төмөнкүдөй жол менен аныкталат:

```
range(1, 10, 1)
```

Биринчи сан баштапкы чекит болуп саналат. Экинчи сан - акыркы чекит. Үчүнчү сан - милдеттүү эмес – эки чекит ортосундагы сандардын айырмасы катары белгилүү жана range() эсептелген өсүштү башкарат. Мисалы, айырма 2 болсо, циклдин ар бир кайталанышында сандар 2ге көбөйүп отурат.

- Ал эми циклдер шарт же критерийлер аткарылганга чейин же логикалык жактан туура деп бааланганга чейин кайталанат. Мисалы:

```
    айлыкАкы = 0
while айлыкАкы < 10:
    print("Мен карызмын, карызмын, демек жумуштан кетип калам.")
    айлыкАкы = айлыкАкы +1
```

- Чексиз циклдер тентек келишет. Ошондуктан алардан оолак болуу керек. Алар цикл программалоо логикасында кемчиликтер болгондо же адатта ката кетиргенде пайда болот. Бул жакпаган алгебра сабагы сыяктуу эч качан бүтпөгөн жана түбөлүктүү цикл .

  Балдар, бул жердеги бир мисалды ***колдонбогула!***

```
x = 0
while x == 0:
        print:("Сен айлампа менен түш?")
        print("Ооба, сен мени билесиң!")
```

  x өзгөрмөсү 0 болгондуктан, критерийлер цикл 0 болгондо иштеши керектигин айткандыктан, бул цикл чексиз уланат.

- Str.lower() - сапты кичине тамгага которуучу функция. Мисалы:

```
аты = "Керемет Бала"
print(аты.lower())
```

  Бардыгы “керемет бала” деп кичинекей тамгалар менен басып чыгарат.

- Str.upper() str.lower() менен окшош иштейт, болгону саптагы бардык тамгаларды чоң тамгага өзгөртөт. Мисалы:

```
аты = "Керемет Бала"
print(аты.upper())
```

- Break белгилүү бир шарт аткарылса жана циклди эртерээк бүтүшүн кааласаңыз, циклди өзүнүн итерациясынан чыгууга же бузууга мажбурлайт.
- Continue циклдеги итерацияны циклден толугу менен чыкпастан өткөрүп жиберүүгө мүмкүндүк берет.
- Pass - кийинчерээк алардын ичиндеги айрым шарттарды аныктабастан, циклдерди түзүүгө мүмкүндүк берүүчү толтургуч. Ушундай жол менен сиз цикл структурасын түзүп, кийинчерээк өзүңүздүн кодуңузду текшергенде эч кандай ката кетирбестен, кандай критерийлерди колдоно ала тургандыгыңызды чече аласыз.

6 - БӨЛҮМ

ҮЙРӨНГӨНҮБҮЗДҮ КОЛДОНУУ БИЛҮҮ

Сиз узак жолду басып өттүңүз. Бардыгы сиз микротолкундуу мештен бир кесим пиццаны уурдаймын деп чоң ката кетирип, сизди радиоактивдүү программист чагып алган окуядан башталды. Ошол жерде сиздин күчүңүз ашынып, өзүңүздүн татыктуу жолдош экениңизди далилдедиңиз. Бирок эми сиздин билимиңизди чындап текшерүү мезгили келди. Сиз ага даярсызбы?

Бул бөлүмдө ушул кезге чейин үйрөнгөндөрдүн бардыгын кайталап, өзүңүздүн толук метраждуу программаңызды жаратуу үчүн колдоносуз. Мындан тышкары, сиз дагы бир нече жаңы амалдарды үйрөнүп, ушул бөлүмдүн аягында ишенимдүү талапкерден толук баатыр статусуна айланганга жетишесиз.

Сиз дагы эле Керемет Бала бойдон каласыз. Эми мындан кийин Керемет Атанын бут кийимин тазалабайсыз!

Алгачкы чыныгы программаны түзүү

Биринчи жолу толук иштей турган тиркемесибизди куруп баштаардан мурун биз аны Супер Баатыр Генератору 3000 деп атайбыз да, адегенде биздин программабыз эмнени аткарыш керек экендигин тактайбыз. Негизги түшүнүк жөпжөнөкөй: бизге туш келди же капысынан супер каармандарды жарата турган тиркеме керек. Бул жерде коркунучтуу эч нерсе жок, туурабы?

Бул башталышы, бирок, албетте, биз андан дагы көп нерселерге муктажбыз. Мисалы, баатыр деген эмне? Алардын кандайдыр бир касиеттери барбы? Укмуштуудай супер күчү барбы? Аттары ким? Мунун баарына жообубуз бар.

Жакшы, баатыр программисттер катары биз ар дайым өзүбүз түзгөн программаларды пландаштырууну каалайбыз. Программанын максатын, анын кандайча иштей тургандыгын, аны коддоодо жана курууда бизди туура жолго салууга жардам бере турган бардык маалыматтарды билишибиз керек.

Мисалы, биз бул программада төмөнкүлөрдүн керек болорун билебиз:

• Супер Баатырдын аты (кокустан, туш келди түзүлгөн);

• Супер Баатырдын фамилиясы (кокустан, туш келди түзүлгөн);

• Супер Баатырдын атын / фамилиясын бир сапка кошуу үчүн код;

• Берилген маалыматтын чегинде статистиканын жыйындысын кокустан түзүү үчүн код;

• Кокустан пайда болгон күч генератору.

Мындан тышкары, биздин бардык маалыматтарыбызды сактоо үчүн өзгөрмөлөр, биздин биринчи, акыркы жана бириккен ысымдарыбыз жана биздин супер күчтөрүбүз керек болот. Ошондой эле бизге маалымат структурасы керек. Бул учурда **тизмелер** - биздин ысымдарыбыздын жана супер күчтөрүбүздүн баалуулуктарын дайындоо үчүн керек. Алардын ичинен биз баатырга берилүүчү ысымдарды / ыйгарым укуктарды туш келди тандап алабыз.

Татаал сыяктуу болуп жатабы? Кабатыр болбоңуз, андай эмес. Программанын ар бир бөлүмүн этап-этабы менен карап чыгып, буга чейин камтыган нерселердин бардыгын жаңыртабыз. Ошентсе да, коргоочу кийимибизди жана маскаларыбызды кийип, Супер Баатыр Генератордун биринчи бөлүгүн баштайлы!

Модулдарды импорттоо

Алгач, биздин программа эки *модулга* - убакытты үнөмдөөгө жана адамдардын каталарын азайтууга колдоно турган жалпы тапшырманы аткаруу үчүн мурда иштелип чыккан коддун бөлүктөрүнө таянат. Биринчиси - биз буга чейин иштеп келген **random** - кокустук. Эсиңизге сала кетсек, **random** туш келди сандарды жаратуу үчүн колдонулат. Ошондой эле, анын жардамы менен сиз кокустан, божомолдогон маанини (же маанилерди), башка нерселер менен катар эле тизмеден тандап алсаңыз болот. Биз аны программабызда эки максатта тең колдонобуз.

Биз импорттогон экинчи модуль **time -** убакыт деп аталат жана биз буга чейин аны камтыган эмеспиз. Анын негизги функцияларынын бири - бул программабыздын аткарылышында "тыныгуу" түзүүгө мүмкүнчүлүк берүүсү. Айрым коддорду аткарууда кечиктирүүнү каалаган көптөгөн себептер болушу мүмкүн. Максаттарга жетүүбүз үчүн биз **time** туура колдонуп, биздин программа татаал нерсени эсептеп жаткандай таасир калтырабыз.

Келгиле, **SuperHeroGenerator3000.py** аттуу жаңы файл түзүп, төмөнкү кодду ага кошолу:

```
# Кийинчерээк сан жана саптарды туш келди тандап алуу үчүн  random #модулду импорттоо
import random

# Тыныгуу түзүү үчүн time модулун импорттоо
import time
```

Өзгөрмөлөрдү түзүү

Мурда белгиленгендей, бул программа бир нече өзгөрмөгө жана сакталган маалыматтардын тизмелерине таянат. Биздин статистикалык маалыматтарды сактоо үчүн өзгөрмө колдонобуз. Азыркы учурда сиз алардын колдонулушун жана аларды кантип аныктоону билишиңиз керек. Айткандай эле, келгиле, ушул кодду

өзүбүздүн SuperHeroGenerator3000.py файлыбызга, импорттолгон **time** жана **random** модулдарыбыздын астына кошолу.

```
Зээндүүлүк = 0
Тапкычтык = 0
Чыдамдуулук = 0
Акылмандык = 0
Күч = 0
Дисциплина = 0
Эптүүлүк = 0
Ылдамдык = 0

жооп = ' '
```

Биринчи өзгөрмө топтому сиздин мүнөзүңүз канчалык акылдуу, канчалык күчтүү жана башка ушул сыяктуу статистиканы кармоо үчүн колдонулат. Баштапкы маанилердин бардыгы 0 деп коюлгандыгына көңүл буруңуз. Кийинчерээк колдонмодо биз ушул маанилерди өзгөртүү үчүн **random** колдонобуз. Бирок азырынча биз аларга бир маани беришибиз керек, демек, 0!

Бизде *жооп* деп аталган статистикалык топтун сыртында турган өзгөрмө бар экендигин байкасаңыз керек. Программа иштетилгенден кийин ал колдонуучуга улантуу үчүн суроо берет. Биз колдонуучунун жообун кармап туруу үчүн жооптор сабынын өзгөрмөсүн колдонобуз. Азырынча биз ага эч кандай маани бербейбиз; аны кийинчерээк колдонуучу толтурат.

Тизмелерди аныктоо

Тизмелер бир нече маалыматты сактоо үчүн колдонулат. SuperHeroGenerator3000.py тиркемеси үч тизмеге таянат: бири мүмкүн болгон супер күчтөрдүн тизмесин камтыйт. Ал *суперКүчтөр* деп аталат. Ал эми калган экөө мүмкүн болгон ысымдардын жана фамилиялардын тизмесин камтыйт.

Кийинчерээк программада биз каармаңыбызга ысым жана супер күч берүү үчүн ушул тизмелердин арасынан **random** модулун колдонуп тандап алабыз. Азырынча биз тизмелерди түзүп, аларга кандайдыр бир маанилерди беришибиз керек.

Азырынча мен берген маанилерди колдонуңуз. Бирок, программаны бир нече жолу сынап көргөндөн кийин, бул тизмелерге өзүңүздүн супер каармаңыздын ысымынын айкалыштарын жана күчүн кошуңуз - чыгармачылык менен алектенип, көңүл ачыңыз!

Бул жерде тизмелердин коду бар. Аны өзгөрмөлөр тизмесинин астына файлыңызга кошуңуз:

```
# Мүмкүн болгон супер күчтөрдүн тизмесин түзүңүз

суперКүчтөр = ['Учуучу', 'Укмуш Күчтүү', 'Телепатия', 'Супер Ылдам',
'Көп Хот-Догдорду Жей Алат', 'Аркандан Секиргени Жакшы']

# Мүмкүн болгон ысымдардын жана фамилиялардын тизмесин түзүү

суперЫсымдар = ['Керемет','Уатта','Кутурган','Укмуш', 'Таң Калычтуу',

'Татыктуу', 'Даңазалуу', 'Ортодон-Жогору', 'Ал Жигит', 'Өзгөчө']
суперФамилиялар = ['Бала', 'Киши', 'Динго', 'Бифкейк', 'Кыз', 'Аял',
'Жигит','Баатыр', 'Макс', 'Кыял', 'Мачо','Аргымак']
```

Азыр биздин маалымат структураларыбыз болгондуктан, кийинки бөлүккө өтүүгө кез келди.

Кириш текст жана колдонуучудан алынган маалыматты кабыл алуу

Биздин коддун кийинки бөлүгү колдонуучу менен саламдашуу жана алардан кандайдыр бир билдирүүнү кабыл алуу болуп саналат. 5-бөлүмдөн үйрөнгөнүбүздөй, колдонуучудан **input**() функциясын колдонуп, сунушту кабыл алабыз. Жаңы түзүлгөн тизмелердин астына файлыңызга төмөнкү кодду кошуңуз:

```
# Киришүү текст

print("Супер Баатыр генератору 3000 менен супер баатыр жаратууга
даярсызбы?")

# Колдонуучуга суроо берип, аларга жооп сунуштаңыз

# input()-клавиатурадан эмнени жазгандыгын 'угат'

# Колдонуучунун жоопторун чоң тамгаларга айландыруу үчүн upper()
#колдонобуз

print("Клавиатурадан О киргиз:")

жооп = input()

жооп = (жооп.upper())
```

Жумушту жеңилдетүү үчүн колдонуучунун билдирүүсүн кабыл алгандан кийин, алардын жоопторун чоң тамгаларга айландырабыз. Муну эмне үчүн жасап жатабыз? Жооп берип жатканда кичине жана чоң тамгалардын айкалышын текшерип отурбашыбыз үчүн колдонобуз.

Эгерде биз текстти чоң тамгага айландырбасак, анда ооба, Ооба жана башкаларды текшерип отурмакпыз. Сапты конверттөө жана жөнөкөй "ООБА" деп (бул мисалыбызда "О" тамгасы) текшерүү кыйла жеңил жана натыйжалуу.

Колдонуучунун жообу ооба болгончо кайталанган **while** циклин колдонуп текшеребиз. Шарт аткарылып, цикл бүткөндөн кийин, программа уланып, көңүл ачуу башталат!

Кириш тексттин жана **input**() бөлүмүнүн астына кодуңузга бир аз **while** циклын кошуңуз:

```
# "О"жообун текшерүү үчүн while цикли
# Бул while цикли жооптун мааниси "О" БОЛБОГОН учурда улантылат
# Колдонуучу "О" деп жазганда гана цикл чыгып, программа уланат
while жооп != "О":
      print("Кечиресиз, бирок улантуу үчүн "О" теришиңиз керек!")
      print("Клавиатурадан О киргиз:")
      жооп = input()
      жооп = жооп.upper()
print("Сонун, баштайлы!")
```

Күтүү!

Чыныгы жашоодогу жазуудай эле, кээде компьютердик программалообузда колдонуучуга, чындыгында, сонун нерсе болуп жатат деп ойлонуп калышы үчүн чыңалууну же тыныгууну сөзсүз кошкубуз келет. Же болбосо, колдонуучуну тез жылдырбай, экранда текстти окууга убакыт берүү үчүн, программаны мезгил-мезгили менен тындырып алабыз.

Кандай болгон күндө дагы, биз сиз үйрөнө элек жаңы модулду колдонуп, бул укмуштуудай натыйжага жетише алабыз, ал - **time**().

Кодубузда **time()** модулун колдонуп жатканда, азырынча мунун бардык функцияларын толугу менен камтыбайбыз. Азырынча биз бул ыңгайлуу жаңы куралдын бир гана аспектисин, тактап айтканда, анын **sleep()** функциясын колдонууну каалайбыз.

Башка функциялар сыяктуу эле, **time** параметрлерди кабыл алат. Алар алты жалпы жана анча көп колдонулбаган башка бир нече параметрлерден турат. **Sleep()** сиздин программаңызда бир нече секунда менен өлчөнгөн тыныгууну жаратат жана бизде төмөнкүчө көрүнөт:

```
time.sleep(3)
```

Кашаанын ичиндеги сан - бул тыныгууга берилген секунддардын саны. Биз муну терип, аны менен эле бүтүрсөк болот. Бирок айтылгандай - биз кандайдыр бир укмуштай шык-жөндөмдү каалайбыз! Ошентип, time.sleep() деп пайдаланууну ордуна колдонуучунун экранына бир нече күтүү убактысын окшоштуруу үчүн бир нече чекит(…) басып чыгарууну каалайбыз. Бул жөн эле укмуш көрүнөт!

Бул үчүн time() функциясын үч жолу кайталанган for циклине жайгаштырабыз. Ар бир жолу цикл аркылуу биздин программа колдонуучунун экранына басып чыгарылат.

Бул кодду SuperHeroGenerator3000.py файлыңызга кошуңуз:

```
print("Атыбызды туш келди тандап жатат...")
# Time () функциясын колдонуп, тыныгуу жаратуу
for i in range(3):
      print(".....")
      time.sleep(3)
print("(Сиз күткөндү жаман көрөсүз деп ойлойм!)")
print(" ")
```

Теориялык жактан, эгерде биз ушул учурда программабызды иштете турган болсок, анда экрандан төмөнкү натыйжаны көрө алабыз:

```
Супер Баатыр генератору 3000 менен супер баатыр жаратууга даярсызбы?
Клавиатурадан О/Ж киргиз:
о
Атыбызды туш келди тандап жатат...
.....
.....
.....
(Сиз күткөндү жаман көрөсүз деп ойлойм!)
```

time() функциясы башталганда, "....." саптарынын ар бири туптуура 3 секундага созулуп, биздин "драмалык паузаны" түзөт.

Азыр биздин кириш текстибиз бар, программаларыбызды бир азга кантип токтото турууну түшүндүк, анан дагы биздин баштапкы тизме жана өзгөрмөлөрүбүз, ошондой эле импорттолгон модулдарыбыз бар. Ошондуктан тиркеменин өзөгүнө жетүүгө кез келди!

Бул кийинки бөлүмдө биз жараткан супер баатырлардын ар кандай бөлүктөрүн кокустуктан тандап кодубуздун бир бөлүгүн түзөбүз. Бул үчүн биз эски алтын **random()** модулуна таянабыз.

Супер баатырлардын аттарын кокустан божомолдоп тандоо

Ар бир супер баатырга беш нерсе керек:

- Укмуш кийим;
- Супер күч;
- Күндүзү иштебеш үчүн зыянсыз каражат булагы;
- Жалгыз отуруп түшкү тамакты жегенде колун сүрткөнга бет ааарчы (супер баатырлардын жоолугушууга убактысы жок!);
- Анан дагы, албетте, керемет ысым.

SuperHeroGenerator3000 кодундагы кийинки кадам - ысымдарды жаратуу бөлүмүн программалоо. Буга чейин эсиңиздерде болсо керек, биз супер баатырдын ысымдары жана фамилияларына толгон эки тизме түзгөнбүз. Эскерте кетсек, бул тизмелер:

```
суперКүчтөр = ['Учуучу', 'Укмуш Күчтүү', 'Телепатия', 'Супер Ылдам',
'Көп Хот-Догдорду Жей Алат', 'Аркандан Секиргени Жакшы']

суперФамилиялар = ['Бала', 'Киши', 'Динго', 'Бифкейк', 'Кыз', 'Аял',
'Жигит''Баатыр', 'Макс', 'Кыял', 'Мачо','Аргымак']
```

Ысым жаратуу кодунун идеясынын негизи - бул эки тизменин ар биринен бирден ысым чыгарып, аларды бириктирип, каармаңыздын атын жаратуу. Муну ишке ашыруунун көптөгөн жолдору бар. Бирок биздин максат үчүн биз эки маанини туш келди тандап алгыбыз келет. Ошондо программа башталган сайын, ал ысымдардын уникалдуу айкалышын жаратат.

Тереңирээк кирүүдөн мурун ушул натыйжага жетүү үчүн колдонуп жаткан кодду карап көрөлү. **time()** кодунан кийин төмөнкү кодду SuperHeroGenerator3000.py файлыңызга кошуңуз:

```
# Супер баатырдын атын туш келди тандоо

# Муну эки тизменин ар биринен бирден ат тандоо менен жасайбыз

# Жана аны суперАталыш өзгөрмөсүнө кошуу

суперАталыш = random.choice(суперЫсымдар)+ " "
+random.choice(суперФамилиялар)

print("Сиздин Супер Баатырдын аты: ")

print(суперАталыш)
```

Коддун бул бөлүмүн түшүнүү абдан жөнөкөй. Биз *суперАталыш* деген өзгөрмө түзүп баштайбыз. Анын милдети каармаңыздын ысымдары менен фамилияларын бириктирүү (биз *суперЫсымдар* жана *суперФамилиялар* тизмелеринен алабыз).

Андан кийин биз *суперЫсымдар* тизмесинен бир маанини жана *суперФамилиялар* тизмесинен бир маанини туш келди тандоо үчүн random() - жана атайын **random.choice** колдонобуз.

Коддун сабынын бөлүгү мындайча окулуп:

```
+ " " +
```

түшүнүксүз сезилиши мүмкүн. Бирок, ал жөнөкөй. Бул учурда + белгиси биздин эки сапты *бириктирүү* же кошуу үчүн колдонулат. Биз ысым менен фамилиянын ортосундагы боштукту каалагандыктан, алардын ортосуна " " кошуу менен боштукту кошушубуз керек же бириктиришибиз керек. Болбосо, биз жөн гана **random.choice (***суперЫсымдар***) + random.choice (***суперФамилиялар***)** деп жаза алмакпыз.

Акыры, программабыздын ушул бөлүгүн жаңы түзүлгөн *суперАталыш* маанисин, **print** (*суперАталыш*) аркылуу басып чыгаруу менен аяктайбыз.

Эми биз программабызды иштете турган болсок, анда мындай нерсе келип чыгат:

```
Супер Баатыр генератору 3000 менен супер баатыр жаратууга даярсызбы?
Клавиатурадан О/Ж киргиз:
о
Атыбызды туш келди тандап жатат...
............
............
............
(Сиз күткөндү жаман көрөсүз деп ойлойм!)

Сиздин Супер Баатырдын аты:
Өзгөчө Макс
```

Же болбосо

```
Супер Баатыр генератору 3000 менен супер баатыр жаратууга даярсызбы?
Клавиатурадан О/Ж киргиз:
о
Атыбызды туш келди тандап жатат...
............
............
```

```
...........
(Сиз күткөндү жаман көрөсүз деп ойлойм!)


Сиздин Супер Баатырдын аты:
Уатта Макс
```

Эскертүү. Маанилери кокустан түзүлгөндүктөн, супер баатырыңыздын аты мен көрсөткөн мисалдан башкача болушу мүмкүн.

Текшерүү

Мындан ары барардан мурун оюндун ушул этабында кодуңуз меники менен дал келгенин текшерип көрөлү. Эгерде сиз көрсөтмөлөрдү аткарып, кодуңузду туура тартипке салсаңыз, анда программаңыз ушундай болушу керек. Эгер андай болбосо, кабатыр болбоңуз - жөн гана кодуңузду менин кодума окшоштуруп өзгөртүп, бөлүмдөрдү кайталап окуп, кайсыл жери туура эмес болгонун билип алыңыз!

Сиздин кодуңуз эми ушундай болушу керек:

```
# Кийинчерээк сан жана саптарды туш келди тандап алуу үчүн  random
#модулду импорттоо
import random

# Тыныгуу түзүү үчүн time модулун импорттоо

import time

#Мүнөздөрдүн статистикасын көрсөткөн өзгөрмөлөр

Зээндүүлүк = 0
Тапкычтык = 0
Чыдамдуулук = 0
Акылмандык = 0
Күч = 0
Дисциплина = 0
Эптүүлүк = 0
Ылдамдык = 0

жооп = ' '
```

```
# Мүмкүн болгон супер күчтөрдүн тизмесин түзүңүз
суперКүчтөр = ['Учуучу', 'Укмуш Күчтүү', 'Телепатия', 'Супер Ылдам',
'Көп Хот-Догдорду Жей Алат', 'Аркандан Секиргени Жакшы']

# Мүмкүн болгон ысымдардын жана фамилиялардын тизмесин түзүү

суперЫсымдар = ['Керемет','Уатта','Кутурган','Укмуш', 'Таң Калычтуу',
'Татыктуу', 'Даңазалуу', 'Ортодон-Жогору', 'Ал Жигит', ' Өзгөчө']
суперФамилиялар = ['Бала', 'Киши', 'Динго', 'Бифкейк', 'Кыз', 'Аял',
'Жигит','Баатыр', 'Макс', 'Кыял', 'Мачо','Аргымак']

# Киришүү тексти
print("Супер Баатыр генератору 3000 менен супер баатыр жаратууга
даярсызбы?")
# Колдонуучуга суроо берип, аларга жооп сунуштаңыз
# input()-клавиатурадан эмнени жазгандыгынызды 'угат'
# Колдонуучунун жоопторун чоң тамгаларга айландыруу үчүн upper()
колдонобуз
print("Клавиатурадан О же Ж киргиз:")
жооп = input()
жооп = (жооп.upper())
# “О” жоопту текшерүү үчүн While цикли
# Бул while цикли жооптун маниси "О" БОЛБОГОН учурда улантылат
# Колдонуучу "О" деп жазганда гана цикл чыгып, программа уланат
while жооп != "О":
      print("Кечиресиз, бирок улантуу үчүн “О” теришиңиз керек!")
      print("Клавиатурадан О киргиз:")
      жооп = input()
      жооп = жооп.upper()
print("Сонун, баштайлы!")
print("Атыбызды туш келди тандап жатат...")
# Time () функциясын колдонуп, тыныгуу жаратуу
for i in range(3):
      print(".....")
      time.sleep(3)
print("(Сиз күткөндү жаман көрөсүз деп ойлойм!)")
print("")
```

```
# Супер баатырдын атын туш келди тандоо
# Муну эки тизменин ар биринен бирден ат тандоо менен жасайбыз
# Жана аны суперАтальш өзгөрмөсүнө кошуу

суперАтальш = random.choice(суперЫсымдар)+ " "
+random.choice(суперФамилиялар)

print("Сиздин Супер Баатырдын аты: ")
print(суперАтальш)
```

Супер күчтөрдү кокустан тандоо

Эми кызыктуу бөлүгү келди, ал биздин каармaныбыздын супер күчүн кокустан пайда кылуу! Кантсе да, ал мурдунан лазер нурларын атып же бир күндүн ичинде толук сакал өстүрө албаса, ушунчалык супер болмок эмес эле?

Биздин *суперЫсымдар* жана *суперФамилиялар* тизмелеribиздегидей эле, биз супер күчтөргө ээ болуу үчүн тизме түзгөнүбүздү эсиңизден чыгарбаңыз, ал - *суперКүчтөр* тизмеси. Дал ушул тизмеден биз супер баатырдын арсеналында кандай күчтөр бар экендигин тандайбыз.

Эскертүү. Программаны толугу менен аткарып бүткөндөн кийин жана аны бир нече жолу сыноодон өткөргөндөн кийин супер кучтөрдүн тизмесине өзүңүздүн супер күчтөрдү кошуп алсаңыз болот. Мүмкүн болушунча чыгармачыл болуңуз!

Төмөнкү кодду superHeroGenerator3000.py файлыңызга кошуп, аны баатырдын ысымын кокустан жараткан коддун бөлүгүнүн астына жайгаштырыңыз:

```
print("")
print("Эми сизде кандай супер күч бар экендигин көрүүгө кез келди!)")
print("(супер баатырдын күчүн чыгаруу...)")

# Дагы бир жолу эффект жаратууда

for i in range(2):
     print(".....")
     time.sleep(3)

print("(аах... БУЛ сизге жакпайт...)")
for i in range(2):
     print(".....")
     time.sleep(3)
```

```
print("(дээрлик даяр....)")

# Муну суперКүчтөр тизмесинен туш келди тандоо менен жасайбыз
# Жана аны күч өзгөрмөсүнө кошобуз

күч = random.choice(суперКүчтөр)

# Күч өзгөрмөсүн жана кээ бир тексттерди басып чыгаруу
print("Сиздин жаңы күчүңүз: ")
print(күч)
print("")
```

Көрүнүп тургандай, бул код колдонуучунун экранына бир нече текст басып, баатырдын супер күчү пайда болору жөнүндө кабарлоо менен башталат. Ушундан кийин, биз дагы бир укмуштуудай эффект жаратуу жана программаны акырындатуу үчүн **time.sleep()** колдонобуз. Бул жолу, биз **for loop** аркылуу эки жолу гана “.....” саптарын басып чыгарабыз - алардын ар бири 3 секундага созулат.

Коддун кийинки бөлүгү:

```
күч = random.choice(суперКүчтөр)
```

Күч деп аталган жаңы өзгөрмө түзүп, андан кийин *суперКүчтөр* тизмесинен туш келди маанини алат. Акыр-аягы коддун бөлүмү кубаттуулуктун маанисин басып чыгарып, колдонуучу кандай супер кубаттуулукту тандап алгандыгын көрө алат.

Эгерде ушул этапта программаны иштете турган болсок, анда теориялык жактан төмөндөгүдөй натыйжаларга ээ болмокпуз:

```
Супер Баатыр генератору 3000 менен супер баатыр жаратууга даярсызбы?
Клавиатурадан О же Ж киргиз:
О
Сонун, баштайлы!
Атыбызды туш келди тандап жатат...
..........
..........
..........
(Сиз күткөндү жаман көрөсүз деп ойлойм!)

Сиздин Супер Баатырдын аты:
Керемет Жигит

Эми сизде кандай супер күч бар экендигин көрүүгө кез келди!)
(супер баатырдын күчүн чыгаруу...)
```

```
...........
...........
(аах... БУЛ сизге жакпайт...)
...........
(дээрлик даяр....)
...........
(дээрлик даяр....)
Сиздин жаңы күчүңүз:
Учуучу
```

Же

```
Супер Баатыр генератору 3000 менен супер баатыр жаратууга даярсызбы?
Клавиатурадан О же Ж киргиз:
О
Сонун, баштайлы
Атыбызды туш келди тандап жатат...
...........
...........
...........
(Сиз күткөндү жаман көрөсүз деп ойлойм!)

Сиздин Супер Баатырдын аты:
Керемет Жигит

Эми сизде кандай супер күч бар экендигин көрүүгө кез келди!)
(супер баатырдын күчүн чыгаруу...)
...........
...........
(аах... БУЛ сизге жакпайт...)
...........
(дээрлик даяр....)
...........
(дээрлик даяр....)
Сиздин жаңы күчүңүз:
Көп Хот-Догдорду Жей Алат
```

Эсиңизде болсун, сиздин натыйжаларыңыз айырмаланышы мүмкүн. Анткени супер күч жана супер баатыр ысымдары туш келди пайда болду.

Биздин программабыздын аягы

Алгач толук кандуу программабызды түзүп бүтөбүз! Теңдешсиз Стен Линин сөзү менен айтканда - Эксельсиор!

Биздин колдонмонун акыркы бөлүгү каарманыбыздын статистикасын кокустан пайда кылат. Эсиңизде болсо, биздин коддун башында биз бир канча өзгөрмө түздүк (*Зээндүүлүк*, *Тапкычтык*, *Чыдамдуулук*, *Акылмандык*, *Акылмандык*, *Дисциплина, Эптүүлүк* жана *Ылдамдык*) жана алардын ар бирине 0 маанисин бергенбиз.

Төмөнкү коддо биз баатырдын статистикасын чагылдырган ушул өзгөрмөлөрдүн ар бирин туш келди бүтүн сан менен 1ден 20га чейин беребиз. Муну биз 2-бөлүмдө талкуулаган **random.randint()** функциясын колдонуп жасайбыз.

SuperHeroGenerator3000.py файлыңызга төмөнкүлөрдү кошуңуз:

```
print("Акыркы, бирок, маанилүү, Сиздин статистикаңызды чыгаралы!")
print("Сиз супер акылдуу болосузбу? Супер күчтүү? Супер сулуу?")
print("Муну билүүгө убакыт келди!")

#  эффект жаратуу жана программаны кайрадан жайлатуу

for i in range(3):
       print(".....")
       time.sleep(3)

# Төмөндө берилген өзгөрмөлөрдүн ар бирине жаңы туш келди маанилерди
#берүү
# Жаңы маанилер 1-20 арасында болот

Зээндүүлүк = random.randint(1,20)
Тапкычтык = random.randint(1,20)
Чыдамдуулук = random.randint(1,20)
Акылмандык = random.randint(1,20)
Дисциплина = random.randint(1,20)
Эптүүлүк = random.randint(1,20)
Ылдамдык = random.randint(1,20)

# Статистиканы басып чыгаруу

print("Сиздин жаңы статисткаңыз: ")
print("")
print("Зээндүүлүк: ", Зээндүүлүк)
print("Тапкычтык: ", Тапкычтык)
```

```
print("Чыдамдуулук: ", Чыдамдуулук)
print("Акылмандык: ", Акылмандык)
print("Дисциплина: ", Дисциплина)
print("Эптүүлүк: ", Эптүүлүк)
print("Ылдамдык: ", Ылдамдык)
print("")

# Супер баатырдын толук баяндамасын басып чыгаруу
# Буга баатырдын аты, супер күчү жана статистикасы кирет

print("Жаңы Супер Баатырыңыздын толук баяндамасы!")
print("Супер Баатыр генератору 3000 колдонуп жатканыңыз үчүн рахмат!")
print("Бардык досторуңузга айтып бериңиз!")
print("")
print("Персонаждын кыскача баяндамасы:")
print("")
print("")
print("Супер Баатырдын аты: ", суперАталыш)
print("Супер күчү: ", Күч)
print("")
print("Зээндүүлүк: ", Зээндүүлүк)
print("Тапкычтык: ", Тапкычтык)
print("Чыдамдуулук: ", Чыдамдуулук)
print("Акылмандык: ", Акылмандык)
print("Дисциплина: ", Дисциплина)
print("Эптүүлүк: ", Эптүүлүк)
print("Ылдамдык: ", Ылдамдык)
```

Эгерде жаңы программанын ушул бөлүгүн карап көрсөк,

```
Зээндүүлүк = random.randint(1,20)
Тапкычтык = random.randint(1,20)
Чыдамдуулук = random.randint(1,20)
Акылмандык = random.randint(1,20)
Дисциплина = random.randint(1,20)
Эптүүлүк = random.randint(1,20)
Ылдамдык = random.randint(1,20)
```

бүтүн санды өзгөрмөбүзгө кокустан ыйгаруу канчалык оңой экендигин көрө алабыз. Кашаанын ичиндеги сандар мүмкүн болгон диапазондун эң төмөнкү маанисин жана эң жогорку маанисин билдирет. Ал сан ар дайым 1ден 20га чейин болот.

Супер Баатыр генератору 3000 бүттү!

Биринчи аяктаган программабыздын даңкына бөлөнө турган кез келди! Өзүңүздү далыга таптап, чуркаңыз жана бардык досторуңузга менин кандай мыкты мугалим экенимди жана бул китеп эң жакшы дос экендигин айтыңыз!

Биз жасашыбыз керек болгон эң акыркы нерсе - сиздин кодуңуз ушул китепте камтылган нерселер менен дал келиши керек. Ал дал келгенден кийин сиз программаны кайра-кайра иштетип, тизмелердин баалуулуктарын өзгөртүп, бардык досторуңузду жана мугалимдериңизди өздөрүнүн супер баатырларын жаратууга чакырасыз!

Бул жерде superHeroGenerator3000.py коду толугу менен бар - кодуңузду салыштырып, дал келгенин текшериңиз:

```
# Кийинчерээк сан жана саптарды туш келди тандап алуу үчүн  random модулун импорттоо

import random

# Тыныгуу түзүү үчүн time модулун импорттоо

import time

Зээндүүлүк = 0

Тапкычтык = 0

Чыдамдуулук = 0

Акылмандык = 0

Күч = 0

Дисциплина = 0

Эптүүлүк = 0

Ылдамдык = 0

Жооп = ' '

# Мүмкүн болгон супер күчтөрдүн тизмесин түзүңүз

суперКүчтөр = ['Учуучу', 'Укмуш Күчтүү', 'Телепатия', 'Супер Ылдам', 'Көп Хот-Догдорду Жей Алат', 'Аркандан Секиргени Жакшы']

# Мүмкүн болгон ысымдардын жана фамилиялардын тизмесин түзүү

суперЫсымдар = ['Керемет','Уатта','Кутурган','Укмуш', 'Таң Калычтуу',

'Татыктуу', 'Даңазалуу', 'Ортодон-Жогору', 'Ал Жигит', ' Өзгөчө']
```

```
суперФамилиялар = ['Бала', 'Киши', 'Динго', 'Бифкейк', 'Кыз', 'Аял',
'Жигит','Баатыр', 'Макс', 'Кыял', 'Мачо','Аргымак']

# Киришүү тексти

print("Супер Баатыр генератору 3000 менен супер баатыр жаратууга
даярсызбы?")

# Колдонуучуга суроо берип, аларга жооп сунуштаңыз

# input() клавиатурадан эмнени жазгандыгын 'угат'

# Колдонуучунун жоопторун чоң тамгаларга айландыруу үчүн upper()
#колдонобуз

print("Клавиатурадан О/Ж киргиз:")

жооп = input()

жооп = (жооп.upper())

# "О" жоопту текшерүү үчүн While цикли

# Бул while цикли жооптун маниси "О" БОЛБОГОНДО улантылат

# Колдонуучу "О" деп жазганда гана циклден чыгып, программа уланат

while жооп!= "О":
      print("Кечиресиз, бирок улантуу үчүн "О" теришиңиз керек!")
      print("Клавиатурадан О киргиз:")
      жооп = input()
      жооп = жооп.upper()
      print("Сонун, баштайлы!")

print("Атыбызды туш келди тандап жатат...")

# Time () функциясын колдонуп, тыныгуу жаратуу

for i in range(3):
      print(".....")
      time.sleep(3)

print("(Сиз күткөндү жаман көрөсүз деп ойлойм!)")
print("")

# Супер баатырдын атын туш келди тандоо

# Муну эки тизменин ар биринен бирден ат тандоо менен жасайбыз

# Жана аны суперАтальш өзгөрмөсүнө кошуу
```

```
суперАталыш = random.choice(суперЫсымдар)+ " " +
random.choice(суперФамилиялар)

print("Сиздин Супер Баатырдын аты: ")
print(суперАталыш)
print("")
print("Эми сизде кандай супер күч бар экендигин көрүүгө кез келди!)")
print("(супер баатырдын күчүн чыгаруу...)")

# Дагы бир жолу эффект жаратууда

for i in range(2):
     print(".....")
     time.sleep(3)

print("(аах... БУЛ сизге жакпайт...)")
for i in range(2):
     print(".....")
     time.sleep(3)
print("(дээрлик даяр....)")
# Муну суперКүчтөр тизмесинен туш келди тандоо менен жасайбыз
# Жана аны күч өзгөрмөсүнө кошобуз
Күч = random.choice(суперКүчтөр)
print("Сиздин жаңы күчүңүз: ")
print(Күч)
print("")
print("Акыркы, бирок, маанилүү, Сиздин статистикаңызды чыгаралы!")
print("Сиз супер акылдуу болосузбу? Супер күчтүү? Супер сулуу?")
print("Муну билүүгө убакыт келди!")
#  Эффект жаратуу жана программаны кайрадан жайлатуу
for i in range(3):
     print(".....")
     time.sleep(3)
```

```
# Төмөндө берилген өзгөрмөлөрдүнүн ар бирине жаңы маани берүү
# Жаңы маанилер 1-20 арасы болот

Зээндүүлүк = random.randint(1,20)
Тапкычтык = random.randint(1,20)
Чыдамдуулук = random.randint(1,20)
Акылмандык = random.randint(1,20)
Дисциплина = random.randint(1,20)
Эптүүлүк = random.randint(1,20)
Ылдамдык = random.randint(1,20)

# Статистиканы басып чыгаруу

print("Сиздин жаңы статисткаңыз: ")
print("")
print("Зээндүүлүк: ", Зээндүүлүк)
print("Тапкычтык: ", Тапкычтык)
print("Чыдамдуулук: ", Чыдамдуулук)
print("Акылмандык: ", Акылмандык)
print("Дисциплина: ", Дисциплина)
print("Эптүүлүк: ", Эптүүлүк)
print("Ылдамдык: ", Ылдамдык)
print("")

# Супер баатырдын толук баяндамасын басып чыгаруу
# Буга баатырдын аты, супер күчү жана статистикасы кирет

print("Жаңы Супер Баатырыңыздын толук баяндамасы!")
print("Супер Баатыр генератору 3000 колдонуп жатканыңыз үчүн рахмат!")
print("Бардык досторуңузга айтып бериңиз!")
print("")
print("Персонаждын кыскача баяндамасы:")
print("")
print("")
print("Супер Баатырдын аты: ", суперАталыш)
print("Супер күчү: ", Күч)
print("")
print("Зээндүүлүк: ", Зээндүүлүк)
```

```
print("Тапкычтык: ", Тапкычтык)
print("Чыдамдуулук: ", Чыдамдуулук)
print("Акылмандык: ", Акылмандык)
print("Дисциплина: ", Дисциплина)
print("Эптүүлүк: ", Эптүүлүк)
print("Ылдамдык: ", Ылдамдык)
```

Бул программаны иштетип жатканда, супер баатырдын аты, супер күчү жана статистикасы ар башка болорун эске алып, төмөнкүлөргө окшош натыйжаны көрүшүңүз керек. Анткени алардын бардыгы кокустан пайда болгон. Мен билем, мен бузулган пластинкага окшошуп атамын!

Мүмкүн болгон жыйынтык:

```
Супер Баатыр генератору 3000 менен супер баатыр жаратууга даярсызбы?
Клавиатурадан О/Ж киргиз:
о
Атыбызды туш келди тандап жатат...
...........
...........
...........
(Сиз күткөндү жаман көрөсүз деп ойлойм!)

Сиздин Супер Баатырдын аты:
Укмуш Мачо

Эми сизде кандай супер күч бар экендигин көрүүгө кез келди!)
(супер баатырдын күчүн чыгаруу...)
...........
...........
(аах... БУЛ сизге жакпайт...)
...........
(дээрлик даяр....)
...........
(дээрлик даяр....)
Сиздин жаңы күчүңүз:
Укмуш Күчтүү

Акыркы, бирок, маанилүү, Сиздин статистиканы чыгаралы!
Сиз супер акылдуу болосузбу? Супер күчтүү? Супер сулуу?
Муну билүүгө убакыт келди!
...........
```

```
...........
...........
Сиздин жаңы статисткаңыз:

Зээндүүлүк:  18
Тапкычтык:  14
Чыдамдуулук:  1
Акылмандык:  5
Дисциплина:  8
Эптүүлүк:  1
Ылдамдык:  4

Жаңы Супер Баатырыңыздын толук баяндамасы!
Супер Баатыр генератору 3000 колдонуп жаткаңыз үчүн рахмат!
Бардык досторуңузга айтып бериңиз!

Персонаждын кыскача баяндамасы:

Супер Баатырдын аты:  Укмуш Мачо
Супер күчү:  Укмуш Күчтүү

Зээндүүлүк:  18
Тапкычтык:  14
Чыдамдуулук:  1
Акылмандуулук:  5
Дисциплина:  8
Эптүүлүк:  1
Ылдамдык:  4
```

7 – БӨЛҮМ

ФУНКЦИЯЛАР, МОДУЛДАР ЖАНА ОРНОТУЛГАН ФУНКЦИЯЛАР

Эми биз расмий түрдө алгач толук кандуу Python тиркемесин түзүп алганыбыздан кийин (эгер 6-бөлүмгө кайрылбай кетсеңиз!) программалоо мүмкүнчүлүктөрүн чындап колдонууну үйрөнүп, мыкты программист боло алабыз.

Буга чейин бул китепте биз кодубузду мүмкүн болушунча натыйжалуу болушу маанилүү экенине токтолдук. Коддоо бир күндө жасай турган жумуштун көлөмүнүн натыйжасын көбөйтүп гана тим болбостон, дагы бир катар артыкчылыктарга ээ. Биринчиден, биздин программалар компьютердин эс-тутумун жана иштөө кубаттуулугун мүмкүн болушунча аз колдонушун камсыз кылууга жардам берет. Экинчиден, биздин коддогу каталардын санын азайтууга жардам берет. Экинчисине жетишкендигинин себеби айдан ачык. Биз канчалык аз басып чыгарсак, туура эмес нерсени терүү же программалоонун логикалык же синтаксистик каталарды кетирүү мүмкүнчүлүгү аз болот.

Натыйжалуу иштөөнүн бир бөлүгү жаңы программаларды түзүп жатканда сыналган жана текшерилген коддун үзүндүлөрүн кайра-кайра колдонууну камтыйт. Бул код бөлүктөрү көбүнчө жалпы милдеттерди аткаруу үчүн жазылат жана бир нече жөнөкөй коддон баштап, миңдеген сапка чейин жетиши мүмкүн. Бирок, эң негизгиси, биз алардын иштээрин билебиз жана ал коддордун бардыгын кайра-кайра терүүнүн ордуна, аларды өз файлдарында сактап, программаларыбызга керектүү түрдө импорттоп, тонналаган убакытты үнөмдөйбүз жана каталардан алыс болобуз.

Ушул жол менен колдонулганда, коддун бул үзүндүлөрү модулдар деп аталат. Жөнөкөй тил менен айтканда, модуль - бул кодду камтыган файл. Дал ушундай.

Ушул убакка чейин бул китепте бир нече модулдарды, анын ичинде убакытты жана кокустукту колдондук. Бул бөлүмдө биз өзүбүздүн модулдарды түзүүнү гана үйрөнбөстөн, Python сунуш кылган эң популярдуу жана кеңири колдонулган модулдарды дагы карап көрөбүз. Кантсе да Python программасына орнотулган, кеңири сыналган жана ошондой эле ишенимдүү чоң Python жамааты тарабынан түзүлгөн Python модулдары, аны программалоону баштоо үчүн башынан эле ушунчалык күчтүү жана маанилүү программалоо тилине айландырган нерселердин бири болуп саналат.

Анда эмесе, жүгөрү аралаш даамдуу куурулган картошканызды ары коюп, колуңузга жугуп калган сырды аарчып салыңыз. Программа жасоо мүмкүнчүлүгүңүздү андан ары кеңейтиш үчүн даярданыңыз, анткени биз супер баатыр программисттин негизги куралы болгон модулдарга тереңирээк сүңгүйбүз!

Модулдарды аныктоо

Модуль деген эмне экендигин билип алгандан кийин “модуль эмнени камтыйт” деп ойлонушуңуз мүмкүн. Мурунку аныктамабызга ылайык, модуль каалаган кодду камтый алат. Ал функциялардын жыйындысына ээ болушу мүмкүн, колдонуучунун экранына бир топ текст жазуу үчүн сценарий болушу мүмкүн, өзгөрмө тизмесин камтышы мүмкүн же башка модулдарды программаңызга импорттогон бир нече сап коду болушу мүмкүн.

Кодду камтыган Python файлдары(.py) модуль болот. Python тилинде техникалык жактан үч түрдөгү модуль бар. Алар:

• Ички орнотуулар;

• Колдонмо программалардын пакети;

• Өз алдынча аныкталган / колдонуучу тарабынан түзүлгөн.

Ички орнотуулар

Built-ins – Ички - орнотулган модулдар Python китепканасынын стандарттуу бөлүгү болгон модулдарды жана функцияларды билдирет. Python жумушчу мейкиндигин орнотуу учурунда бул модулдар анда алдын-ала коюлат. Аларга datetime (маалыматтын датасы жана убакттын түрлөрү менен иштөөгө мүмкүндүк берет), random (сандарды туш келди түзүү үчүн колдонулат) жана SocketServer (тармак серверинин алкактарын түзүү үчүн) сыяктуу пайдалуу функциялар кирет.

Сиз буга чейин орнотулган бир канча материалдар менен таанышкансыз. Анткени биз аларды бул китепте мисал катары колдондук. Python менен шайкеш келген бир нече орнотулган модулдар бар. Толук тизмени көрүү үчүн төмөнкү дарекке кире аласыз: https: //docs.python. org/ 3.7 /py-modindex.html. Бирок, бул тизме ар бир версияга жараша өзгөрүп турарын эске алыңыз. Андыктан Python.org сайтына киргенде, иштеп жаткан Python версиясын текшериңиз.

Албетте, орнотулган Python модулдарынын тизмесин көрүүнүн оңой жолу - жөн гана төмөнкү коддун бөлүгүн колдонуу:

```
# Python Built-in модулдарын басып чыгаруу
print(help(“modules”))
```

Бул кодду иштеткенде Python сиз орноткон бардык орнотуулардын тизмесин басып чыгарат. Ал төмөндөгүлөр:

Сураныч, мен бардык жеткиликтүү модулдардын тизмесин чогултканча бир аз күтө туруңуз…

BooleanExamples	_testmultiphase	gettext	reprlib
BooleanLogic	_thread	glob	rlcompleter
ConditionalStatements	_threading_local	grep	rpc
Count10	_tkinter	gzip	rstrip
DoomsdayClock	_tracemalloc	hashlib	run
Example1	_warnings	heapq	runpy
InfiniteLoop	_weakref	help	runscript
LearningText	_weakrefset	help_about	sched
ListExample	_winapi	history	scrolledlist
LogicalOperatorsExample	abc	hmac	search
MathIsHard	aifc	html	searchbase
MultipleElifs	antigravity	http	searchengine
OrExample	argparse	hyperparser	secrets
PowersWeaknesses	array	idle	select
RandomGenerator	ast	idle_test	selectors
...			

Албетте, орнотулган модулдардын тизмесин көрүү абдан сонун, бирок онлайн режимине кирип, аларды Google издөө системасынан издебей эле, чындыгында, эмне кылууну билип алганыбыз оң. Бактыга жараша, Python программасында буга жардам берген эки орнотуу бар!

Биринчиси - .__ doc__, ошондой эле *docstring* же документ катары белгилүү. Сиз кезиккен ар бир модулда анын аныктамасынын бөлүгү катары docstring болушу керек. Ал негизинен функция же модуль колдонулуп жаткан "документти" тейлейт. Модулдун документтерин окуу үчүн биз аны төмөнкүдөй импорттоого болот:

```
# Алгач модулду импорттошубуз керек

import time

# Андан кийин биз анын документин басып чыгара алабыз

print(time.__doc__)
```

Документте көргүңүз келген модулдун аталышы .__ doc__ командасынын алдында пайда болот.

Эгер сиз бул кодду файлга киргизип, иштетсеңиз, анда сиздин натыйжаңыз мындай болмок:

This module provides various functions to manipulate time values.
There are two standard representations of time. One is the number
of seconds since the Epoch, in UTC (a.k.a. GMT). It may be an integer
or a floating point number (to represent fractions of seconds).
The Epoch is system-defined; on Unix, it is generally January 1st, 1970.
The actual value can be retrieved by calling gmtime(0).
The other representation is a tuple of 9 integers giving local time.
The tuple items are:
year (including century, e.g. 1998)
month (1-12)
day (1-31)
hours (0-23)
minutes (0-59)
seconds (0-59)
weekday (0-6, Monday is 0)
Julian day (day in the year, 1-366)
DST (Daylight Savings Time) flag (-1, 0 or 1)
If the DST flag is 0, the time is given in the regular time zone;
if it is 1, the time is given in the DST time zone;
if it is -1, mktime() should guess based on the date and time.

Документтерди көрүү үчүн дагы бир вариант бар. Чындыгында, эки вариантты тең колдонсоңуз болот. Анткени эки командандын документтери эки башка болушу мүмкүн. Мисалы, сиз бул кодду киргизиңиз:

```
# Алгач модулду импорттошубуз керек

import time

# Андан кийин биз анын документациясын басып чыгара алабыз

help(time)
```

жана аны иштетсеңиз, .__ doc__, колдонулганга караганда башкача, кеңири жооп аласыз:

```
Help on built-in module time:

NAME

time - This module provides various functions to manipulate time values.

DESCRIPTION

    There are two standard representations of time.  One is the number
    of seconds since the Epoch, in UTC (a.k.a. GMT).  It may be an integer
    or a floating point number (to represent fractions of seconds).
    The Epoch is system-defined; on Unix, it is generally January 1st, 1970.
    The actual value can be retrieved by calling gmtime(0).
    The other representation is a tuple of 9 integers giving local time.
    The tuple items are:
        year (including century, e.g. 1998)
        month (1-12)
        day (1-31)
        hours (0-23)
        minutes (0-59)
        seconds (0-59)
        weekday (0-6, Monday is 0)
        Julian day (day in the year, 1-366)
        DST (Daylight Savings Time) flag (-1, 0 or 1)
        If the DST flag is 0, the time is given in the regular time zone;
        if it is 1, the time is given in the DST time zone;
        if it is -1, mktime() should guess based on the date and time.
```

Бул жыйынтык иш жүзүндө басып чыгарыла турган документтин кичинекей гана бөлүгүн түзөт. Айырмасын көрүү үчүн экөөнү бир эле учурда колдонуп көрүңүз.

Бул кодду printDocumentation.py деген аталыштагы жаңы Python файлына киргизип, айырмачылыктарын көрүү үчүн натыйжаларын карап чыгыңыз:

```
# Алгач документин көрө турган модулду импорттошубуз керек

import time

# .__ doc__ же docstring колдонуп документтерди басып чыгаруу

print (time.__doc__)

# Биз көрө тургандай кылып бөлүүчү сызык түзүү
# Документтин кийинки топтому башталат.

print("HELP колдонуунун көрүнүшү мына ушундай...")
print("########################################")

# help() колдонуу менен документтерди басып чыгаруу

help(time)
```

Бул кодду иштетүүнүн натыйжасын бул китепке киргизүү өтө эле көп бет талап кылат. Бирок бардык документтерди көрүү үчүн программаны өзүңүз иштетсеңиз болот. Документти кайсы метод менен окуганыңызды белгилеп коюңуз.

Колдонмо программалардын пакети

Модулду импорттоодон мурун эгерде ал алдын ала Python орнотууда кошулган эмес болсо, алгач аны орнотушуңуз керек. Пакетти орнотуунун стандарттык эмес же коомчулук тарабынан иштелип чыккан ыкмасы - бул ички орнотулган функцияны колдонуу. Бул ыкма pip. деп аталат. Python иштөө мейкиндигинин учурдагы көпчүлүк версияларында pip автоматтык түрдө орнотулат. Андыктан анын мурунку версиясын колдонбосоңуз, анда баары даяр болушу керек.

pip - бул Python 3.4 жана андан жогору версиялары менен кошо орнотулган программа. Программаны колдонуу үчүн сиз буйрук сабынын терезесин ишке киргизишиңиз керек (муну Start, андан кийин run, андан кийин CMD терүү аркылуу иштете аласыз). Ошол жерден, жөн гана буйрук тилкесине "pip" деп терүү менен мүмкүн болгон pip буйруктарынын тизмесин көрө аласыз.

Азырынча бир жөнөкөй pip буйругун түшүнүшүңүз керек. Бирок, аны колдонуудан мурун ар дайым өзүбүз орнотууну каалаган пакеттин бардыгын текшерип алышыбыз керек.

Бул үчүн биз IDLE терезесине кайтып келип төмөнкүнү теребиз:

```
import <модулдун аталышы>
```

Мисалы, *time* модулу орнотулганбы же жокпу, билгибиз келсе, мындай деп теребиз:

```
import time
```

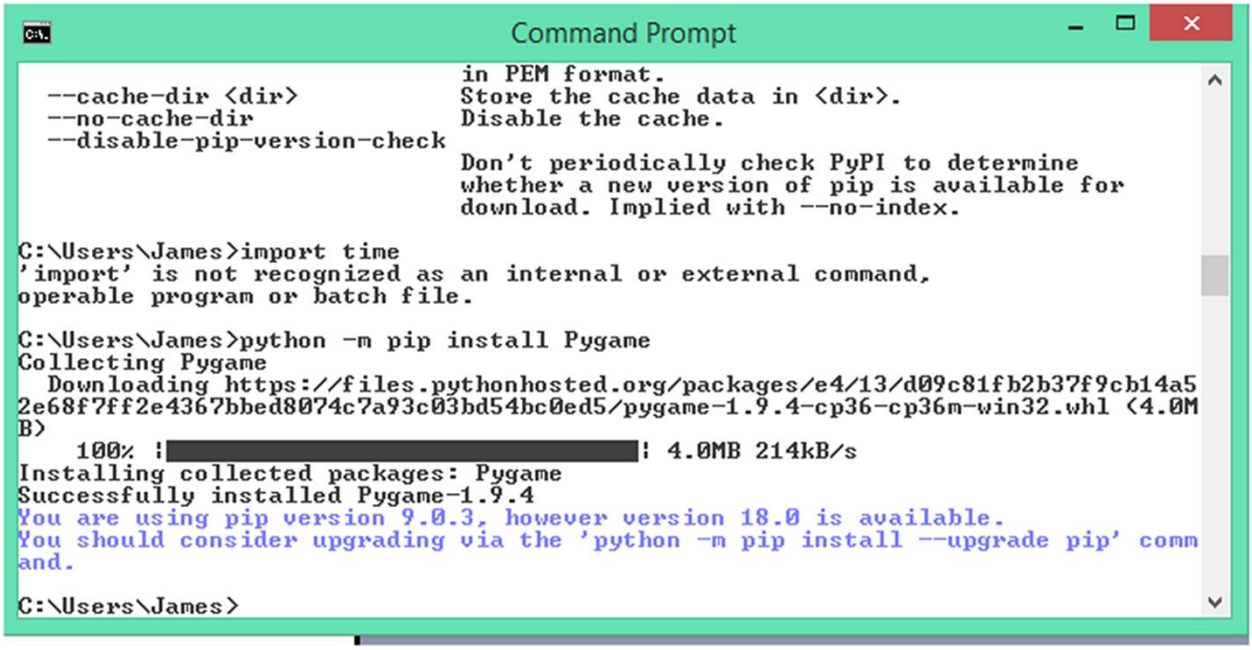
Эгер ката деп чыкса, анда так ушул модуль орнотула электигин билебиз.

Муну оңдоо үчүн биз буйрук сабыбызга, модулду орнотобуз. Биздин мисал үчүн видео оюнун өнүктүрүүгө жардам берген популярдуу пакет болгон Pygame пакетин колдонолу (муну кийинки бөлүмдө талкуулайбыз).

Буйрук тилкесинде жөн гана төмөндөгүнү киргизиңиз:

```
python -m pip install Pygame
```

Бир нече секунддан кийин буйрук сабы пакетти жүктөө жана орнотуу процессин баштайт. Бүткөндөн кийин сиз 7-1-сүрөткө окшогон билдирүүнү көрө аласыз.



7-1-сүрөт. *Куттуктайбыз! Сиз биринчи Python пакетиңизди орноттуңуз!*

Өзүңүздүн модулуңузду түзүңүз

Мурда орнотулган модулдарды жана топтомдорду колдонуу программаларыңызды натыйжалуу кылат жана каталарга аз кабылтат. Убактыңызды үнөмдөп, башыңызды оорутпай турган (көбүнчө түндөсү программаңыздагы катаны издегенде же Мета́ллика группасын укканыңызда) дагы бир курал - бул сиз кайра - кайра колдоно турган жеке өзүңүздүн модулдарыңызды жаратуу.

Биздин модулду түзүүнүн биринчи бөлүгү - бул башка программага кайрылып же шилтеме бере турган функцияны түзүү. Бул көнүгүү үчүн бизге эки Python файлы керек болот. Негизги программабызды колдоно турган чыныгы модулду түзүүдөн баштайбыз.

OurFirstModule.py деп аталган файл түзүп, төмөнкү кодду киргизиңиз:

```
# def  ыкмасын колдонуп функцияны аныктаңыз

def firstFunction():

     print("Бул биздин биринчи функциябыз!")
```

Файлды сактап, аны иштетип көрүңүз. Программа чындыгында иштеп жаткандыгын көрүп турсаңыз да, эч нерсе болбойт. Себеби, биз өзүбүздүн функцияны аткара турган нерсени гана аныктадык, бирок аны чакыра элекпиз. Ага кайрылган жокпуз жана ага эч нерсе жаса деп айта элекпиз.

Бул функцияны иш жүзүндө колдонуу үчүн биз аны башка файлдан чакырышыбыз керек.

TestingModule.py деген башка файл түзүп, төмөнкү кодду киргизиңиз:

```
# Алгач модулубузду импорттошубуз керек
# Биз модулду импортоодо файлдын аталышын жазабыз

import ourFirstModule

# Эми биз аны колдонуу үчүн функцияны чакырабыз

ourFirstModule.firstFunction()
```

Бул файлды иштетип жатканда төмөнкү натыйжаны көрүшүңүз керек:

```
Бул биздин биринчи функциябыз!
```

Куттуктайбыз! Сиз биринчи модулуңузду түзүп, аны башка программанын ичинен ийгиликтүү чакырдыңыз!

Албетте, модулдар бир нече функцияны аткара алышат, андыктан дагы бир нече функцияны кошуп, аларды файлга чакырып көнүгүп көрөлү. Биздин FirstModule.py файлынын көчүрмөсүн ачып, кодду төмөнкүдөй кылып тууралаңыз:

```
# Биринчи функцияңызды аныктаңыз

def firstFunction():

     print("Бул биздин биринчи функциябыз!")

# Экинчи функцияңызды аныктаңыз

def secondFunction():

     print("Кара, экинчи функция!")

# Өзгөрмө аныктаңыз

a = 2+3
```

Андан кийин жаңы аныкталган функцияларыбызды жана өзгөрмөлөрдү колдонуу үчүн testModule.py файлын оңдошубуз керек. Кодуңузду төмөнкүдөй өзгөртүңүз:

```
# Алгач модулубузду импорттошубуз керек

# Биз модулду импортоодо файлдын аталышын жазабыз, узартылышын-.ру ды жазбайбыз

import ourFirstModule

# Эми биз аны колдонуу үчүн функцияны чакырабыз

FirstModule.firstFunction()

# Экинчи функцияны чакырабыз

FirstModule.secondFunction()

# Биздин модулдагы өзгөрмөнү чакырып жана басып чыгаруу

print("a өзгөрмөсүнүн мааниси: ",FirstModule.a)
```

Бул код бир эмес, эки функцияны чакыргандан тышкары *a* деп аталган өзгөрмөбүздүн маанисин да чыгарат. Биз буга коддун басып чыгарылышы аркылуу жетишебиз (ourFirstModule.a). OurFirstModule бөлүгү ourFirstModule.py файлына шилтеме берип, программага функцияны кайдан алып келүү керектигин айтса, *.a* кандай өзгөрмөнү басып чыгаруу керектигин айтат. Эгерде биздин өзгөрмө lastName деп аталып калган болсо, анда анын ордуна мындай болмок: print (ourFirstModule.lastName).

Биз жараткан бардык коддордо болгондой эле, биз ар дайым өз ишибизди документтештирип алганды каалайбыз. Буга чейин, .__ doc__ жана help() модулдарын документтерди басып чыгаруу үчүн колдонгонбуз. Эми биз өзүбүздүн документтерибизди түзүү үчүн көп саптуу комментарийлерди колдонобуз (же үч топтом ").

Биздин FirstModule.py файлыбызды ачыңыз жана ага төмөнкү комментарийлерди кошуу менен биринчи функциянын - firstFunction() кодун өзгөртүңүз:

```
# Функцияңызды аныктаңыз

def firstFunction():
    """ Бул firstFunction () - же docstring үчүн документация

Колдонуунун мисалдарын же жөн гана функциянын документациясын бул жерге койсок болот. Ошентип, келечектеги программисттер - же өзүбүз кийинчерээк - "helpfile" биздин firstFunction үчүн жазылганын жана ал эмнеге арналганын окуй алабыз

"""

print("Бул биздин биринчи функциябыз!")
```

Биринчи """ чегинүү топтому менен """ акыркы топтомунун ортосундагы нерселердин бардыгы мурунку бөлүмдө талкуулангандай, комментарий же документтештирүү катары каралат.

Эми, TestingModule.py файлыңызды ачып, документтерди басып чыгаруу үчүн, ага төмөнкү кодду кошолу:

```
# firstFunction() үчүн helpfile басып чыгаруу
help(FirstModule)
```

Бул кодду сиз файлдын каалаган жерине жайгаштыра аласыз, бирок мен аны firstFunction функциясын басып чыгарган print() функциясынын астына жайгаштырууну туура көрдүм.

Программаны иштетип, төмөнкү натыйжаларды көрүшүңүз керек:

```
Бул биздин биринчи функциябыз!
Help on module FirstModule:
NAME
    FirstModule - # Биринчи функцияңызды аныктаңыз
FUNCTIONS
    firstFunction()
        Бул firstFunction () - же docstring үчүн документация
Колдонуунун мисалдарын же жөн гана функциянын документациясын бул жерге койсок болот. Ошентип, келечектеги программисттер - же өзүбүз кийинчерээк - "helpfile" биздин firstFunction үчүн жазылганын жана ал эмнеге арналганын окуй алабыз
    secondFunction()
DATA
    a = 5
Кара, экинчи функция!
a өзгөрмөсүнүн маанизи:  5
```

Жалпы орнотулган функциялар

Python көптөгөн орнотулган ички функцияларга бай. Бул китепке анын көптөгөн мүмкүнчүлүктөрүн камтыдык. Бирок, ишенимдүү жардамчы болгон колуңуздагы шаймандар эч качан ашыкча болбойт. Албетте, акуланы коркута турган банкадагы репеллент күлкүлүү сезилиши мүмкүн. Бирок сиз Тигил-Демин - Кармай -

Алган – жана- Дагы- Акула - Менен - Сүйлөшө - Алган - Киши менен согушканга чейин күтө туруңуз. Эми акулага каршы күрөшүү канчалык акылсыз деп ойлойсуз?

70ке жакын орнотулган функциялар бар. Алардын көпчүлүгүн сиз жашоодо программист катары колдоносуз. Азырынча ушул кезге чейин сөз кылбаган бир нече кеңири таралган функцияларды талкуулайбыз. Биз аларды сап функцияларынан баштап категориялары боюнча чечмелейбиз.

Сап функциялары

Сап функциялары, сиз болжолдогондой, саптарда иштеген функциялар. Биз алардын айрымдарын, анын ичинде str.upper() жана str.lower(), алардын катарларын, тиешелүүлүгүнө жараша чоң жана кичине тамгага айландырганбыз.

Ошондой эле, чындыгында, сапты чоң же кичине регистрде түзүү менен сиз саптын мазмуну кандай экендигин карап текшере аласыз. Мисалы, колдонуучу бардыгын чоң тамгалар менен жазган-жазбаганын билгиңиз келеби? Текшерүү үчүн төмөнкү кодду колдонсоңуз болот:

```
# Сөздөрү баш тамгалардан гана турган сүйлөм түзүңүз
тестСап = "МЕН КЫЙКЫРЫП ЖАТАМ!"
print("КОЛДОНУУЧУ КЫЙКЫРЫП ЖАТАБЫ??")

# тестСап өзгөрмөсүнүн маанисинин баары чоң тамгалардан турарын текшерип көрүңүз
print(тестСап.isupper())
```

Бул учурда, str.isupper() деп аталган сап функциясын колдонуп, сапта баш тамгалар бар экендигин текшеребиз. Эгер сиз ушул кодду иштете турган болсоңуз, анда логикалык жооп аласыз (True - Чын же False - Жалган):

```
КОЛДОНУУЧУ КЫЙКЫРЫП ЖАТАБЫ?
True
```

Эгерде саптагы тамгалардын бардыгы кичине болсо, анда анын ордуна ***False*** мааниси келип чыгат. Анткени функция толугу менен баш тамгаларды камтыгандыгын текшерип жатат.

Эгерде ал кичинекей тамга экендигин текшерип көргүбүз келсе, анда str.islower() сап функциясын колдонмокпуз:

```
# Сөздөрү баш тамгалардан гана турган сүйлөм түзүңүз
тестСап = "МЕН КЫЙКЫРЫП ЖАТАМ!"

print("Колдонуучу кыйкырып жатабы??")
```

```
# тестСап өзгөрмөсүнүн бардык маанилери кичинекей тамгалардан турарын текшерип көрүңүз
print(тестСап.islower())
```

Албетте, ушул учурда False кайтарды.

Колдонуучу кандай символдорду киргизгенин текшергибиз келген учурлар болот. Мисалы, колдонуучу форманы толтуруп жатса, биз алардын аттарын жана фамилияларын билгибиз келсе, алардын сандык маанисин киргизишин каалабайбыз. Көңүл буруңуз, албетте, эгер алар роботтор же келгиндер болбосо.

Сапта тамгалардын гана бар экендигин (жана сандардын жоктугун) текшерүү үчүн, str.isalpha() ны колдонуңуз:

```
# Өзгөрмө тамгалардан гана түзүлгөнүн текшерүү үчүн сап түзүңүз
аты = "Асан8"

# аты сабынын маанисинде сан камтылбагандыгын текшериңиз
print("Сенин атың сандардан турабы?")

if аты.isalpha() == False:
      print("Сен эмнесиң, роботпу?")
```

аты сабынын мааниси тамгаларды гана камтыбагандыктан (анда сан бар), анда if (*False*) маани берилип, print() функциясы өзүнүн текстин басып чыгарат:

```
Сенин атың тамгалардан турабы?
Сен эмнесиң, роботпу?
```

Эгерде *аты тамга* белгилерди *гана* камтыса (A – Z жана a – z), анда if True деп кайтып келип, эч нерсе басылмак эмес.

Ошондой эле, маанилердин сан экендигин же жөн эле сандарды камтыгандыгын текшере алабыз. Мисалы, кимдир бирөө социалдык камсыздоо же телефон номери талаасына тамганы киргизбей жаткандыгын текшергибиз келет. Саптагы гана маанилерди текшерүү үчүн str.isnumeric() функциясын колдонобуз:

```
 # Өзгөрмө сандардан гана түзүлгөнүн текшерүү үчүн сап түзүңүз
колдонуучуIQ = "2000"

# колдонуучуIQ өзгөрмөсүнүн маанисинде тамга камтылбагандыгын текшериңиз
if колдонуучуIQ.isnumeric() == False:
        print("Суранам, сандар гана!")
else:
        print("Куттуктайм, сиз сан менен тамганын айырмасын билесиз!")
```

Дагы бир жолу, userIQ сандарды гана камтыгандыгын баалоонун натыйжасы True (Чын) же False (Жалган) экенин текшеребиз. userIQ сандарды гана камтыгандыктан жана тамгалар жок болгондуктан, натыйжа True болуп, төмөндөгү натыйжага жетишебиз:

```
Куттуктайм, сиз сан менен тамганын айырмасын билесиз!
```

Ошондой эле, биздин саптар боштуктарды камтый тургандыгын текшере алабыз. Ал боштук деп да аталат. Ал үчүн str.isspace() функциясын колдонобуз:

```
# колдонуучуIQ мааниси боштуктарды же боштукту камтыбаганын текшер
if колдонуучуIQ.isspace() == True:
        print("Сураныч, боштуктардан башка маанини киргизиңиз!")
```

колдонуучуIQ бардык боштуктарды камтыбагандыктан, эч нерсе болбойт. Эгерде ал боштуктарга гана толгон болсо, анда Python биз аныктаган print() функциясын аткармак.

Биз колдоно турган дагы бир пайдалуу сап функциясы - бул саптагы белгилердин санын эсептөөгө мүмкүнчүлүк берген len(). Сиз өзүңүзгө: "Мунун мага эмне кереги бар?" деп сурашыңыз мүмкүн. Жооп жөнөкөй: сиз өзгөрмө ичиндеги белгилердин санын, мисалы, сыр сөздү чектей аласыз же анын белгилери жетиштүү экенин текшере аласыз.

Же, балким, сиз да мен сыяктуу тактыкты жакшы көрүп, бардыгын эсептөөнүн зарылдыгы бар деп ойлойсуз. Мен муну менин көптөгөн супер күчтөрүмдүн бири деп эсептейм...

Саптагы белгилердин санын эсептөө үчүн төмөнкүдөй кодду колдонсоңуз болот:

```
# Len() колдонуп белгилердин санын эсептөө үчүн өзгөрмө түзүңүз

сырСөзТест = "МенинсырСөзүмсырСөз!"

print(len(сырСөзТест))
```

Бул кодду иштеткенде төмөндөгүдөй натыйжага ээ болосуз:

```
20
```

Сан функциялары

Эми бир нече жаңы сап функцияларын үйрөнгөндөн кийин сандар менен иштөөгө өтөлү. Буга чейин биз сандар менен иштөөгө жардам берген бир нече функцияларды, ошондой эле мээбизге зыян келтирбестен, элүү математикалык теңдемени аткарууга мүмкүнчүлүк берген операторлорду изилдедик (өтө эле көп болуп кетти окшойт).

Мээбизди сандар менен иштөөдө мындан дагы акылдуурак кылыш үчүн программалоо чеберчилигибизди өркүндөтүп, белгилүү ракета жасаган окумуштууларга окшоштура турган дагы бир нече өзгөчөлүктөрдү карап көрөлү.

Кээде цифралар менен иштөөдө, биздин кожоюндарыбыз кайсы сан баарынан жогору экендигин айтып берүүнү сурап калышат. Сандар катарындагы кайсы сан максималдуу экендигин билүү үчүн max() колдонобуз.

```
# Сандардын тобун камтыган тизме түзүңүз

студентБаллы = [100, 90, 80, 70, 60, 50, 0]

# Max() колдонуп, студентБаллы тизмесиндеги эң жогорку санды табыңыз

print("Тизмедеги эң жогорку балл кайсы?")
print:("Жооп: ")
print(max(студентБаллы))
```

Эгер биз ушул кодду иштете турган болсок, анда төмөнкүлөр пайда болмок:

```
Тизмедеги эң жогорку балл  кайсы?
100
```

Себеби 100 биздин *студентБаллы* тизмесиндеги эң жогорку мааниге ээ. Эгерде биз сандардын тизмесинин минималдуу маанисин билгибиз келсе, анда min() колдонобуз:

```
# Сандардын тобун камтыган тизме түзүңүз

студентБаллы = [100, 90, 80, 70, 60, 50, 0]

# Max() колдонуп, студентБаллы тизмесиндеги эң жогорку санды табыңыз

print("Тизмедеги эң жогорку балл кайсы?")
print:("Жооп: ")
print(max(студентБаллы))

# Min() колдонуп, студентБаллы тизмесиндеги эң төмөнкү санды табыңыз

print("Тизмедеги эң төмөнкү балл кайсы?")
print:("Жооп: ")
print(min(студентБаллы))
```

Иштетилгенден кийин бул код мындай жыйынтык чыгарат:

```
Тизмедеги эң жогорку балл кайсы?
100
Тизмедеги эң төмөнкү балл кайсы?
0
```

Ошондой эле, тизме түзбөстөн min() жана max()ти колдонсок болот. Аларды өз-өзүнчө колдонуу үчүн төмөнкүнү теришиңиз керек:

```
print(min(100, 90, 80, 70, 60, 50, 0))
print(max(100, 90, 80, 70, 60, 50, 0))
```

Эскертүү! Ошондой эле, саптарда min() жана max() белгилерин колдонсоңуз болот. Мисалы, алфавитте min колдонуп, a дан z ге чейин көрсөткөндө, “a” min(), ал эми “z” болсо, max() менен кайтарылат.

Дагы бир кеңири тараган практика - берилген тизмедеги сандардын бардыгын кошуу. Балким, сиз компания жамаатынын жалпы эмгек акысын же иштеген саатын эсептеп чыгышыңыз керек. Ал үчүн sum() функциясын колдоносуз. Келгиле, аны төмөнкү код менен аткаралы:

```
# Көбүрөөк сандарды камтыган, кызмат акы көрсөткөн башка тизме түзүңүз
жалпыАкы = [500, 600, 200, 400, 1000]
# Sum() колдонуп, тизмедеги сандардын суммасын эсептеңиз
print("Ушул жумада кызматкерлерге канча кызмат акы төлөдүк?")
print("Жалпы кызмат акы:  ")
print(sum(жалпыАкы))
```

Бул мисалдын натыйжасы:

```
Ушул жумада кызматкерлерге канча кызмат акы төлөдүк?
Жалпы кызмат акы:
2700
```

Жаңы функцияларыңыз менен машыгыңыз

Сиздин “супер баатыр” программалоо куржунуңузга көптөгөн жаңы функцияларды коштук. Чеберчиликти өркүндөтүү үчүн үйрөнгөнүңүздү иш жүзүндө

колдонууга убакыт келди. Андан кийин сиз үйрөнгөн жаңы сап функцияларынын тизмесин жана ушул бөлүмдө биз ойногон жаңы сандык / математикалык функциялардын тизмесин таба аласыз.

Бул кодду өз алдынча киргизүүдөн тартынбаңыз жана ушул жөнөкөй, бирок күчтүү функцияларды колдонуунун жаңы жана кызыктуу жолдорун билип алыңыз.

Сап функцияларына мисалдар

```
# Сөздөрү баш тамгалардан гана турган сүйлөм түзүңүз

тестСап = "МЕН КЫЙКЫРЫП ЖАТАМ!"

# Өзгөрмөдө тамгалар гана бар экендигин текшерүү үчүн сап түзүңүз

аты = "Асан8"

# Өзгөрмөдө сандар гана бар экендигин текшерүү үчүн сап түзүңүз

колдонуучуIQ = "2000"

# len() колдонуп белгилердин санын эсептөө үчүн өзгөрмө түзүңүз

тестСырСөз = "МенинсырСөзүмсырСөз!"

# Төмөндө бир катар функциялар текшерилет

print("Колдонуучу кыйкырып жатабы?")

# тестСап өзгөрмөсүнүн маанисинин бардыгы чоң тамгалардан турарын текшерип көрүңүз

print(тестСап.isupper())

# аты өзгөрмөсүнүн мааниси кандайдыр бир сандарды камтыгандыгын текшериңиз

print("Сенин атың сандардан турабы?")

if аты.isalpha() == False:

print("Сен эмнесиң, роботпу?")

# колдонуучуIQ сандарды гана камтыйбы же тамгаларды дагыбы текшериңиз

if колдонуучу.isnumeric() == False:
     print("Суранам, сандарды гана!")
else:
```

```
    print("Куттуктайм, сиз сан менен тамганын айырмасын билесиз!")

# колдонуучуIQ мааниси бир канча боштуктарды же бир боштукту камтый тургандыгын текшериңиз

if колдонуучуIQ.isspace() == True:
    print("Сураныч, боштуктардан башка маанини киргизиңиз!")

# Сырсөздөгү белгилердин санын эсептөө

print("Келгиле, тестСырСөз канча белги камтыйт экендигин карап көрөлү!")
print("Санадым:  ")
print(len(тестСырСөз))
```

Сандын функцияларынын мисалдары

```
# Сандардын тобун камтыган тизме түзүңүз

студентБаллы = [100, 90, 80, 70, 60, 50, 0]

# Көбүрөөк сандарды камтыган, кызмат акыны көрсөткөн башка тизме түзүңүз

жалпыАкы = [500, 600, 200, 400, 1000]

# Max() колдонуп, тизмесиндеги эң жогорку санды табыңыз

print("Тизмесиндеги эң жогорку балл кайсы?")
print:("Жообу: ")
print(max(студентБаллы))

# Min() колдонуп, тизмесиндеги эң төмөнкү санды табыңыз

print("Тизмесиндеги эң төмөнкү балл кайсы?")
print:("Жообу: ")
print(min(студентБаллы))

# Тизмени аныктабастан min() жана max() колдонуңуз

print(min(100, 90, 80, 70, 60, 50, 0))
print(max(100, 90, 80, 70, 60, 50, 0))

# Sum() колдонуп, тизмедеги сандардын суммасын эсептеңиз

print("Ушул жумада кызматкерлерге канча төлөдүк?")
print("Жалпы кызмат акы: ")
print(sum(жалпыАкы))
```

Бул эпизоддо

Бул эпизод абдан кызыктуу болду! Бул таң каларлык болду! Бул укмуштай болду! Бул жөргөмүш ... Жакшы эле. Айталы, ал укмуш жана аны ошол бойдон эле калтырып коёлу (Эй, тигил чоң комикстер бул сөздөрдү билбейт!).

Сиз бул бөлүмдө программалоо жаатында чоң секирик жасап, өзүңүздүн жеке модулдарыңызды жана функцияңызды түзүүнү үйрөндүңүз. Сиз бир нече орнотулган функцияларды үйрөнүп, Python программалоо тилинин эң күчтүү компоненттеринин бири болгон колдонуучу коому тарабынан түзүлгөн топтомдорду үйрөнүү мүмкүнчүлүгүнө ээ болдуңуз.

Биз көп нерсени камтыдык. Демек, адаттагыдай эле, ушул бөлүмдө биз сиздин супер күчтүү топтомуңузга кошкон айрым сонун нерселердин кыскача баяндамасын сунуштайбыз!

- Модулдардын үч түрү бар. Алар: орнотулган, топтомдор жана буйрутма менен жасалган.
- Орнотулган модулдар программага алдын-ала орнотулуп, пакеттерди үчүнчү жактын провайдерлери / Python коомчулугу түзүп, таңгактар атайын жасалган, өзүңүз жараткан нерселер.
- help() жана .__ doc__ модулдун документтерин же жардам файлын басып чыгарууга көмөктөшөт:

  Мисалы: help(time) жана print(time.__doc__)
- help(“modules”) сизге учурда Python сунуш кылган бардык жеткиликтүү модулдарды санап берет.
- Import модулу программаңызга импорттойт.

  Мисалы: import time
- Топтомду pip аркылуу буйрук сабына орното аласыз:
- python -m pip install <модулдун аты>
- def функцияны аныктоо үчүн колдонулат.

  Мисалы:

```
def firstFunction():
      print("Саламатсызбы!")
```

- str.upper() жана str.lower() сапты чоң жана кичине тамгага айландырат.
- str.isalpha, str.isnumeric жана str.isspace() туура маалыматтын түрү колдонулуп жататкандыгын текшеришет.

- len() саптагы белгилердин санын эсептейт.
- min() жана max() тизмедеги минималдуу жана максималдуу маанини табуучу
- sum() тизмеде камтылган маанилердин суммасын эсептейт.

8 – БӨЛҮМ

КЛАССТАРДЫ ЖАНА ОБЪЕКТИЛЕРДИ КОЛДОНУУ

Ушул учурда биз программалоо тилинин бир топ стандарттуу өзгөчөлүктөрүн жана тажрыйбаларын карап чыктык. Бул бөлүм да ушул салтты улантат. Бирок, бир караганда, бул теманы түшүнүү бир аз кыйыныраак болушу мүмкүн. Анткен менен кабатыр болбоңуз. Сиз ушул деңгээлге жеттиңиз. Биз сизди - мүдүрүлүп жүргөн жаш шакирттен толук кандуу, каарман баатырга айланганыңызды байкап турдук.

Ата-энеңиз Горгон планетасынан оозе элинин кол салуусунан аман калышса, алар сиз менен сыймыктанмак. Анда эмесе алга…

Бул бөлүмдө ООР - ОБП - же объектке багытталган программалоо деп аталган түшүнүккө көңүл бурулат. Сиз класс жана объект, конструктор, суперкласс, субкласс жана мурастоо деп аталган күчтүү куралдар жөнүндө үйрөнөсүз. Андан кийин биз жаңы, күчтүү түшүнүктөрдү жана ыкмаларды колдонуп, 6-бөлүмдө өзүбүз түзгөн программанын версиясын жасайбыз.

Эски Супер Баатыр генератору 3000ди өркүндөтө албайбыз деп ойлогонуңуз туура. Сиздин ишенген досуңуз акыркы үмүтүңүздү үзүп, эсиңизди оодарды! Мен сизди акыл-эсиңизди жыйды го деп ойлойм. Себеби сизди сооротуп отурганга убактым жок!

ОБП деген эмне?

Чындыгында, Python - объектке багытталган программалоо тили. Баары эле аны колдоно бербейт. Баары эле анын күйөрманы эмес. Баары эле ОБП ыкмасынын чыныгы күчүн түшүнүшпөйт. Айрымдар ОБП ыкмасында жазуу Python тилинин күчүн азайтат деп талашат. Башкача айтканда, объектке багытталган программалоонун негизин түзгөн методдорду, класстарды жана объектилерди колдонгондо Python тилин аз кызыгуу менен окушат жана колдонуучуларга ыңгайлуу эмес деп эсептешет.

Бул аргументте кандайдыр бир артыкчылыктар болушу мүмкүн. Бирок жалпысынан алганда программист окуй турган нерседен эмнени жоготот? Алар натыйжалуулуктун, каталарды азайтуунун жана ачык айтканда, программалоонун жакшы адаттарынын ордун толтура алышпай калышат. Мындан тышкары, эгер сиз жакшы код документин колдонуу тажрыйбасын колдосоңуз (биз бул тууралуу бир нече жолу талкууладык), анда кодуңуз *эң сонун* окула турган болот, анткени сиз программаңыздын ар бир бөлүмүндө өз ниетиңизди так айта аласыз.

Объектке багытталган программалоо (ОБП) - бул көп жолу колдонула турган кодду түзүү. Функциянын жана модулдардын артыкчылыктарын кантип талкуулагандыгыбыз эсиңиздеби? Ошол эле эрежелер ОБП практикасын колдонот.

Бул татаал же узак программаларга ылайыктуу, анткени сиз коддун үзүндүлөрүн колдонуп, бардыгын жагымдуу, тыгыз, оңой жеткиликтүү таңгакта сактай аласыз.

Ушул убакка чейин биз, *негизинен,* процедуралык программалоо деп аталган түшүнүккө таяндык. Процедуралык программалоо – бул, негизинен, ырааттуу тартипте пайда болгон жана колдонулган коддордун саптары. Бул бөлүмдө биз ошонун бардыгын өзгөртөбүз!

ОБП программалоонун негизги концепциясы башка көптөгөн программалоо тилдеринде *классстар* жана *объекттер* деп аталган түшүнүктөрдү камтыйт.

Кандай класстар бар (жана мен бааланамбы)?

Капа болбосоңуз, "класс" сөзү сизди коркутуп, математиканын кубанычтары же этруск элдеринин экономикасы жөнүндө кызыктуу окуяларды баяндаган узак лекцияларды эсиңизге салат деп ойлойм. Python сабактары алда канча кызыктуу. Бирок, чындыгында, анын пайдасын көргөндө кубанычыңыз чексиз болот.

Ишенбей турат окшойсуз?

Мен сизди алаксып кеткен сыяктанам.

Классты бир объект үчүн ДНК катары мүнөздөсө болот. Мындан да жакшысы - сиз аны бир нерсенин планы катары, алтургай, шаблону катары карай аласыз. Классты мындайча элестетип көрүңүз: эгерде сиз унаа жасай турган болсоңуз, анда жөн эле кокустан бир нече темир жана резина дөңгөлөктөрдү балка менен ургулап жасап алып, анан жакшы нерседен үмүт кылмак эмессиз. Эгер андай кылсаңыз, анда унааңыз алыска жетпейт же анчалык сонун көрүнбөйт!

Анын ордуна сиз унааңызда болушун каалаган айрым деталдарды же өзгөчөлүктөрдү камтыган план же класс түзөсүз. Андан ары биз, баатыр программисттер, натыйжалуулукка умтулгандыктан, биз каалаган унааны курганда колдоно турган план (класс) түзгүбүз келет. Ошентип, биз унаанын дагы бир моделин жасоону каалаганыбызда кайра-кайра план түзө бербешибиз керек болот.

Эгер биз, мисалы, унаа үчүн класс түзсөк, анда ар бир унаанын төрт дөңгөлөгү, алдыңкы айнеги, эшиги, мотору ж.б болушу керек. Булардын бардыгы ар бир унаада боло турган жалпы деталдары болмок. Түстөрү, боёктору, эшиктердин саны, дөңгөлөктөрдүн көлөмү жана башкалар ар кандай болушу мүмкүн. Бирок ошол негизги мүнөздөмөлөр ар бир автоунаада бар эле.

Демек, жалпылап айтканда, класс негизинен бирдей объектилерди түзүүгө мүмкүндүк берген план болуп саналат. Алардын бардыгы окшош негизги өзгөчөлүктөргө ээ. Жаңы объект түзгөн сайын ошол өзгөчөлүктөрдү коддоонун же аныктоонун ордуна, биз жөн гана биздин класстын мисалын чакырабыз. Баары даяр болуп, бардык жумуштар аткарылып бүтөт.

Эгер бул түшүнүк сиздин мээңизде толук орной элек болсо, анда кабатыр болбоңуз. Аны чыныгы коддо колдоно баштаганда, ал толугу менен айкын болот. Азырынча негизги түшүнүктү эсиңизден чыгарбаңыз:

Класстар - бул долбоорлор!

Объектилер деген эмне?

Эгерде класстар пландар болсо, анда объектилер - бул биз алардан жараткан нерселер! Программалоо көз карашынан алып караганда, объект түзгөндө биз класстын *мисалын* түзөбүз.

Программадагы нерселердин бир тобун көрсөтүү үчүн объектилерди колдонсо болот. Жогоруда айтылгандай, сиз, мисалы, унаа түзүү үчүн аларды колдоно аласыз. Же алар иттин тукумун же кызматкердин бир түрүн чагылдырышы мүмкүн.

Албетте, бул - супер баатыр программалоо китеби. Андыктан өзүбүздүн супер баатыр планыбызды түзгөндөн көрө, класстар жана объектилер түшүнүгүн кандайча жакшылап түшүндүрүүгө жана колдонууга болоруна назар салалы.

Биздин биринчи классты түзүү

Классты түзүү салыштырмалуу оңой. Бул, чындыгында, функцияны түзүүгө абдан окшош. Класс түзгөндө - функцияларга байланыштуу - бул *класс аныктамасы* деп аталат. Биз муну *class* ачкыч коду менен жасайбыз:

```
class супербаатыр():
     ...(кандайдыр бир код жазыңыз)
     ...(көбүрөөк код жазыңыз)
```

Бул мисалда супербаатыр аттуу классты кантип түзүү керектиги көрсөтүлгөн. Баштапкы сөздүн биринчи тамгасын чоң тамга менен жазуу үчүн класстарга ат коюу шартына көңүл буруңуз. Эгерде аталышта эки же андан көп сөз болсо, анда ар бир сөздүн биринчи тамгасын баш тамга менен басасыз. Мисалы, сиз “Америкалык супер баатыр” деген класс түзгүңүз келсе, анда сиз төмөндөгүнү колдонмоксуз:

```
class америкалыкСуперБаатыр ():
     ...(кандайдыр бир код жазыңыз)
     ...(көбүрөөк код жазыңыз)
```

Албетте, бул класстар техникалык жактан эч нерсе жасабайт. Алардын пайдалуу болушу жана өз функцияларын аткарышы үчүн биз аларга эмне кылыш керектигин же алардан түзө турган объектилеribизди аныктоого жардам берген кодду кошушубуз керек.

Функцияны класска кошкондо функция метод катары белгилүү болот. Методдор алар кирген класстын астында чегиниши керек.

```
class супербаатыр():
     def учуу(self):
          print("Мени карачы, мен ушинтип учам!")
```

Бул код "Мени карачы, мен ушинтип учам!" текстин чыгарган *учуу* ыкмасы менен супербаатыр аттуу классты жаратат.

Методду аныктоодо, аны def жардамы менен жасайбыз. Анын артынан методдун аты жазылат. Методдор кашаага алынган аргументтерди камтыйт. Ар бир класс ыкмасы жок дегенде self - өзүн-өзү аргументтештириши керек. Аларда дагы башка көптөгөн аргументтер камтылышы мүмкүн (бул тууралуу кийинчерээк!).

self сиз жараткан объектинин нускасына шилтеме берүү үчүн колдонулат. Биз, чындыгында, өз класстарыбызды да түзүп, аларды ишке киргизгенибизде бул дагы чоң мааниге ээ болот.

Ошондой эле, метод аныктамасынын астындагы код ал таандык болгон ыкмага карата чегиниши керектигин эске алыңыз.

Биз бир класстын ичинде каалаган сандагы методдорду жайгаштыра алабыз жана аларга ар кандай коддорду кошо алабыз. Анын ичинде өзгөрмө жана башка ушул сыяктуулар да болот.

Мисалы, биз супербаатыр классына эки ыкманы кошкубуз келсе, бири ага учканга мүмкүнчүлүк берет. Экинчиси ага көптөгөн хот-догдорду жегенге мүмкүнчүлүк берет. Биз өз классыбызды мындайча аныктай алабыз:

```
class супербаатыр():
     def учуу(self):
          print("Мени карачы, мен ушинтип учам!")
     def хотДог(self):
          print("Мен хот-догторду жактырам!")
```

Эгерде биз ушул кодду иштеткен болсок, анда эч нерсе болбойт. Анткени биз өзүбүздүн супербаатырдын классын гана аныктадык. Классты иш жүзүндө колдонуу үчүн класстын нускасын же объектисин түзүшүбүз керек.

Биздин биринчи объектибизди түзүү

Эми бизде Супер баатыр классы аркылуу түзүлгөн супер баатырдын негизги планы бар болгондуктан, биз биринчи каарманыбызды, тагыраак айтканда, биринчи каарман объектибизди жарата алабыз.

Объектти (же класстын мисалын) түзүү үчүн биз өзгөрмө түзүүгө жана ага маани бергенге окшош класстын көчүрмөсүн башташыбыз керек же жаратышыбыз керек.

```
хотДогКиши = супербаатыр()
```

Объект түзүү ушунчалык оңой. *хотДогКиши* объектиси эми биздин супербаатыр классынын бардык өзгөчөлүктөрүнө ээ. *супербаатыр* классынын

мисалы / объектиси *хотДогКишиде* сакталат. Анын ичинде биз классты түзүүдө аныкталган бардык атрибуттар жана ыкмалар бар.

Муну иш жүзүндө көрүү үчүн биз *супербаатыр* классында аныкталган эки ыкманы чакыра алабыз. Алар *хотДогКиши* объектисинин бөлүгү болуп саналат. Чакыруу сөзү кээ бир коддордун бир бөлүгүн аткаруу же иштетүү дегенди билдирет:

```
хотДогКиши.учуу()
хотДогКиши.хотДог()
```

Бул коддун биринчи сабы *хотДогКиши* объектисине кирип, *учуу* деп аталган ыкманы издеп, андан кийин аны тапкандан кийин аткарууну айтат. Экинчи сап дагы ошол эле нерсени жасайт. Ал *хотДог* ыкмасын издеп, коддун ошол бөлүгүн иштетет.

Ушул убакка чейин эмнени үйрөнгөнүбүздү жакшыраак түшүнүү үчүн SampleClassandObject.py аттуу жаңы файл түзүп, ага төмөнкү кодду кошолу (Эскертүү: бул код бир файлга чогултулган, ушул кезге чейин талкуулаган код):

```
class супербаатыр():
      def учуу(self):
            print("Мени карачы, мен ушинтип учам!")
      def хотДог(self):
            print("Мен хот-догторду жактырам!")
хотДогКиши = супербаатыр()
супербаатыр.учуу()
супербаатыр.хотДог()
```

Бул кодду иштеткенде, биз төмөнкү натыйжага ээ болобуз:

```
Мени карачы, мен ушинтип учам!
Мен хот-догторду жактырам!
```

Мунун бардыгы жакшы жана сонун. Бирок, чындыгында, бул мисалдар класстар менен объектилер сунуш кылган чыныгы күчтү жана объектке багытталган күчтү көрсөтө албайт. Эми класстардын жана объектилердин негизги түшүнүгүн түшүнгөндөн кийин аларды практикада жана чыныгы турмушта колдонолу.

Супер Баатыр генератору - 3000ди өркүндөтүү!

Эгер сиздин эсиңизде болсо, 6-бөлүмдө биз супер баатырды кокустан жараткан программа түзгөнбүз. Тактап айтканда, биз кокустан супер баатырдын ысымын, күчүн жана айрым статистикаларын жаратканбыз. Ал үчүн колдонуучу

программаны иштетип, натыйжаларын көрсөтүүдөн мурун бир нече жөнөкөй суроолорго жооп бериши керек.

Биз ал программаны абдан ырааттуу тартипте түздүк. Башкача айтканда, биз жазган кодду Python тили жана аны карап чыккан бардык программисттер саптан-сапка окушкан. Программа аткарылып жатканда (кемчиликсиз, мен дагы кошумчалайм!) эгер биз экинчи, же миңинчи супер баатырды жараткыбыз келсе, эмне болот? Ал үчүн программанын учурдагы абалында, колдонуучу программаны кайра-кайра иштетүүгө аргасыз болот.

Эгерде колдонуучу бир нече баатырды сураса, биз ар дайым супер баатырды тандоо процессин улантуу үчүн цикл түзүп алмакпыз же биз көбүрөөк супер баатырларга мүмкүнчүлүк берүү үчүн кодду дагы кошо берсек болмок. Бирок, дагы бир жолу биз программаларыбыздын иштешин жакшыртуу үчүн жана ката кетирүү мүмкүнчүлүгүн азайтуу үчүн мүмкүн болушунча аз саптардан турган код түзүүнү каалайбыз.

Бул жөнүндө ойлонуп көрсөңүз: биздин эски Супер Баатыр генератору - 3000 программабыз ар бир супер баатырды кол менен жараткан система болгон. Эгер анын ордуна класстарды жана объектилерди колдонгон болсок, бизде миңдеген адамдар супер баатырларды басып чыгарган жана адамдардын катасынан коркпой турган жогорку технологиялуу фабрика болмок. Андан тышкары, бул убакытты да үнөмдөмөк. Анткени биз мынчалык көп код жазып отурмак эмспиз.

Ушунун бардыгын эске алуу менен, бул жолу класстарды жана объектилерди колдонуп, Супер Баатыр генератору - 3000ди кайра жараталы.

Эсиңиздерде болсо, биздин программанын түп нускасында ар бир баатырдын физикалык жана психикалык өзгөчөлүктөрүн аныктаган статистикалык маалыматтар топтому болгон. Аларга төмөнкүлөр кирген:

- *Зээндүүлүк*: Баатыр канчалык зээндүү;
- *Тапкычтык*: Баатыр кандай тапкыч;
- *Чыдамдуулук*: Каармандын канчалык энергиясы бар, чыдамдуу;
- *Акылмандык*: Алар канчалык акылдуу жана канчалык чыныгы турмуштук тажрыйбасы бар;
- *Дисциплина*: алардын денеси жаракаттан канчалык айыгып, ооруларга туруштук бере алат, дисциплиналуу;
- *Эптүүлүк*: Каарманыбыз канчалык акробат жана шамдагай;
- *Ылдамдык*: Баатыр канчалык тез кыймылдайт.

Биз бул атрибуттардын ар бирин *супербаатыр* классына ыйгара алабыз. Ошентип, бул класстын объектисин түзгөндө биз жасаган бардык баатырлар бирдей статистикага ээ болот. Анткени биз ар бир баатырдын жок дегенде кандайдыр бир акылы, күчү, ыкчамдыгы жана башкалары болушу керектигин билебиз. Мунун баары - кадимки баатырдын жалпы мүнөздөмөлөрү. Демек, биздин долбоордун же баатырдын шаблонунун бир бөлүгү болот.

Келгиле, SuperHeroClass.py аттуу жаңы файл түзүп, ага төмөнкү кодду кошолу:

```
# random модулду импорттоп алыңыз, анткени биз сандарды туш келди алабыз.
import random
# class супербаатыр() классын түзгөн бардык баатырлар үчүн шаблондун ролун аткарган супербаатыр классын түзүңүз:
class супербаатыр():
  # Биздин классты баштоо жана анын атрибуттарын орнотуу:

     def __init__(self):
          self.суперАталыш = " "
          self.күч = " "
          self.тапкычтык = тапкычтык
          self.зээндүүлүк = зээндүүлүк
          self.чыдамдуулук = чыдамдуулук
          self.акылмандык = акылмандык
          self.дисциплина = дисциплина
          self.эптүүлүк = эптүүлүк
          self.ылдамдык = ылдамдык

# Random () модулунун жардамы менен ар бир статистикага кокустук маанилерди кошуу:
тапкычтык = random.randint (1,20)
зээндүүлүк = random.randint(1,20)
чыдамдуулук = random.randint(1,20)
акылмандык = random.randint(1,20)
дисциплина = random.randint(1,20)
эптүүлүк = random.randint(1,20)
ылдамдык = random.randint(1,20)
```

Бул коддо биз конструктор методу деп аталган жаңы ыкма менен тааныштык. Биз аны класска таандык болгон жаңы маалыматтарды инициализациялоо үчүн колдонобуз. Конструктор методу дагы __init__ методу деп аталып, ар кандай өзгөрмөлөргө алдын-ала маалыматтарды кошуу керек болгондо, биз класста жараткан биринчи ыкма болуп саналат.

Биз параметрлерибизди __init__ ыкмасынын кашаанын ичине киргизип, андан кийин self шилтемени ар бир параметрге барабар кылдык. Мисалы:

self.*зээндүүлүк* = *зээндүүлүк* деп орнотот

Ошентип, кийинчерээк программабызда объектилерибизди түзгөндө, ар кандай параметрлерге кайрылсак болот. Бул учурда биздин каармандын статистикасы билинет. Биз аларды программабызда колдонсок болот.

Андан кийин бул учурда биз баатыр шаблондорун жаратууну каалайбыз жана ар бир баатыр статусунун кокустан болушун каалайбыз. Ошондуктан ар бир параметрде random() модулун колдонуп, биздин баатырдын статистикасын көрсөтөбүз. Мисалы:

```
тапкычтык = random.randint(1,20)
```

1ден 20га чейинки аралыктагы туш келди маанини кошот.

Класстарды, объектилерди жана методдорду түшүнүү жаңы программистерге кыйынга турушу мүмкүн. Ошондуктан чыдамдуу болуп, код менен жүрүңүз, нерселер 100% мааниге ээ болбосо дагы кээде анын эмне экендигин толук түшүнүү үчүн жасоого ниеттенген кодду иш жүзүндө көрүшүңүз керек экенин дагы бир жолу кайталаймын.

Эми биз алгачкы *супербаатыр* классын түзүп, биз жараткан супер баатырлардын шаблону кандай болорун чечкенден кийин, келгиле, класстын мисалын түзүүгө аракет кылалы (кайталаймын, ал объект жаратуу деп да аталат). Андан кийин биз каармаңыбыздын статистикасын басып чыгарабыз. SuperheroClass.py файлыңызга төмөнкү кодду кошуңуз:

```
# Биз кокустан сандарды иштеп чыгышыбыз үчүн random модулун импорттоп алыңыз.

import random

# Биз жараткан бардык баатырлар үчүн шаблондун ролун аткарган супербаатыр классын түзүңүз.

class супербаатыр():

# Биздин классты баштоо жана анын атрибуттарын орнотуу:
    def __init__(self):
            self.суперАталыш = суперАталыш
            self.күч = куч
            self.тапкычтык = тапкычтык
            self.зээндүүлүк = зээндүүлүк
            self.чыдамдуулук = чыдамдуулук
            self.акылмандык = акылмандык
            self.дисциплина = дисциплина
            self.эптүүлүк = эптүүлүк
            self.ылдамдык = ылдамдык
```

```
# Random() функциясын колдонуп, ар бир статуска кокустук маанилерди кошуу:
тапкычтык = random.randint(1,20)
зээндүүлүк = random.randint(1,20)
чыдамдуулук = random.randint(1,20)
акылмандык = random.randint(1,20)
дисциплина = random.randint(1,20)
эптүүлүк = random.randint(1,20)
ылдамдык = random.randint(1,20)
print("супер баатырыңыздын аты-жөнүн киргизиңиз: ")
# супербаатыр объектисин түзүү
баатыр = супербаатыр()
# Колдонуучунун киргизүүсүн колдонуп, суперАталыш  маани берүү
баатыр.суперАталыш = input('>')
# Биз түзүлгөн объектинин натыйжаларын, анын параметрлерин басып чыгарабыз:
print("Баатырдын аты %s." % (баатыр.суперАталыш))
print("Статистикасы:")
print("")
print("Зээндүүлүгү: ", баатыр.зээндүүлүк)
print("Тапкычтыгы: ", баатыр.тапкычтык)
print("Чыдамдуулугу: ", баатыр.чыдамдуулук)
print("Акылмандуулугу: ", баатыр.акылмандык)
print("Дисциплинасы: ", баатыр.дисциплина)
print("Эптүүлүгү: ", баатыр.эптүүлүк)
print("Ылдамдыгы ", баатыр.ылдамдык)
print("")
```

Биздин программанын ушул нускасында колдонуучудан баштапкы Супер Баатыр генератору - 3000 программабыздагыдай, кокустан жаралган атын эмес, өзүнүн супер баатырынын атын киргизүүсүн суранабыз. Капа болбоңуз, жакын арада бул маанини кокустан чыга тур гандай жасайбыз. Азырынча биз колдонуучунун input() функциясын колдонуп, өзүнүн атын киргизишине мүмкүнчүлүк берип, жөнөкөй нерселерди жасап жатабыз. Input() функциясынын мааниси *суперАталыш* объектинин параметринде жайгаштырылат. Мунун баары сапта жетишилет:

```
баатыр.суперАталыш = input('>')
```

Бул жерде input() функциясын мурункуга караганда бир аз башкача пайдалангандыгыбызды байкагандырсыз. Кашаанын ичиндеги '>' колдонуучунун экранына жөн гана > билдирүүсүн жайгаштырат, ошондо ал кайда терүү керектигин билет.

Андан кийин, мисалы, төмөнкү сапты колдонуп, баатыр объектинин ар бир параметрине туш келди түзүлгөн маанилерди басып чыгардык:

```
print("Зээндүүлүгү: ", баатыр.зээндүүлүк)
```

Андан соң, ошол саптын *баатыр.зээндүүлүк* Python баатыр объекттеринин мээ параметринде сакталган маанини басып чыгарууну айтат - бул өзгөрмөгө окшош иштейт.

Эгер сиз ошол программаны иштетсеңиз, анда сиз мындай натыйжага ээ болосуз - сиздин баалуулуктарыңыз ар башка болот, анткени алар randomly-кокустан пайда болот:

```
Супер баатырыңыздын аты-жөнүн киргизиңиз:
>СуперАсан
Баатырдын аты СуперАсан
Статистикасы:
Зээндүүлүгү: 10
Тапкычтыгы: 17
Чыдамдуулугу: 5
Акылмандуулугу: 1
Дисциплинасы: 19
Эптүүлүгү: 16
Ылдамдыгы: 10
```

Буга чейин баары мыкты болду! Эми баатырдын ысымын жана супер күчүн кокустан жаратуу үчүн код кошолу. Бул бөлүк үчүн биз саптарды кошобуз:

```
# Мүмкүн болгон супер күчтөрдүн тизмесин түзүү:
 суперКүчтөр = ['Учуучу', 'Укмуш Күчтүү', 'Телепатия', 'Супер Ылдам', 'Көп Хот-Догдорду Жей Алат', 'Аркандан Секиргени Жакшы']
# Супер күчтөр тизмесинен супер күчтү кокустан тандап алуу
# жана аны өзгөрүлмө кубаттуулукка ыйгаруу:
күч = random.choice(суперКүчтөр)
# Мүмкүн болгон ысымдардын жана фамилиялардын тизмесин түзүү:
```

```
суперЫсымдар = ['Керемет','Уатта','Кутурган','Укмуш', 'Таң Калычтуу',
'Татыктуу', 'Даңазалуу', 'Ортодон-Жогору', 'Ал Жигит', 'Өзгөчө']

суперФамилиялар = ['Бала', 'Киши', 'Динго', 'Бифкейк', 'Кыз', 'Аял',
'Жигит','Баатыр', 'Макс', 'Кыял', 'Мачо','Аргымак']
# Супер баатырдын атын кокустан тандайбыз.
# Муну биз эки ысым тизмебиздин ар биринен бирден ат тандоо менен
жасайбыз.
# Жана суперАталыш өзгөрмөсүнө кошобуз.

суперАталыш = random.choice(суперЫсымдар)+ " " +random.choice(суперФамилиялар)
```

Төмөндө биз супер баатыр статистикасын аныктадык. Азыр супер баатырдын ысымы кокустан жаралып жаткандыктан, колдонуучудан алардын оюн сурап кереги жок. Андыктан саптарды алып салабыз:

```
print("Сураныч, супер баатырдын ысымын киргизиңиз: ")
```

ошондой эле:

```
# Колдонуучунун киргизүүсүн колдонуп, суперАталышка маани беребиз:
баатыр.суперАталыш = input('>')
```

Бизге эми алардын кереги жок, анткени биз *суперАталыш* программабыздын түп нускасында болгондой эле, *суперЫсымдар* жана *суперФамилиялар* тизмелеринен кокустан жаратабыз.

Эми кодуңуз төмөнкүлөргө дал келиши керек. Эгер туура келбесе, анда ушул бөлүмдү дагы бир жолу карап, кодуңузду меники менен дал келгидей кылып өзгөртүңүз:

```
# Биз кокустан сандарды иштеп чыгышыбыз үчүн, random модулун импорттоп
#алыңыз
import random
# Биз жараткан бардык баатырлар үчүн шаблондун ролун аткарган супербаатыр
#классын түзүңүз
class супербаатыр():
    # Биздин классты баштоо жана анын атрибуттарын орнотуу
    def __init__(self):
        self.суперАталыш = суперАталыш
        self.күч = күч
        self.тапкычтык = тапкычтык
        self.зээндүүлүк = зээндүүлүк
        self.чыдамдуулук = чыдамдуулук
```

```
            self.акылмандык = акылмандык
            self.дисциплина = дисциплина
            self.эптүүлүк = эптүүлүк
            self.ылдамдык = ылдамдык
# random() функциясын колдонуп, ар бир статуска кокустук маанилерди кошуу
тапкычтык = random.randint(1,20)
зээндүүлүк = random.randint(1,20)
чыдамдуулук = random.randint(1,20)
акылмандык = random.randint(1,20)
дисциплина = random.randint(1,20)
эптүүлүк = random.randint(1,20)
ылдамдык = random.randint(1,20)

# Мүмкүн болгон супер күчтөрдүн тизмесин түзүү:

суперКүчтөр = ['Учуучу', 'Укмуш Күчтүү', 'Телепатия', 'Супер Ылдам',
'Көп Хот-Догдорду Жей Алат', 'Аркандан Секиргени Жакшы']

# Супер күчтөрдүн тизмесинен супер күчтү кокустан тандап алуу
# жана аны өзгөрүлмө кубаттуулукка ыйгаруу

күч = random.choice(суперКүчтөр)

# Мүмкүн болгон ысымдардын жана фамилиялардын тизмесин түзүү:

суперЫсымдар = ['Керемет','Уатта','Кутурган','Укмуш', 'Таң Калычтуу',
'Татыктуу', 'Даңазалуу', 'Ортодон-Жогору', 'Ал Жигит', 'Өзгөчө']

суперФамилиялар = ['Бала', 'Киши', 'Динго', 'Бифкейк', 'Кыз', 'Аял',
'Жигит','Баатыр', 'Макс', 'Кыял', 'Мачо','Аргымак']

# Кокустан супер баатырдын атын тандоо

# Муну биз эки ысым тизмебиздин ар биринен бирден ат тандоо менен
жасайбыз
# Жана суперАталыш өзгөрмөсүнө кошуңуз

суперАталыш = random.choice(суперЫсымдар)+ " "
+random.choice(суперФамилиялар)

print("Сураныч, супер баатырдын ысымын киргизиңиз: ")

# супербаатыр объектисин түзүү
баатыр = супербаатыр()
```

```
# Колдонуучунун киргизүүсүн колдонуп, суперАталышка маани берүү
# баатыр.суперАталыш = input ('>')

# Биз түзүлгөн объекттин натыйжаларын, анын параметрлерин басып
чыгарабыз:

print("Баатырдын аты %s." % (баатыр.суперАталыш))
print("Баатырдын күчү: ", күч)
print("Баатырдын статистикасы:")
print("")
print("Зээндүүлүгү: ", баатыр.зээндүүлүк)
print("Тапкычтыгы: ", баатыр.тапкычтык)
print("Чыдамдуулугу: ", баатыр.чыдамдуулук)
print("Акылмандуулугу: ", баатыр.акылмандык)
print("Дисциплинасы: ", баатыр.дисциплина)
print("Эптүүлүгү: ", баатыр.эптүүлүк)
print("Ылдамдыгы ", баатыр.ылдамдык)
print("")
```

Эгер сиз азыр ушул программаны иштетсеңиз, анда төмөнкүдөй натыйжага ээ болосуз (кайрадан, алардын мааниси башкача болот, анткени алар кокустан түзүлгөн):

```
Сураныч, супер баатырдын ысымын киргизиңиз:
>Асан
Баатырдын аты Асан.
Баатырдын күчү:  Көп Хот-Догдорду Жей Алат
Баатырдын статистикасы:

Зээндүүлүгү:  19
Тапкычтыгы:  13
Чыдамдуулугу:  13
Акылмандуулугу:  2
Дисциплинасы:  18
Эптүүлүгү:  4
Ылдамдыгы  20
```

Ошентип, учурда, биздин программа Супер Баатыр генератору - 3000ин баштапкы версиясы менен дээрлик бирдей иштейт. Болгону коддордун саптары азыраак жана ката кетүү мүмкүнчүлүгү аз. Айрым кошумчалар дагы бар. Мисалы, биз али колдонуучудан баатырды жаратууну каалайсызбы деп сурай элекпиз. Ошондой эле, маанилер пайда болуп жатканда пауза эффекттерибизди киргизген жокпуз.

Бирок, толтурула элек нерселерибиз өз ордунда. Буга кийинки бөлүмдө эски коңгуроолор менен ышкырыктардын айрымдарын, ошондой эле класстардын жана объектилердин *чыныгы* күчүн көрсөткөн жаңы, чындыгында, сонун функцияларды кошобуз!

Мурас, субкласстар жана башкалар!

Класстардын эң сонун касиеттеринин бири - буларды колдонуп, башка класстарды түзүп, inheritance - мурас деп аталган түшүнүк менен көптөгөн узун коддорду колдонбостон, алардын атрибуттарын жаңы түзүлгөн класска өткөрүп берсеңиз болот. Бул сиздин ата-энеңиздин генетикалык кодун сизге кантип өткөрүп бергенине окшош болот. Бир гана Python классы так эмнени мураска ала тургандыгын айта алабыз.

Биз башка класстын негизинде класс түзгөндө, бул жаңы түзүлгөн классты **subclass**- субкласс деп атайбыз. Демейки шартта бул субкласстар өздөрү түзүлгөн класстын методдорун жана параметрлерин мурастап алышат. Алар, айтмакчы, башкы класстар же суперкласстар деп аталат.

Бардык нерселер сыяктуу эле, кээде бул идея иш жүзүндө кандай программа аркылуу иштээрин көрсөткөн жакшы.

Азырынча Супер Баатыр генератору 3000 бизге эски супер баатырларды жаратууга мүмкүнчүлүк берет. Бирок, өзүңүз билесиз, бардык эле баатырлар бирдей жарала бербейт. Мисалы, Супермен башка бир планетадан келген келгин. Ал кийим аркылуу көрө алат жана күчтүү болуу үчүн күндүн нурун (буфф өсүмдүгү сыяктуу) жейт. Ошол эле учурда Бэтменде эч кандай күч жок же тескерисинче, анын супер күч-кубаты толгон-токой акча, укмуштай унаа, жана компьютердик программалоо көндүмдөрүнө ээ болгон батлер. Чынын айтайынбы? Мында менин ата-энем смс жазганды билбей жатса, ал эми Альфред дүйнөдөгү эң күчтүү супер компьютерди колдонуп жатат.

Мен бир аз алаксып кеттим.

Программабызды бир аз реалдуу кылуу үчүн биз супер баатырларга жаңы касиет киргизебиз: супер баатыр түрлөрү. Ар бир жараткан түрүбүз үчүн аларга кандайдыр бир бонусту беребиз. Азырынча, биздин баатырлардын жаңы "түрлөрүн" чагылдырган эки субклассты түзүүгө токтололу. Бири робот болгон супер баатырлар үчүн, экинчиси мутацияланган супер баатырлар үчүн болот.

Бул коддо мындайча көрүнөт:

```
# Мутант аттуу суперкаарман
```

```
# Мутацияланган баатырлар ылдамдыгына +10 бонус алышат.
class Мутант(супербаатыр):
    def __init__(self):
        супербаатыр.__init__(self)
        print("Сиз Мутант жараттыңыз!")
        self.ылдамдык = self.ылдамдык + 10

# Робот аттуу Супер Баатырдын субклассын түзүү
# Робот каармандары тапкычтыгына + 10 бонус алышат.
class Робот(супербаатыр):
     def __init__(self):
          супербаатыр.__init__(self)
          print("Сиз Мутант жараттыңыз!")
          self.тапкычтык = self.тапкычтык + 10
```

Бул жерде биз эки жаңы класс түздүк. Экөө тең биздин *супербаатыр* классынын субкласстары. Буга жетишүү жолу - жаңы түзүлгөн класстын кашаасына башкы классынын атын киргизүү. Мисалы: Мутант(супербаатыр) классы Python котормочусуна супербаатырдын баласы же субклассы болгон классты түзүп, анын ыкмаларын жана параметрлерин мураска алууну айтат.

Андан кийин def __init __ (self) аркылуу жаңы субклассыбызга баштапкы маанини беребиз жана супербаатыр.__ init __ (self) аркылуу супербаатырлар классына кайрадан баштапкы маанини беребиз. Анткени техникалык жактан класска жана субкласска негизделген жаңы объектилерди жаратабыз.

Акыры, биз баатырларыбызга кайсы түрдөгү баатыр экендигине жараша бонус бергибиз келет. Мутация болгон каарман коддун ушул сабында көрсөтүлгөндөй ылдамдыгы үчүн бонусту алат:

```
self.ылдамдык = self.ылдамдык + 10
```

Ал эми роботтор тапкычтык үчүн коддун ушул сабы аркылуу бонусту алышат:

```
self.тапкычтык = self.тапкычтык + 10
```

Баатырлардын башка бардык мүнөздөмөлөрү алар алгач супербаатырлар классында кандай болсо, ошол бойдон калат. Эгер биз алардын маанидерин дагы бир жолу өзгөрткүбүз келсе, анда жаңы түзүлгөн субкласстарда аны так кылышыбыз керек эле.

Эми биз эки жаңы классты түзгөндөн кийин аларды иш жүзүндө көрүү үчүн алардын негизинде мисал / объект түзүшүбүз керек. Ички класстан объект түзүү коду, каалаган класстан объект жаратуу менен бирде. Эгер биз жаңы мутация каармanын жана жаңы робот каармanын жаратууну кааласак, анда мындай коддордун саптарын колдонуп жасамакпыз:

```
баатыр2 = Робот()
баатыр3 = Мутант()
```

Келгиле, кадимки супербаатырдын, роботтун жана мутанттын статистикасын чыгаруу үчүн бир нече код түзөлү:

```
# супербаатыр объектисин түзүү
баатыр = супербаатыр()

# Биз түзүлгөн объекттин натыйжаларын, анын параметрлерин басып
#чыгарабыз
print("Атыңыз %s." % (баатыр.супербаатыр))
print("Сиздин супер күчүңүз: ", баатыр.күч)
print("Сиздин жаңы статистикаларыңыз:")
print("")
print("Зээндүүлүк: ", баатыр.зээндүүлүк)
print("Тапкычтык: ", баатыр.тапкычтык)
print("Чыдамдуулук: ", баатыр.чыдамдуулук)
print("Акылмандык: ", баатыр.акылмандык)
print("Дисциплина: ", баатыр.дисциплина)
print("Эптүүлүк: ", баатыр.эптүүлүк)
print("Ылдамдык: ", баатыр.ылдамдык)
print("")

# Мутант объектисин түзүү

баатыр2 = Мутант()
print("Мутанттын аты %s." % (баатыр2.суперАталыш))
print("Супер күчү: ", баатыр2.күч)
print("Жаңы статистикасы:")
print("")
print("Зээндүүлүгү: ", баатыр2.зээндүүлүк)
print("Тапкычтыгы: ", баатыр2.тапкычтык)
print("Чыдамдуулугу: ", баатыр2.чыдамдуулук)
print("Акылмандуулугу: ", баатыр2.акылмандык)
print("Дисциплинасы: ", баатыр2.дисциплина)
print("Эптүүлүгү: ", баатыр2.эптүүлүк)
print("Ылдамдыгы ", баатыр2.ылдамдык)
```

```
print("")

# Робот каарманын түзүү

баатыр3 = Робот()
print("Роботтун аты %s." % (баатыр3.суперАталыш))
print("Супер күчү: ", Робот3.күч)
print("Жаңы статистикасы:")
print("")
print("Зээндүүлүгү: ", баатыр3.зээндүүлүк)
print("Тапкычтыгы: ", баатыр3.тапкычтык)
print("Чыдамдуулугу: ", баатыр3.чыдамдуулук)
print("Акылмандуулугу: ", баатыр3.акылмандык)
print("Дисциплинасы: ", баатыр3.дисциплина)
print("Эптүүлүгү: ", баатыр3.эптүүлүк)
print("Ылдамдыгы: ", баатыр3.ылдамдык)
print("")
```

Эгерде сиз ушул жаңы коддун бардыгын файлыңызга кошуп (бир аздан кийин кошобуз) иштетип койсоңуз, анда натыйжаларыңыз төмөнкү сыяктуу болмок:

```
Баатырдын аты Ортодон-Жогору Бала.
Баатырдын күчү:  Көп Хот-Догдорду Жей Алат
Баатырдын статистикасы:
Зээндүүлүгү:  16
Тапкычтыгы:  4
Чыдамдуулугу:  4
Акылмандуулугу:  18
Дисциплинасы:  16
Эптүүлүгү:  12
Ылдамдыгы  2

Сиз Мутант жараттыңыз!
Мутанттын аты Ортодон-Жогору Бала.
Баатырдын күчү:  Учуучу
Баатырдын статистикасы:
Зээндүүлүгү:  16
Тапкычтыгы:  4
Чыдамдуулугу:  4
Акылмандуулугу:  18
Дисциплинасы:  16
Эптүүлүгү:  12
Ылдамдыгы  12

Сиз Робот жараттыңыз!
Аты Ортодон-Жогору Бала.
Күчү:  Көп Хот-Догдорду Жей Алат
Статистикасы:
Зээндүүлүгү:  15
Тапкычтыгы:  14
Чыдамдуулугу:  4
Акылмандуулугу:  18
Дисциплинасы:  16
Эптүүлүгү:  12
Ылдамдыгы  2
```

Робот сыяктуу эле кадимки супер баатырдын ылдамдыгы 2ге барабар экенине көңүл буруңуз. Бирок мутанттын ылдамдыгы 12ге жетет. Ошо сыяктуу эле, биздин кадимки баатырдын да, мутанттын да эптүүлүгү 4кө жетет. Ал эми биздин роботтун тапкычтыгы, биз белгиленгендей, 14кө жетет.

Эгер сиз жаңы кодду кошсоңуз, анда SuperheroClass.py файлыңыз ушул сыяктуу болушу керек. Эгер андай болбосо, сураныч, аны окшоштуруу үчүн убакыт бөлүңүз:

```
# Биз кокустан сандарды иштеп чыгышыбыз үчүн, random модулун импорттоп алыңыз

import random

# Биз жараткан бардык баатырлар үчүн шаблондун ролун аткарган Супер Баатырлар классын түзүңүз

class супербаатыр():

# Биздин классты баштоо жана анын атрибуттарын орнотуу

     def __init__(self):
          self.суперАталыш = суперАталыш
          self.күч = күч
          self.тапкычтык = тапкычтык
          self.зээндүүлүк = зээндүүлүк
          self.чыдамдуулук = чыдамдуулук
          self.акылмандык = акылмандык
          self.дисциплина = дисциплина
          self.эптүүлүк = эптүүлүк
          self.ылдамдык = ылдамдык

# random() функциясын колдонуп, ар бир статуска кокустук маанилерди кошуу

тапкычтык = random.randint(1,20)

зээндүүлүк = random.randint(1,20)

чыдамдуулук = random.randint(1,20)

акылмандык = random.randint(1,20)

дисциплина = random.randint(1,20)

эптүүлүк = random.randint(1,20)

ылдамдык = random.randint(1,20)

# Мүмкүн болгон супер күчтөрдүн тизмесин түзүү

суперКүчтөр = ['Учуучу', 'Укмуш Күчтүү', 'Телепатия', 'Супер Ылдам', 'Көп Хот-Догдорду Жей Алат', 'Аркандан Секиргени Жакшы']

# Супер күчтөрдүн тизмесинен супер күчтү кокустан тандап алуу
```

```
# жана аны өзгөрмө күчкө ыйгаруу

күч = random.choice(суперКүчтөр)

# Мүмкүн болгон ысымдардын жана фамилиялардын тизмесин түзүү

суперЫсымдар  = ['Керемет','Уатта','Кутурган','Укмуш', 'Таң Калычтуу',

'Татыктуу', 'Даңазалуу', 'Ортодон-Жогору', 'Ал Жигит', 'Өзгөчө']

суперФамилиялар = ['Бала', 'Киши', 'Динго', 'Бифкейк', 'Кыз', 'Аял',
'Жигит',
'Баатыр', 'Макс', 'Кыял', 'Мачо','Аргымак']

# Кокустан супер баатырдын атын тандоо
# Муну биз эки ысым тизмебиздин ар биринен бирден ат тандоо менен жасайбыз
# Жана суперАталыш өзгөрмөсүнө кошуңуз

суперАталыш = random.choice(суперЫсымдар)+ " " +random.choice(суперФамилиялар)

# Мутант аттуу Супер Баатырдын субклассын түзүү
# Мутант каармандары ылдамдыгы үчүн +10 бонус алышат.

class Мутант(супербаатыр):
     def __init__(self):
          супербаатыр.__init__(self)
          print("Сиз Мутант жараттыңыз!")
          self.ылдамдык = self.ылдамдык + 10

# Робот аттуу супер баатырдын субклассын түзүү
# Робот каармандары тапкычтык упайына + 10 бонус альшат.

class Робот(супербаатыр):
     def __init__(self):
          супербаатыр.__init__(self)
          print("Сиз Робот жараттыңыз!")
          self.тапкычтык = self.тапкычтык + 10

# супербаатыр объектисин түзүү

баатыр = супербаатыр ()

# Биз түзүлгөн объекттин натыйжаларын, анын параметрлерин басып чыгарабыз

print("Баатырдын аты %s." % (баатыр.суперАталыш))
print("Анын күчү: ", баатыр.күч)
```

```
print("Жаңы статистикасы:")
print("")
print("Зээндүүлүгү: ", баатыр.зээндүүлүк)
print("Тапкычтыгы: ", баатыр.тапкычтык)
print("Чыдамдуулугу: ", баатыр.чыдамдуулук)
print("Акылмандуулугу: ", баатыр.акылмандык)
print("Дисциплинасы: ", баатыр.дисциплина)
print("Эптүүлүгү: ", баатыр.эптүүлүк)
print("Ылдамдыгы ", баатыр.ылдамдык)
print("")

# Мутант объектисин түзүү

баатыр2 = Мутант()
print("Мутанттын аты %s." % (баатыр2.суперАталыш))
print("Анын күчү: ", баатыр2.күч)
print("Жаңы статистикасы:")
print("")
print("Зээндүүлүгү: ", баатыр2.зээндүүлүк)
print("Тапкычтыгы: ", баатыр2.тапкычтык)
print("Чыдамдуулугу: ", баатыр2.чыдамдуулук)
print("Акылмандуулугу: ", баатыр2.акылмандык)
print("Дисциплинасы: ", баатыр2.дисциплина)
print("Эптүүлүгү: ", баатыр2.эптүүлүк)
print("Ылдамдыгы: ", баатыр2. ылдамдык)
print("")

# Робот  каарманын түзүү
баатыр3 = Робот()
print("Роботтун аты %s." % (баатыр3. суперАталыш))
print("Анын күчү: ", баатыр3.күч)
print("Жаңы статистикасы:")
print("")
print("Зээндүүлүгү: ", баатыр3.зээндүүлүк)
print("Тапкычтыгы: ", баатыр3.тапкычтык)
print("Чыдамдуулугу: ", баатыр3.чыдамдуулук)
print("Акылмандуулугу: ", баатыр3.акылмандык)
```

```
print("Дисциплинасы: ", баатыр3.дисциплина)
print("Эптүүлүгү: ", баатыр3.эптүүлүк)
print("Ылдамдыгы ", баатыр3.ылдамдык)
print("")
```

Коңгуроолор менен ышкырыктарды кошуу

Эми биз аткарышыбыз керек болгон акыркы нерсе - биздин программага бир нече коңгуроо жана ышкырыктарды, жагымдуу кошумча өзгөчөлүктөрдү же жасалгаларды кошуу. Эсиңизде болсун, бул бөлүмдүн максаты биздин Супер Баатыр генератору 3000 программабызды ошол принциптер менен кайра түзүү үчүн объектке багытталган программалоону колдонууну үйрөнүү болгон. Биздин түп нускабызда бир аз укмуштуу паузалар болуп, колдонуучуга бир нече суроолорду бергенбиз. Мына, биз ушул функциялардын бардыгын кайрадан программабызга кошуп, аларга баатырдын түрүн тандоо мүмкүнчүлүгүн беребиз.

Ушул кезге чейин ушул китептен үйрөнгөн принциптерибизди, анын ичинде if-elif-else билдирүүлөрүн, random(), input(), and time() модулдарын жана, албетте, ушул бөлүмдөн алынган ООР - ОБП принциптерин колдонобуз.

Көнүгүү катары сизди коддун ар бир кадамы боюнча кайталоонун ордуна, мен азыр кошо турган негизги код өзгөчөлүктөрүнүн айрымдарын бөлүп көрсөтөм. Андан кийин өзүңүзгө карап чыгуу жана коддоо үчүн бардык программаны киргизем.

Баштапкы колдонуучуларга баштапкы программабыздагыдай эле, тандоо мүмкүнчүлүгүн берүүнү каалайбыз. Негизинен, “алар Супер Баатыр генератору – 3000ди колдонууну каалайбы?” деп сурайбыз. Эгер алар "О" тандап алышса, программа улантылат. Эгер жок болсо, цикл андан ары улантууну каалайбы деп сурай берет:

```
# Киришүү тексти

print("Супер Баатыр генератору 3000 менен супер баатыр жаратууга даярсызбы?")

# Колдонуучуга суроо берип, аларды жооп сураңыз
# input() клавиатурадан киргизилген маанини 'угат'

# Андан кийин колдонуучулардын жоопторун чоң тамгаларга өзгөртүү үчүн upper() колдонобуз

print("О же Ж жазыңыз:")
жооп = input()
жооп = жооп.upper()

# “О” деген жоопту текшерүү үчүн while цикли жазылат
# Жообубуз “О” БОЛГОНГО чейин улана берет
```

```
# Колдонуучу “О” тергенде гана циклдан чыгып, программа уланат
while жооп != "О":
    print("Кечиресиз, бирок улантуу үчүн О жазышыңыз керек!")
    print("О же Ж жазыңыз:")
    жооп = input()
    жооп = (жооп.upper())
print("Сонун, баштайлы!")
```

Бул биздин программанын түп нускасынан алынган код экенин дагы бир жолу эсиниздерге салалы. Андыктан анын колдонулушу менен тааныш болушуңуз керек.

Андан кийин, биз жаңы кодду кошкубуз келет. Бул жаңы коддун максаты колдонуучуга өзү каалаган баатырдын түрүн тандап алууну камсыз кылуу болот. Биз аларга үч жолду беребиз: **кадимки, мутант же робот**.

```
# Колдонуучуга түзө турган баатырдын түрүн тандоого мүмкүнчүлүк берүү
print("Төмөнкүлөрдөн бирөөнү тандаңыз: ")
print("Кадимки Супер Баатыр үчүн 1 баскычын басыңыз ")
print("Мутант Супер Баатыр үчүн 2 баскычын басыңыз ")
print("Робот Супер Баатыр үчүн 3 баскычын басыңыз ")
жооп2 = input()
```

Андан кийин **if-elif-else** блогу колдонуучунун жообунун маанисин текшерип чыгат. Биз аны жооп2 өзгөрмөсүндө сактадык жана ошого жараша жооп беребиз. Мисалы, колдонуучу 1-вариантты тандаса, кадимки супер баатыр жаралат: вариант 2, мутант жана башкалар.

Мында коддун блогу:

```
if жооп2=='1':
    # супербаатыр объектисин түзүү
    баатыр = супербаатыр()
    # Түзүлгөн объекттин натыйжаларын, анын ичинде анын параметрлерин
    #басып чыгарабыз
    print("Сиз Кадимки Супер Баатыр жараттыңыз!")
    print("Статистика, ысым жана супер күчтөрүн түзүү.")
    # Эффект түзүү
    for i in range(1):
        print(".....")
        time.sleep(3)
```

```
        print("(Аа...күткөндү жаман көрөсүңбү?...)")

    for i in range(2):
        print(".....")
        time.sleep(3)

    print("(дээрлик даяр...)")
    print(" ")
    print("Баатырдын аты %s." % (баатыр.суперАталыш))
    print("Анын күчү: ", баатыр.күч)
    print("Жаңы статистикасы:")
    print("")
    print("Зээндүүлүгү: ", баатыр.зээндүүлүк)
    print("Тапкычтыгы: ", баатыр.тапкычтык)
    print("Чыдамдуулугу: ", баатыр.чыдамдуулук)
    print("Акылмандуулугу: ", баатыр.акылмандык)
    print("Дисциплинасы: ", баатыр.дисциплина)
    print("Эптүүлүгү: ", баатыр.эптүүлүк)
    print("Ылдамдыгы ", баатыр.ылдамдык)
    print("")
elif жооп2=='2':
      # Мутант объектисин түзүү
      баатыр2 = Мутант()
      print("Статистика, ысым жана супер кубаттуулуктарды түзүү.")

      # эффект түзүү
      for i in range(1):
          print(".....")
          time.sleep(3)
          print("(Аа...күткөндү жаман көрөсүңбү...)")
      for i in range(2):
          print(".....")
          time.sleep(3)

      print("Мутанттын  аты %s." % (баатыр2.суперАталыш))
      print("Анын күчү: ", баатыр2.күч)
      print("Жаңы статистикасы:")
      print("")
```

```
    print("Зээндүүлүгү: ", баатыр2.зээндүүлүк)
    print("Тапкычтыгы: ", баатыр2.тапкычтык)
    print("Чыдамдуулугу: ", баатыр2.чыдамдуулук)
    print("Акылмандуулугу: ", баатыр2.акылмандык)
    print("Дисциплинасы: ", баатыр2.дисциплина)
    print("Эптүүлүгү: ", баатыр2.эптүүлүк)
    print("Ылдамдыгы ", баатыр2.ылдамдык)
    print("")

elif жооп2=='3':

    # Робот объектисин түзүү

    баатыр3 = Робот()

    print("Статистика, ысым жана супер кубаттуулуктарын түзүү.")

    # эффект түзүү
    for i in range(1):
       print(".....")
       time.sleep(3)
       print("(Аа...күткөндү жаман көрөсүңбү...)")

    for i in range(2):
         print(".....")
         time.sleep(3)
  print("Роботтун аты %s." % (баатыр3.суперАтальш))
  print("Анын күчү: ", баатыр3.күч)
  print("Жаңы статистикасы:")
  print("")
  print("Зээндүүлүгү: ", баатыр3.зээндүүлүк)
  print("Тапкычтыгы: ", баатыр3.тапкычтык)
  print("Чыдамдуулугу: ", баатыр3.чыдамдуулук)
  print("Акылмандуулугу: ", баатыр3.акылмандык)
  print("Дисциплинасы: ", баатыр3.дисциплина)
  print("Эптүүлүгү: ", баатыр3.эптүүлүк)
  print("Ылдамдыгы ", баатыр3.ылдамдык)
  print("")
```

```
else:
     print("Туура жоопту тандаган жоксуз! Эми программа өзүн-өзү жок
     кылат!")
```

Акыры биз дагы **import time** -убакытты импорттошубуз керек. Болбосо биздин драмалык эффекттерибиз иштебей калат! Биз муну биздин коддун эң жогору жагында, **import random** - импорттолгон кокустук билдирүүсүнүн астында жасайбыз.

Жаңы жана өркүндөтүлгөн Супер Баатыр генератору - 3000 коду!

Бардык бөлүктөрдү коддоп бүткөндөн кийин алардын бардыгы иретке келтирилгендигин текшериңиз. Кодуңузду төмөнкү код менен салыштырып, бардыгы дал келгенин текшериңиз. Андан кийин программаны бир нече жолу иштетип, анын иштешин көрүү үчүн бардык параметрлерди сынап көрүңүз:

```
# Биз кокустан сандарды иштеп чыгышыбыз үчүн, random модулун импорттоп
#алыңыз

# Убакыттын драмалык эффектиси үчүн импорттолуучу time модулу

import random

import time

# Биз жараткан бардык баатырлар үчүн шаблондун ролун аткарган Супер
#Баатырлар классын түзүңүз

class супербаатыр():

# Биздин классты баштоо жана анын атрибуттарын орнотуу
     def __init__(self):
          self.суперАталыш = суперАталыш
          self.күч = күч
          self.тапкычтык = тапкычтык
          self.зээндүүлүк = зээндүүлүк
          self.чыдамдуулук = чыдамдуулук
          self.акылмандык = акылмандык
          self.дисциплина = дисциплина
          self.эптүүлүк = эптүүлүк
          self.ылдамдык = ылдамдык

# Random() функциясын колдонуп, ар бир статуска кокустук маанилерин кошуу
тапкычтык = random.randint(1,20)
зээндүүлүк = random.randint(1,20)
```

```
чыдамдуулук = random.randint(1,20)
акылмандык = random.randint(1,20)
дисциплина = random.randint(1,20)
эптүүлүк = random.randint(1,20)
ылдамдык = random.randint(1,20)

# Мүмкүн болгон супер күчтөрдүн тизмесин түзүү
суперКүчтөр = ['Учуучу', 'Укмуш Күчтүү', 'Телепатия', 'Супер Ылдам', 'Көп
Хот-Догдорду Жей Алат', 'Аркандан Секиргени Жакшы']

# Супер күчтөр тизмесинен супер күчтү кокустан тандап алуу
# жана аны күч өзгөрмө ыйгаруу
күч = random.choice(суперКүчтөр)

# Мүмкүн болгон ысымдардын жана фамилиялардын тизмесин түзүү
суперЫсымдар  = ['Керемет','Уатта','Кутурган','Укмуш', 'Таң Калычтуу',
'Татыктуу', 'Даңазалуу', 'Ортодон-Жогору', 'Ал Жигит', 'Өзгөчө']
суперФамилиялар = ['Бала', 'Киши', 'Динго', 'Бифкейк', 'Кыз', 'Аял',
'Жигит','Баатыр', 'Макс', 'Кыял', 'Мачо','Аргымак']
# супербаатырдын атын кокусунан тандоо
# Муну биз эки ысым тизмебиздин ар биринен бирден ат тандоо менен жасайбыз
# Жана суперАталыш өзгөрмөсүнө кошуңуз
суперАталыш = random.choice(суперЫсымдар)+ " " +random.choice(суперФамилиялар)

# Мутант аттуу Супер Баатырдын субклассын түзүү
# Мутант каармандары ылдамдыгы үчүн +10 бонус алышат.
class Мутант (супербаатыр):
     def __init__(self):
         супербаатыр.__init__(self)
         print("Сиз Мутант жараттыңыз!")
         self.ылдамдык = self. ылдамдык + 10

# Робот аттуу Супер Баатырдын субклассын түзүү
# Робот каармандары braun упайына + 10 бонус алышат.
class Робот(супербаатыр):
     def __init__(self):
         супербаатыр.__init__(self)
```

```
        print("Сиз Робот жараттыңыз!")
        self.тапкычтык = self.тапкычтык + 10

# Киришүү тексти

print("Супер Баатыр генератору 3000 менен супер баатыр жаратууга даярсызбы?")

# Колдонуучуга суроо берип, ага жооп сунуштаңыз
# input() баскычтопто эмне тергенин "угат"
# Андан кийин колдонуучунун жообун бардык чоң тамгага өзгөртүү үчүн upper() колдонобуз.

print("О же Ж жазыңыз:")

жооп = input()
жооп = жооп.upper()


# While цикли "О" жообун текшерүү үчүн
# Бул цикл жооптун мааниси "О" БОЛГОНЧО улантылат.
# Колдонуучу "О" деп жазганда гана цикл чыгып, программа улана берет.

while жооп != "О":
    print("Кечиресиз, бирок улантуу үчүн О теришиңиз керек!")
    print("О же Ж жазыңыз:")
    жооп = input()
    жооп = жооп.upper()
print("Сонун, баштадык!")


# Колдонуучу баатырдын кайсы түрүн жаратууну өзү тандап алсын.
print("Төмөнкүлөрдөн бирөөнү тандаңыз: ")
print("Кадимки Супер Баатыр үчүн 1 баскычын басыңыз ")
print("Мутант Супер Баатыр үчүн 2 баскычын басыңыз ")
print("Робот Супер Баатыр үчүн 3 баскычын басыңыз ")

жооп2 = input()

if жооп2=='1':

    # супербаатыр объектисин түзүү

    баатыр = супербаатыр()
```

```
    #Биз түзүлгөн объекттин натыйжаларын, анын параметрлерин басып
    чыгарабыз.
    print("Сиз кадимки супер баатырды жараттыңыз!")
    print("Статистика, ысым жана супер күчтөрдү түзүү.")

    # Эффект жаратуу
    for i in range(1):
       print(".....")
       time.sleep(3)

       print("(Аа... Сиз күткөндү жаман көрөсүзбү?...)")

    for i in range(2):
       print(".....")
       time.sleep(3)
    print("(дээрлик баштадык...)")
    print(" ")
    print("Баатырдын аты %s." % (баатыр.суперАталыш))
    print("Анын күчү: ", баатыр.күч)
    print("Жаңы статистикалар:")
    print("")
    print("Зээндүүлүгү: ", баатыр.зээндүүлүк)
    print("Тапкычтыгы: ", баатыр.тапкычтык)
    print("Чыдамдуулугу: ", баатыр.чыдамдуулук)
    print("Акылмандуулугу: ", баатыр.акылмандык)
    print("Дисциплинасы: ", баатыр.дисциплина)
    print("Эптүүлүгү: ", баатыр.эптүүлүк)
    print("Ылдамдыгы ", баатыр.ылдамдык)
    print("")

elif жооп2=='2':
     # Мутант объектисин түзүү
     баатыр2 = Мутант ()
     print("Статистика, ысым жана супер күчтөрдү түзүү.")

    # эффект жаратуу

     for i in range(1):
        print(".....")
```

```
        time.sleep(3)

        print("(Аа...Сиз күткөндү жаман көрөсүзбү?...)")

    for i in range(2):
        print(".....")
        time.sleep(3)

    print("Мутанттын аты %s." % (баатыр2.суперАталыш))
    print("Анын күчү: ", баатыр2.күч)
    print("Жаңы статистикалар:")
    print("")
    print("Зээндүүлүгү: ", баатыр2.зээндүүлүк)
    print("Тапкычтыгы: ", баатыр2.тапкычтык)
    print("Чыдамдуулугу: ", баатыр2.чыдамдуулук)
    print("Акылмандуулугу: ", баатыр2.акылмандык)
    print("Дисциплинасы: ", баатыр2.дисциплина)
    print("Эптүүлүгү: ", баатыр2.эптүүлүк)
    print("Ылдамдыгы ", баатыр2.ылдамдык)
    print("")

elif жооп2=='3':
    # Робот каарманын түзүү
    баатыр3 = Робот()

    print("Статистика, ысым жана супер күчтөрдү түзүү.")

        # эффект жаратуу

    for i in range(1):
        print(".....")
        time.sleep(3)

    print("(Аа... Сиз күткөндү жаман көрөсүзбү?...)")

    for i in range(2):

        print(".....")

        time.sleep(3)

    print("Роботтун аты  %s." % (баатыр3.суперАталыш))
    print("Анын күчү: ", баатыр3.күч)
```

```
      print("Жаңы статистикасы:")
      print("")
      print("Зээндүүлүгү: ", баатыр3.зээндүүлүк)
      print("Тапкычтыгы: ", баатыр3.тапкычтык)
      print("Чыдамдуулугу: ", баатыр3.чыдамдуулук)
      print("Акылмандуулугу: ", баатыр3.акылмандык)
      print("Дисциплинасы: ", баатыр3.дисциплина)
      print("Эптүүлүгү: ", баатыр3.эптүүлүк)
      print("Ылдамдыгы ", баатыр3.ылдамдык)
      print("")
else:
     print("Туура жоопту тандаган жоксуз! Эми программа токтойт")
```

Бул эпизоддо

Ушул бөлүмдө биз укмуштай секириктерди жасадык. Анткени бул китепте талкуулаган бардык темаларды өздөштүрүү татаал болду. Туура, калгандары буга салыштырмалуу оңой-олтоң!

Кыскача эскертүү, келечектеги жардамчы катары, ушул бөлүмдө камтылган нерселердин кыскача баяндамасы берилген:

- ОБП- объектиге багытталган программалоо;
- Объектке багытталган программалоо - бул биздин программаларыбызда кайрадан колдонула турган код түзүүнү үйрөткөн түшүнүк;
- Процедуралык программалоо, көбүнчө, сап боюнча же сызыктуу түрдө аткарууга ылайыкталган код жазууну камтыйт;
- ОБП өзөгү класстардын, объектилердин жана методдордун айланасында жүрөт;
- Класс план же шаблонго окшош;
- Объект - бул класстын мисалы. Мисалы, класс үйдүн планы болсо, объект ушул пландан түзүлгөн чыныгы үй;
- Класстын ичинде колдонулган функция метод катары белгилүү;
- Классты аныктоо үчүн биз төмөндөгүдөй теребиз:

  class супербаатыр:

  ... код ... ;

- Def оператору класстагы методду аныктоо үчүн колдонулат. Мисалы :

```
def Учуу:
... код ...;
```

- __init__ методу инициализациялоо үчүн колдонулат;
- self класстын мисалын түзгөндө параметрге шилтеме берүү үчүн колдонулат;
- Биз объектини өзгөрмөгө ыйгаруу менен аныктайбыз, мисалы:

  ```
  баатыр = супербаатыр();
  ```

- Класстар иерархиялык мүнөзгө ээ; бизде негизги класс болушу мүмкүн (ата-эне), андан кийин субкласс (бала);
- Субкласстар негизги класстын же Суперкласстын методдорун жана параметрлерин мурастап алышат;
- Субклассты аныктоо үчүн төмөнкүдөй кодду колдоносуз:

  ```
  class Мутант (супербаатыр).
  ```

9 - БӨЛҮМ

БАШКА МААЛЫМАТ СТРУКТУРАЛАРЫН ТААНЫШТЫРУУ

Кайрадан кош келиңиз, өсүп келе жаткан жаш баатыр! Сиз узак убакыт бою үй тапшырмасын аткарып, үй жумуштарын жасап жана, албетте, кылмыштуулукка каршы күрөшүп жүргөндөйсүз. Эми жашылчаларды жеп, идиш-аякты тазалап, анан тишиңизди тазалап коюуңуз гана калды!

Албетте, бүгүн кечинде тиштериңизди ылдам тазаласаңыз болот. Анткени сиздин программалык мээңизге дагы бир нече кылмыштуулукка каршы күрөшүү жөндөмүн киргизүү үчүн убакыттан бир аз уттуруп албайлы! Менин айтайын дегеним - эгер сиз тамакты чайнап берген тиркемени программалап алсаңыз, анда тиштин кимге кереги бар?

Тамашасы жок, барып тишиңизди тазалаңыз...

Биз бул китептин жарымын окуп үйрөнүп койдук. Сиз иштеп баштаганда же өзүңүздүн бестселлер программаңызды иштеп чыгарганыңызда өзүңүз менен кошо ала турган программалоонун жакшы тажрыйбаларын жана практикалык тил көндүмдөрүнүн жакшы, бекем пайдубалын үйрөнүп алдыңыз.

Албетте, үйрөнө турган дагы көп нерсе бар. Программалык билимдин бул кереметтүү томун окуп чыккандан кийин да, сиздин сапарыңыз соңуна чыкпайт. Программист болуу өмүр бою студент болуу сыяктуу - сиз ар дайым өз жөндөмүңүздү өркүндөтүп, эң жаңы жана мыкты технологияны үйрөнүп турушуңуз керек.

Тил жаңыртууларынан тышкары (компьютердик тилдер тез-тез жаңыланып турарын айтканбыз), кандайдыр бир мезгилде сиз башка программалоо тилдерине жана алкактарына кирип кетүүнү каалайсыз. Бирок, бул жакынкы келечектеги дагы бир бөлүм үчүн тема болуп калат.

Бул бөлүм ошол эле учурда, артка кылчайтып көз чаптыртат. Биз маалымат структураларын мурунку бөлүмдөрдө талкуулаганбыз. Өзгөрмө жана тизмелер менен иштөөнү үйрөндүк. Бул экөө тең маалыматты сактоо үчүн колдоно турган күчтүү шаймандар болсо да, алар биз үчүн бирден-бир маалымат структурасы эмес.

Дагы эки нерсени талкуулашыбыз керек. Алар: *кортеждер* жана *сөздүктөр*. Бул эпизоддо ушул теманын айланасында сөз кылабыз. Ошондой эле, ушул эки сактагыч бөлүктү иштетүү боюнча айрым функцияларды карап чыгып, аларды жаңы программаларга киргизебиз.

Эмне жасоо керек экенин билесизби? Жок. Ушул жумадагы математика тестинин жоопторун аңдуу үчүн рентген көзүңүздү колдонбоңуз.

Тишиңизди тазалаңыз!

Андан кийин бул жерге кайтып келип, баатырдай болуп код жазууну үйрөнүүгө даярданыңыз. Алардын дагы бир нечесин үйрөнөсүз.

Дагы көбүрөөк маалымат структуралары

Жогоруда айтылгандай, биз буга чейин эки маалымат структурасын карадык: тизмелер жана өзгөрмөлөр. Маалыматтар структурасы - бул маалыматты же маалыматтын бир бөлүгүн/бөлүктөрүн камтыган сактоочу контейнер. Бул маалымат структураларында маалыматты сактай алабыз. Маалыматтарды алып сала салабыз жана аларга ар кандай маалыматтарды кошо алабыз. Ошондой эле, биз маалыматтарды чыгарып, программанын бир бөлүгү үчүн колдонуп (метафоралык түрдө), аны кайра жайгаштыра алабыз (ал контейнерден эч качан чыкпайт).

Өзгөрмө бир маалыматты камтый алат. Бул маалыматтар тамга, сан же бүтүн сан, белгилер сабы, сүйлөм, абзац жана башкалар болушу мүмкүн. Андан тышкары, өзгөрмө тизмелер сыяктуу объектилерди камтышы мүмкүн. Бул техникалык жактан алганда "бирден ашык" маалыматты камтыйт дегенди билдирет. Ошол эле учурда тизме бир нече маалыматты камтыйт.

Өзгөрмөнү бир файл папкасы деп эсептеп, тизмени файл шкафы катары көрүңүз.

Өзгөрмөнү аныктоо үчүн, эсиңизде болсо керек, биз төмөнкүдөй кодду колдонобуз:

```
a = "Саламатсызбы!"
b = 7
c = "Саламатсызбы, мен өзгөрмө түрмөдө отурам!"
```

Тизмени аныктоо үчүн биз төмөнкү ыкманы колдонобуз:

```
жумушчу = ['Чоң Е.', 'Блок Хоган', 'Чарба  башчы Альфредо']
баалар = [5, 10, 20, 30, 40, 50]
```

Эгер биз өзгөрмөдөн басып чыгарууну кааласак, анда төмөнкү саптар боюнча бир нерсе жазмакпыз.

```
print(a)
print("Сизде ушунча алма бар: ", b)
```

% s форматтагычын өзгөрмөңүздүн ордуна колдонсоңуз болот. Мисалы, сиз сүйлөм жазгыңыз келди дейли: “Сизде X алма бар”, мында X – өзгөрмө, b - мааниси. Эгер сиз ушул кодду төмөндөгүдөй терсеңиз:

```
print("Сиздин %s алмаңыз бар!" , b)
```

Аны иштетип жатканда сиз төмөнкү натыйжаны алмаксыз:

```
_____________________________________________________________________________________

Сиздин 7 алмаңыз бар!

-------------------------------------------------------------------------------------------------------------------
```

Тизмени басып чыгаруу үчүн биз төмөнкүлөрдү колдоно алабыз:

```
print(жумушчу)
```

Же тизмеден бир элементти басып чыгаруу үчүн, биз анын индексин колдонобуз (Эсиңизде болсун: тизмедеги биринчи элемент 0 индексинде жайгашкан):

```
print(жумушчу [1])
```

Бул төмөнкүнү басып чыгармак:

```
_____________________________________________________________________________________

Блок Хоган

-------------------------------------------------------------------------------------------------------------------
```

Эми өзгөрмөлөрдү жана тизмелерди карап чыгып, маалымат структуралары кандайча аз-аздан иштей тургандыгы жөнүндө эсибиздегилерди жаңыртып алганыбыздан кийин, Python сунуш кылган башка эки түрдүү маалымат структураларын үйрөнүүгө өтөлү.

Кортеж деген эмне?

Кортеждер (англ. tuples) - тизмелер жана өзгөрүлмө сыяктуу, маалымат структурасынын бир түрү. Бирок өзгөрмөлөрдөн жана тизмелерден айырмаланып, кортеждер *туруктуу* деп эсептелет. Сиз алардын маанисин алмаштыра албайсыз же аларды кадимкидей түрдө өзгөртө албайсыз дегенди билдирет.

Кортеждер *элементтердин* иреттелген ырааттуулугунан турат. Бул элементтер же маанилер кашаанын ортосунда аныкталат жана үтүр менен ажыратылат. Кортежди аныктоо үчүн төмөнкүдөй кодду колдоносуз:

```
кылмышкер = ('Каш көтөрүүчү', 'Ачуулуу Хеклер', 'Анчалык бактылуу эмес адам', 'Каргышка калган Рейзер')
```

Тизмедегидей эле, жөнөкөй print() функциясын колдонуп, кортежибиздин мазмунун басып чыгара алабыз:

```
print(кылмышкер)
```

Натыйжада:

```
('Каш көтөрүүчү', 'Ачуулуу Хеклер', 'Анчалык бактылуу эмес адам',
'Каргышка калган Рейзер')
```

Тизмелерге окшош, кортеждеги элементтерге алардын индексинин номери боюнча шилтеме берсе болот. Кортеждеги элементтер 0 индексинен башталат. Демек, мисалы, биз *кылмышкер* адамдар кортежиндеги биринчи элементти басып чыгарууну кааласак, анда төмөнкүнү колдонмокпуз:

```
print(кылмышкер[0])
```

Бул бизге коркунучтуу шумпайды берет:

```
Каш көтөрүүчү
```

Эгер биз шумпай кортежин сүйлөмдүн бир бөлүгү катары колдонууну кааласак, анда аны жасоонун бир нече жолу бар:

```
# Биздин кортежди аныктоо
кылмышкер = ('Каш көтөрүүчү', 'Ачуулуу Хеклер', 'Анчалык бактылуу эмес
адам', 'Каргышка калган Рейзер')

# Кортеждеги элементтерди басып чыгаруу
print(кылмышкер)

# Кортеждеги элементтерди бирден басып чыгаруу
print(кылмышкер[0])
print(кылмышкер[1])
print(кылмышкер[2])
print(кылмышкер[3])

# Сүйлөмдөргө кортеждин элементтерин кошуунун жолдору

print("Биринчи каардуу күнөөкөр ", кылмышкер[0])
print("Экинчи каардуу коркунучтуу адам " + кылмышкер[1])
```

Бизге төмөндөгүнү берет:

```
('Каш көтөрүүчү', 'Ачуулуу Хеклер', 'Анчалык бактылуу эмес адам',
'Каргышка калган Рейзер')
Каш көтөрүүчү
Ачуулуу Хеклер
```

```
Анчалык бактылуу эмес адам
Каргышка калган Рейзер
Биринчи каардуу күнөөкөр  Каш көтөрүүчү
Экинчи каардуу коркунучтуу адам Ачуулуу Хеклер
```

Элементтерди кортежде колдонуунун дагы бир жолу - аларды кесүү. Кортежди кескенде, сиз колдонгуңуз келген бир катар маанилерди көрсөтүп жатасыз. Мунун форматы мисал катары - *кылмышкер*[0:3]. Эгерде биз ушул кодду иштетсек, анда:

```
print(кылмышкер[0:3])
```

натыйжасы мындай болмок:

```
('Каш көтөрүүчү', 'Ачуулуу Хеклер', 'Анчалык бактылуу эмес адам')
```

Сиздин оюңузду билем. 3 - индекстеги пункт - "Каргышка калган Рейзер" эмне үчүн ал басылып чыккан жок?- деп атасыз го.

Жообу жөнөкөй. Биз кескенде, кош чекиттин *алдындагы* биринчи сан программага эмнеден баштоо керектигин айтат. Кош чекиттен *кийинки* сан ушул санга *чейин* бүтүшү керектигин билдирет.

Эгерде биз print (*кылмышкер*[0: 4]) деп жаза турган болсок, анда ал биздин төрт элементти тең басып чыгармак, анткени Python 4 индекстеги элементти издейт. Андай элемент жок болгондуктан, ал ага чейинки элементти басып чыгарат.

Эске салсак, индекстин баштапкы саны 0 болбошу керек. Эгерде, биз, мисалы, кортежибиздеги биринчи элементти басып чыгарууну каалабасак, анда жөн гана print(*кылмышкер*[1: 4]) колдонсок болот. Ал экинчи пунктту басып чыгарышы мүмкүн:

```
('Ачуулуу Хеклер', 'Анчалык бактылуу эмес адам', 'Каргышка калган
Рейзер')
```

Кортеждерди жасай турган дагы бир айла-амал - аларды бириктирүү. Мисалы, сизде жалтырак кызгылт көк түстөгү коргоочу кийим жана буурчактай болгон тегерек гүлү бар дагы башка коргоочу кийимдери бар кортеж бар деп коёлу. Балким, сиздин коргоочу кийимдерге толгон шкафтарыңыздын көптүгүнөн тажагандырсыз, аларды бириктиргиңиз келип жакткандыр. Эгер ошондой болсо, анда сиз ар дайым кортеждериңизди бириктирип, жаңы кортеж түзө аласыз.

Бул мисалды карап көрөлү:

```
# Менин кочкул кызыл түстөгү баш кийимдеримдин кортежин түзүү

кызылБашкийим = ('Бүктөлгөн кочкул кызыл түстөгү баш кийим', 'Кыска кочкул кызыл түстөгү баш кийим', 'Тешиктери бар кочкул кызылтүстөгү баш кийим')

# Менин чекиттүү баш кийимдеримдин кортежин түзүү

чекиттүүБашкийим = ('Кара жана Ак чекиттүү баш кийимдерим', 'Ак жана Агыш чекиттүү баш кийимдер', 'Көк чекиттүү баш кийимдер')

# Эки кортежди бириктирип же кошуп жаңы кортеж түзүү

бардыкБашкийимдерим = (кызылБашкийим + чекиттүүБашкийим)

# Жаңы түзүлгөн кортеждин маанилерин басып чыгаруу

print(бардыкБашкийимдерим)
```

Бул код *кызылБашкийим* кортежин *чекиттүүБашкийим* тизмесиндеги элементтер менен айкалыштырат жана аларды *бардыкБашкийимдерим* деп аталган жаңы түзүлгөн кортежде сактайт. Эгер сиз ушул коддун үзүндүсүн иштетсеңиз, төмөнкүнү аласыз:

```
('Бүктөлгөн кочкул кызыл түстөгү баш кийим', 'Кыска кочкул кызыл түстөгү баш кийим', 'Тешиктери бар кочкул кызылтүстөгү баш кийим', 'Кара жана Ак чекиттүү баш кийимдер', 'Ак жана Агыш чекиттүү баш кийимдер', 'Көк чекиттүү баш кийимдер')
```

Бул *кызылБашкийим* же *чекиттүүБашкийим* маанилерин өзгөртпөйт же таасир этпейт. Эсиңизде болсун, кортеждеги маанилерди өзгөртө албайсыз.

Кортеждер үчүн «+» же бириктирүү операторун колдонуудан тышкары, кортежде сакталган маанилерди кайталоо үчүн «*» же көбөйтүү операторун колдонсоңуз болот:

```
print(бардыкБашкийимдерим[1] * 3)
```

Бул *бардыкБашкийимдерим* кортежиндеги индекс 1де жайгашкан элементти үч жолу басып чыгарат, натыйжада:

```
Кыска кочкул кызыл түстөгү баш кийимКыска кочкул кызыл түстөгү баш кийимКыска кочкул кызыл түстөгү баш кийим
```

Биздин кортежде көрсөтүлгөн элементтерден кийин боштук жок экендигин эске алыңыз, ошондуктан биз аларды басып чыгарганда, боштуктар жок болот.

Кортеж функциялары

Тизмелердегидей эле, кортеждерде дагы, алардын ичинде сакталган маалыматтар менен иштөө үчүн сиз колдоно ала турган бир катар функциялар бар. Бул функциялар кортеждерге гана мүнөздүү эмес жана аларды Python кодуңуздун башка жерлеринде да колдонсоңуз болот.

Эки тааныш кортеж функциясы min() жана max() болушу керек. Аларды мурунку бөлүмдө колдонгонуңузду эстесеңиз болот. Бул эки функцияны кортежде колдонууда алар кадимки ролду аткарышат. Башкача айтканда, кортеждеги элементтин (элементтердин) минималдуу жана максималдуу маанилерин кайтарып беришет.

Мисалы:

```
# Сандардын жыйындысын камтыган кортеж түзүңүз

эңкичине_маани = (1, 5, 10, 15, 20, 50, 100, 1000)

# Кортеждеги эң кичине маанини кайтаруу үчүн минимум функциясын колдонуңуз

print(min(эңкичине_маани))
```

Бул код кайтып келет:

```
_____________________________________________________________________

1
_____________________________________________________________________
```

Бул техникалык жактан алганда, биздин кортеждеги эң кичине маанидеги сан.

Эгер биз эң чоң санды кааласак, анда max() функциясын колдонмокпуз:

```
# Сандардын жыйындысын камтыган кортеж түзүңүз
эңчоң_маани = (1, 5, 10, 15, 20, 50, 100, 1000)

# Кортеждеги эң чоң маанини кайтаруу үчүн max()  функциясын колдонуңуз
print(max(эңчоң_маани))
```

Сиз болжогондой, 1000 кайтып келет.

Дагы бир пайдалуу кортеж функциясы len() болуп саналат. Ал эсиңизде болсо, саптын узундугун же тизмедеги элементтердин санын кайтарып берет. Кортеж менен колдонулганда, ал кортеждеги камтылган элементтердин санын кайтарып берет.

```
# Ар кандай элементтерди камтыган кортеж түзүңүз
супер_чач = ('Супер Чач Тарач', 'Сакалдуу Мачо', 'Грагантуан Чачынын Ээси',
```

```
'Кыжырды Келтирген Парик', 'Такыр Баш')
# Биздин кортеждеги элементтердин санын басып чыгарыңыз
print(len(супер_чач))
```

Бул 5ти кайтарып берет, анткени биздин *супер_чач* кортежибизде беш элемент бар.

len() функциясын колдонуу мисалдары, компаниядагы кызматкерлердин санын билүү керек болгондо, же *Мурунку Супер Шумпайлардын Кампасында* канча шумпайды камап койгондугуңузду билиш керек болгон сценарийлерди камтыйт. Эгерде сизде ушул жаман каармандардын аттарын камтыган кортеж болсо, анда жөн гана len() функциясын колдонуп, алардын санын тез эсептеп чыксаңыз болот.

Албетте, эгерде бизде канча кылмышкер бар экендигин тез билгибиз келсе, кылмышкерлер фильминдеги каардуу адамдардын санын кайтарып берүү пайдалуу болмок, бирок бул тизменин кандайдыр бир тартипте басылып чыкканын көргүбүз келсечи, анда ал үчүн кандайдыр бир функция болгон болсо жакшы болмок...

О, күтө туруңуз, андай функция бар!

```
# Түрмөдө камалып жаткан кылмышкерлердин тизмеси
кылмышкер = ('Укпаган адам', 'Тайгактын Белгиси', 'Мистер Миллениал',
'Джек Хаммер', 'Ызылдаган Аары', 'Сүттүн Баарын Ичкен Адам', 'Керемет-
Веджи', 'Качкан Эчки')

# Кортеждеги кылмышкердин тизмесин иреттелген тартипте басып чыгарыңыз
print(sorted(кылмышкер))
```

Кортеждин сорттолгон тизмесин (же ал боюнча тизмени) басып чыгаруу үчүн мурунку коддо көрсөтүлгөндөй, sorted() функциясын колдонобуз. Белгилей кетүүчү бир нече маанилүү жагдай бар: Биринчиси, сорттолгон жыйынтык алфавиттик тартипте кайтарылат. Экинчиси жана эң маанилүүсү, sorted() функциясы гана иреттелген чыгууну кайтарып берет. Бул биздин кортеждеги маалыматтарды чындыгында иргебейт. Эске салсак, атүгүл, sorted() сыяктуу күчтүү функция менен кортеждер өзгөрүлбөйт жана аларды өзгөртүүгө болбойт!

Эгер мурунку кодду иштете турган болсок, анда биздин натыйжабыз мындай:

```
['Джек Хаммер', 'Качкан Эчки', 'Керемет-Веджи', 'Мистер Миллениал',
'Сүттүн Баарын Ичкен Адам', 'Тайгактын Белгиси', 'Укпаган адам',
'Ызылдаган Аары']
```

Албетте, биз сандарды дагы да оңой ирээттей алабыз. Бул кодду карап көрөлү:

```
 # Биз иреттей турган сандардын кортежи
сандар_ирети = (10, 20, 5, 2, 18)
# Биздин кортеждеги сандарды сорттоо
print(sorted(сандар_ирети))
```

Эгер биз кодду иштетсек, анда ал бул натыйжаны кайтарып берет:

```
[2, 5, 10, 18, 20]
```

Сандарга толгон кортежди карап жатып, дагы бир пайдалуу функцияны, sum() карап көрөлү. Берилген башка функциялар сыяктуу эле, sum() сизге да тааныш болушу керек. Ал маалымат структурасындагы сандарды суммалоо же жалпылаштыруу үчүн колдонулат.

Кортеждеги элементтердин жалпы санын кошуу үчүн колдоно турган код:

```
# Биз кошо турган сандар кортежи
сандар_суммасы = (10, 20, 5, 2, 18)

# Кортеждин элементтерин суммалоо же кошуу
print(sum(сандар_суммасы))
```

Муну иштетсек, бизге *сандар_суммасы*кортежиндеги элементтердин жалпы суммасы берилет, ал - 55.

Акыр-аягы биз дагы башка маалымат структураларын, мисалы, тизмелерди жана өзгөрмөлөрдү tuple() функциясын колдонуп, кортежге айланта алабыз:

```
# Биз кортежге айланта турган тизме
кылмышкерТизмеси = ['Укпаган адам', 'Тайгактын Белгиси', 'Мистер Милленииал', 'Джек Хаммер', 'Ызылдаган Аары', 'Сүттүн Баарын Ичкен Адам', 'Керемет-Веджи', 'Качкан Эчки']

# tuple() колдонуп кылмышкерТизмесин кортежге айландыруу
tuple(кылмышкерТизмеси)

# Кортежге айланта турган сап
кылмышкер1 = "Мурутчан Коркунуч!"
tuple(кылмышкер1)
```

Кортеждер менен иштөө мындан да кызыктуурак

Күчтүү кортеж менен иштөөнү үйрөндүк деп кубанып турганда сиз бонустук раундга туш болдуңуз! Маалыматтар структурасынын кийинки түрүнө өтүүдөн мурун дагы бир канча нерсени үйрөнүшүбүз керек.

Кортежге киришүүбүздө кортеждердин тизмелерден айырмаланып, бир маанилүү өзгөчөлүгү бар экендигин билдик. Ал өзгөчөлүк менен кортеждер өзгөрүлбөйт жана алардагы маалыматтарды эч кандай жол менен өзгөртүүгө мүмкүн эмес. Ал эми тизмелерди башкарууга, жаңыртууга жана кошууга болот.

Эгерде сиздин маалымат түзүмүңүз үчүн маалыматтын бүтүндүгүн ойлонсоңуз, бул кортеждерди күчтүү куралга айландырат. Эгер сизде таптакыр өзгөрүлбөшү керек болгон элементтер тобу бар болсо, аларды сактоо кортеждин аткарган эң негизги жумушу болот.

Айткандай эле, программаңыздан кортежди алып салууну каалаган учурлар болот. Мисалы, сизде баатырлар жана терс каармандарда болушу мүмкүн болгон чач, сакал, муруттун бардык түрүн сактай турган кортеж бар. Күтүлбөгөн жерден (балким) ушул жасалгалоо модадан чыгып калса эмне болот? Кортеждеги элементтер кайрадан колдонулбашы үчүн жана биздин кодду мүмкүн болушунча тыкан жана натыйжалуу кармоо үчүн бизде эки жол бар.

Биринчиден, биз жөн гана # **же “ ‘ ’ ”** комментарийди колдонуп, биздин кортежге шилтеме берген бардык коддорду комментарийлеп алсак болот. Бирок, кимдир бирөө сиздин кодду комментарийлебей калса, анын натыйжасында каталар пайда болушу мүмкүн, же болбосо, чач, сакал, мурут тренди кайтып келиши мүмкүн.

Дагы бир вариант - кортежге шилтеме берген кодду жок кылуу же өзгөртүү, андан кийин кортеждин өзүн жок кылуу.

Бүтүндөй кортежди жок кылуунун жолу бар. Бирок биз кортеждеги элементтерди жок кыла албайбыз. Кортежди кантип жок кылса болорун төмөндө көрүңүз:

```
# Кылмыштуу адамдарга жана баатырларга арналган чач стилдерин камтыган
кортеж
 чач_стили = ('Супер Чач Тарач', 'Сакалдуу Мачо', 'Грагантуан Чачынын
Ээси', 'Чачы кыркылган')
# чач_стилин басып чыгаруу
print(чач_стили)

# Кортежди толугу менен жок кылуу үчүн del колдонулат
del чач_стили

# чач_стили эми бош экендигин көрсөтүү үчүн басып чыгаруу
print(чач_стили)
```

Бул коддун үзүндүсүндө биз алгач *чач_стили* кортежин түзүп, ага бир топ элементтерди ыйгарабыз – бирөөсүнүн аты коркунучтуу угулат - **‘Грагантуан Чачынын Ээси’** (бул эмнени билдирерин мен деле билбейм).

Андан кийин биз жараткан кортеждин чындыгында иштегенин далилдөө үчүн *чач_стили* элементтерин басып чыгарабыз. Адамдар бетине өстүрүүгө даяр турган ырайы суук элементтердин тизмесин көрүп, биз эң жакшысы *чач_стили* коробкасын алып салып, ал эч качан болбогондой түр көрсөтүү керек деп чечтик. Ал үчүн биз **del** операторун колдонобуз: **del** *чач_стили*.

Акыр-аягы, *чач_стили* чынында эле өчүрүлгөндүгүн текшерүү үчүн, аны дагы бир жолу басып чыгарабыз. Бул кодду иштеткенде, натыйжага байланыштуу эки нерсе болот. Алгач, *чач_стили* кортежиндеги элементтер басылып чыгат. Экинчиден, биз ката жөнүндө кабар алабыз.

Эмне үчүн ката жөнүндө билдирүү пайда болду? Биринчи жолу басылып чыккандан кийин биз *чач_стили*н алып салдык; экинчи жолу басып чыгарганы барсак, котормочу таппай калат. Демек, биз дүйнөнү *чач_стили* душманынан арылта алдык!

Баатырдын жашоосундагы дагы бир күн!

Эгер сиз программаны иштеткен болсоңуз, анда мына ушундай натыйжа болот:

```
('Супер Чач Тарач', 'Сакалдуу Мачо', 'Грагантуан Чачынын Ээси', 'Чачы
кыркылган')
Traceback (most recent call last):
     File "C:/Users ... /Python/Python36-32/
     TupleExamples.py", line 101, in <module>
          print(чач_стили)
NameError: name 'чач_стили' is not defined
```

Айрым учурларда, маалыматтарды сактоо үчүн кортеждерди колдонгондо, белгилүү бир элемент биздин маалымат структурабызда канча жолу пайда болгонун билишибиз керек. Мисалы, "Миссисипи" деген сөздө "и” тамгасы канча жолу кезигет. Ошондой эле көрүнүш "с" тамгасында да бар. Эгерде биз ушул сөздү камтыган кортежди түзө турган болсок, анда адамдар бизден кызыктуу бир нерсе айтып берүүнү суранышканда: “Сиз Миссисисиппи деген сөздө канча "и" жана "с" тамгалары бар экендигин билесизби?” - деп сурап, "и" жана "с" тамгалары ушул сөздө канча жолу кайталанганын эсептей алмакпыз. Бул чыныгы болгон окуя, тууган!

Элемент кортежде пайда болгон учурлардын санын эсептөө үчүн, же элементтердин санын эсептөө үчүн, биз count() методун колдонобуз.

```
# Миссисисиппи сөзүн жазуу үчүн колдонулган бардык тамгаларды камтыган кортеж
штат = ("М", "и", "с", "с", "и", "с", "и","с", "и",  "п", "п", "и")
# Эскертүү: Кортежди оңой эле түзсеңүз болот
# tuple() буйругу сапты автоматтык түрдө кортежге айландырат.
# Биздин кортежде канча жолу "и" кездешкенин санап, натыйжасын басып чыгарыңыз
print("Миссисисиппи сөзүндө бир нече 'и' тамгасы бар: ")
print(штат.count('и'))
# Миссисисиппи сөзүндө канча жолу "с" кездешээрин санаңыз
print("Миссисисиппи сөзүндө бир нече ' с ' тамгасы бар: ")
print(штат.count('с'))
```

штат.count('и') кодундагы кашаанын ичиндеги белгилер программага 'и' штат кортежинде канча жолу пайда болгонун эсептөөнү сунуш кылат.

Эгерде биз ушул үлгү кодун иштетсек, анда төмөнкүдөй натыйжа чыкмак:

```
Миссисисиппи сөзүндө бир нече 'и' тамгасы бар:
5
Миссисисиппи сөзүндө бир нече ' с ' тамгасы бар:
4
```

---

in ачкыч сөзүн колдонуп, биздин кортежден бир элементти издей алабыз. Бул ачкыч сөз, негизинен айтканда, "х" мааниси кортежде бар же жок экендигин сурайт:

```
# Миссисисиппи сөзүн жазуу үчүн колдонулган бардык тамгаларды камтыган кортеж
штат = ("М", "и", "с", "с", "и", "с", "и", "с", "и", "п", "п", "и")
# Биздин кортежде "з" же "и" бар экенин текшерүү
print('з' in штат)
print('и' in штат)
```

in ачкыч сөзү бир элементтин кортежде камтылгандыгын текшергенде, логикалык True же Fasle (**Чын же Жалган**) жооп кайтарат. Бул кодду иштеткенибизде, ал төмөнкүдөй жыйынтык чыгарат:

```
False
True
```

Анткени ал алгач кортежде 'з' бар-жогун текшерип, эч нерсе таппайт (False). Андан кийин ал кортежден "и" белгисин текшерип, албетте, бир же бир нечесин табат (True).

Кортеж мисалдары

Азырынча бул бөлүмдө биз кортеждер менен иштөөнүн көптөгөн жолдорун карап чыктык, андыктан ыңгайлуулук үчүн ушул бөлүмдө жазылган, ушул кезге чейин кортеждерге тиешелүү болгон бардык коддорду камтыган Python файлынын үлгүсүн таба аласыз.

Бул кодду өзгөртүүдөн тартынбаңыз жана белгилүү кортеждердеги элементтердин өзгөрүшү коддун үзүндүлөрүнө жана алардын натыйжаларына кандай таасирин тийгизерин карап көрүңүз:

```
# Биздин кортежди аныктоо
кылмышкер = ('Каш көтөрүүчү', 'Ачуулуу Хеклер', 'Анчалык бактылуу эмес адам', 'Каргышка калган Рейзер')
# Кортеждин элементтерин басып чыгаруу
print(кылмышкер)
# Кортеждеги ар бир элементти бирден элементтерди басып чыгаруу
print(кылмышкер[0])
print(кылмышкер[1])
print(кылмышкер[2])
print(кылмышкер[3])

# Сүйлөмдөргө кортеж элементтерин кошуу ыкмалары
print("Биринчи каардуу күнөөкөр ", кылмышкер[0])
print("Экинчи каардуу адам - коркунучтуу " + кылмышкер[1])

# Кесүү 0 индексинен башталып, 3 индексиндеги элементтин алдында бүтөт
print(кылмышкер[0:3])
# Кесүү 1 индексинен башталып, 4 индексиндеги элементтин алдында бүтөт
print(кылмышкер[1:4])
# Менин кочкул кызыл түстөгү баш кийимдеримдин кортежин түзүү
```

```
кызылБашкийим = ('Бүктөлгөн кочкул кызыл түстөгү баш кийим', 'Кыска кочкул кызыл түстөгү баш кийим', 'Тешиктери бар кочкул кызылтүстөгү баш кийим')

# Менин чекиттүү баш кийимдеримдин кортежин түзүү

чекиттүүБашкийим = ('Кара жана Ак чекиттүү баш кийимдерим, 'Ак жана Агыш чекиттүү баш кийимдер', 'Көк чекиттүү баш кийимдер')

# Эки кортежди бириктирип же кошуп жаңы кортеж түзүү

бардыкБашкийимдерим = (кызылБашкийим + чекиттүүБашкийим)

# Жаңы түзүлгөн кортеждин маанилерин басып чыгаруу

print(бардыкБашкийимдерим)

# 1 индексинде көрсөтүлгөн элементти 3 жолу басып чыгаруу

print(бардыкБашкийимдерим [1] * 3)

# Сандар топтомун камтыган кортеж түзүү

эңкичине_маани = (1, 5, 10, 15, 20, 50, 100, 1000)

# Кортеждеги эң кичине маанини кайтаруу үчүн минимум функциясын колдонуңуз

print(min(эңкичине_маани))

# Сандар топтомун камтыган кортеж түзүү

эңчоң_маани = (1, 5, 10, 15, 20, 50, 100, 1000)

# Кортеждеги эң чоң маанини кайтаруу үчүн максимум функциясын колдонуу

print(max(эңчоң_маани))

# Айрым элементтер менен кортеж түзүү

супер_чач = ('Супер Чач Тарач', 'Сакалдуу Мачо', 'Грагантуан Чачынын Ээси',

'Кыжырды Келтирген Парик', 'Такыр Баш')

# Биздин кортеждеги элементтердин санын басып чыгарыңыз

print(len(супер_чач))

# Түрмөдө камалып жаткан жаман адамдардын тизмеси

кылмышкер = ('Укпаган адам', 'Тайгактын Белгиси', 'Мистер Миллениал', 'Джек Хаммер', 'Ызылдаган Аары', 'Сүттүн Баарын Ичкен Адам', 'Керемет-Веджи', 'Качкан Эчки')

# Кортеждеги жаман адамдардын тизмесин иреттелген тартипте басып чыгарыңыз

print(sorted(кылмышкер))

# Биз ирээттей турган сандардын кортежи
```

```
сандар_ирети = (10, 20, 5, 2, 18)
# Биздин кортеждеги сандарды ирээттөө
print(sorted(сандар_ирети))
# Биз кошо турган сандардын кортежи
сандар_суммасы = (10, 20, 5, 2, 18)
# Кортеждеги элементтердин суммасы
print(sum(сандар_суммасы))
# Биз кортежге өткөрө турган тизме
кылмышкерТизмеси = ['Укпаган адам', 'Тайгактын Белгиси', 'Мистер Миллениал', 'Джек Хаммер', 'Ызылдаган Аары', 'Сүттүн Баарын Ичкен Адам', 'Керемет-Веджи', 'Качкан Эчки']
# tuple() колдонуп кылмышкерТизмесин кортежге айландыруу
tuple(кылмышкерТизмеси)
# Кортежге айланта турган сап
кылмышкер1 = "Мурутчан Коркунуч!"
tuple(кылмышкер1)
# Кылмыштуу адамдарга жана баатырларга арналган чач стилдерине толгон кортеж
чач_стили = ('Супер Чач Тарач', 'Сакалдуу Мачо', 'Грагантуан Чачынын Ээси', 'Чачы кыркылган')
# чач_стилин басып чыгаруу
print(чач_стили)

# Кортежди толугу менен жок кылуу үчүн del колдонулат
del чач_стили
# чач_стили эми бош экендигин көрсөтүү үчүн басып чыгаруу
print(чач_стили)

# Миссисисиппи сөзүн жазуу үчүн колдонулган бардык тамгаларды камтыган кортеж
штат = ("М", "и", "с", "с", "и", "с", "и", "с", "и",  "п", "п", "и")
# Эскертүү: Кортежди оңой эле түзсөңүз болот
# tuple() буйругу сапты автоматтык түрдө кортежге айландырат.
# Биздин кортежде канча жолу "и" кездешкенин санап, натыйжасын басып чыгарыңыз
print("Миссисисиппи сөзүндө бир нече 'и' тамгасы бар: ")
print(штат.count('и'))
# Миссисисиппи сөзүндө канча жолу "с" кездешээрин санаңыз
```

```
print("Миссисисиппи сөзүндө бир нече ' с ' тамгасы бар: ")
print(штат.count('с'))

# Миссисисиппи сөзүн жазуу үчүн колдонулган бардык тамгаларды камтыган кортеж
штат = ("М", "и", "с", "с", "и", "с", "и", "с", "и",  "п", "п", "и")
# Биздин кортежде "з" же "и" бар экенин текшерүү
print('з' in state)
print('и' in state)

# Мурда түзүлгөн кылмышкерТизмеси тизмесин карап, ар бир элементти басып чыгаруу
for var in кылмышкерТизмеси:
    print(var)
```

Сөздүктөр менен иштөө

Python *сөздүк* деп аталган дагы бир маалымат структурасына ээ. Сөздүктөр тизмелерден, өзгөрмөлөрдөн жана кортеждерден кыйла кызыктуу жолдор менен айырмаланат. Тизмелерде жана кортеждерде белгилүү бир индексте сакталган маалымат элементтери бар. Демек, ошол шилтеме номерлерине шилтеме берилиши мүмкүн (0 индексинен башталат) болгон сөздүктөр *маппинге* негизделет.

Маппинг - бул Python үчүн маалыматтарды сактоо ыкмасы. Анда Python ачкычтарды маанилерге чагылдырат. Бул *ачкыч жана маани* жубу катары белгилүү.

Ачкычтар ачкыч - *маани* жубунун маанисинин сол жагында аныкталат. Адатта, алардын оң жагындагы *мааниге* байланыштуу сүрөттөйт. *Ачкычтар* өзгөрүлбөйт жана аларды өзгөртүү мүмкүн эмес. Ал эми *маанилер* өзгөрүлүп турат жана алар каалаган маалыматтардын түрүнөн түзүлүшү мүмкүн.

Сөздүктү аныктоо үчүн ага ат коюп, андан кийин сакталган маалыматты эки фигуралуу кашаа {} арасына киргизиңиз:

```
algebro = {'коддун аталышы': 'Algebro', 'күч': 'Математика', 'чыныгы аты-жөнү':'Al. G. Bro.'}
```

Бул учурда algebro сөздүгү карасанатай кара ниеттикти, математиканын чебери Алжеброну билдирет деп айта алабыз! Биздин кара ниет кишилердин маалымат базасы аркылуу, анча-мынча жакшы эмес жергиликтүү шылуундардын бардыгын байкап турабыз. Биздин сөздүктө бир нече маалымат бар. Алардын аталышы, күчү жана чыныгы аты берилет. Бул маалыматтарды сөздүгүбүздө аларды жупташтыра турган маалыматтарга дал келтирүү үчүн ачкычтарыбызды атоо менен чагылдырабыз.

Ошентип, бул мисалда *коддун аталышы* ачкыч болот. Ал эми algebro - ошол ачкычка таандык маани. Алар чогуу биздин *algebro* сөздүгүбүздөгү бир ачкыч-маанилик жупту түзөт.

algebro сөздүгүндөгү башка ачкыч-маани жуптар:

- *күч: математика*
- *чыныгы аты-жөнү: Al. G. Bro*

Эгер сөздүктү басып чыгарууну кааласак, анда төмөнкүлөрдү колдонмокпуз:

```
# algebro деген ат менен сөздүк түзүңүз жана аны ачкыч-маани жуптары менен толтуруңуз
algebro = {'коддун аталышы': 'Algebro', 'күч': 'Математика', 'чыныгы аты-жөнү':'Al. G. Bro.'}
# Algebro сөздүгүн басып чыгарыңыз
print(algebro)
```

Жыйынтыгында:

```
{'коддун аталышы': 'Algebro', 'күч': 'Математика', 'чыныгы аты-жөнү':
'Al. G. Bro.'}
```

Сөздүктөгү ачкыч-маани жуптарын маалымат элементтери деп атоого болот жана иргелбейт. Ошондой эле, элементтерди сиз каалагандай өз алдынча басып чыгарууга болот. Algebro элементинин чыныгы атын билгибиз келди дейли. Сөздүктө белгилүү бир ачкычтын маанисин гана басып чыгаруу үчүн биз мындай деп жазышыбыз керек:

```
print(algebro['чыныгы аты-жөнү'])
```

Python мындай натыйжаны кайтарып берет:

```
Al. G. Bro.
```

Сөздүк методдору

Сөздүктөрдө бир нече методдор бар. Аларды ачкычтар жана маанилер менен иштөө үчүн колдонобуз. Сөздүктө кандай ачкычтар бар экендигин билгибиз келди дейли. Ушуларды гана басып чыгаруу үчүн биз **dict.keys()** методун колдонобуз:

```
# algebro деген ат менен сөздүк түзүңүз жана аны ачкыч-маани жуптары менен толтуруңуз
```

```
algebro = {'коддун аталышы': 'Algebro', 'күч': 'Математика', 'чыныгы аты-жөнү':'Al. G. Bro.'}

# algebro сөздүгүндөгү ачкычтарды гана басып чыгарыңыз

print(algebro.keys())
```

Бул бизге мындай натыйжаны берет:

```
dict_keys(['коддун аталышы', 'күч', 'чыныгы аты-жөнү'])
```

Эгерде биз algebro сөздүгүнүн маанилерин гана билгибиз келсе, **dict.values()** методун колдонмокпуз, мисалы:

```
# algebro сөздүгүндөгү маанилерди гана басып чыгарыңыз

print(algebro.values())
```

Ал бизге муну берет:

```
dict_values(['Algebro', 'Математика', 'Al. G. Bro.'])
```

Бирок ачкычты да, маанилерди да басып чыгаргыбыз келсе, эмне кылышыбыз керек? Бул үчүн **dict.items()** методу деп аталган бир метод бар:

```
# Ачкыч-маани жуптарын басып чыгарыңыз

print(algebro.items())
```

Жыйынтыгы?

```
dict_items([('коддун аталышы', 'Algebro'), ('күч', 'Математика'), ('чыныгы аты-жөнү', 'Al. G. Bro.')])
```

Ушул сыяктуу сөздүк методдорун колдонуу биз маалыматтарды салыштырып же сөздүктө кандай маалыматтар бар экендигин текшерүү керек болгондо абдан керек болот. Ошондой эле ачкычтарыбызды жана аларга байланыштуу маалыматтарды башка сөздүктөр менен да салыштыра алабыз.

Мисалы, algebro сөздүгүнүн терс каарманы бир нече башка сөздүктөрдө пайда болушу мүмкүн. Бирөө анын супер күч-кубаты жана жашыруун жеке маалыматтарын сактаса, дагы бир сөздүктө ал жөнүндө орто мектептеги маалыматтар жана дене тарбия сабагынан алган баалары камтылышы мүмкүн. (Мага ишенип койуңуз, Математик Algebro орто мектепте окуганда спорттон *аябай начар болгон*!).

Акыр-аягы, сөздүктө маалымат элементтерин басып чыгаруунун дагы бир жолу бар. Биз ар бир кайталоодо маалыматты басып чыгарып, сөздүктү жөн эле

кайталап (же цикл) жасай алабыз. **for loop** эсиңиздеби? Бул жерде ал ыңгайлуу болот:

```
# Сөздүгүбүздү кайталоо жана басып чыгаруу үчүн for циклин колдонуу
for key, value in algebro.items():
print("Ачкыч: ", key, "жана анын мааниси: ", value)
```

Бул пайдалуу коддун үзүндүсү кыйла ылайыктуу натыйжага алып келет:

```
Ачкыч: коддун аталышы  жана анын мааниси:  Algebro
Ачкыч: күч  жана анын мааниси:  Математика
Ачкыч: чыныгы аты-жөнү  жана анын мааниси:  Al. G. Bro.
```

Сөздүктөр менен иштөө мындан да көңүлдүүрөк

Кортеждерден айырмаланып, сөздуктүн маанисин өзгөртсө болот, бирок ачкычты өзгөртө албайбыз. Айталы, биз Algebro каармаңыздын жашына *жаш* кошкубуз келди дейли. Ал үчүн жөн гана төмөнкүдөй кодду колдонсок болот:

```
# algebro деген ат менен сөздүк түзүңүз жана аны ачкыч-маани жуптары менен толтуруңуз
algebro = {'коддун аталышы': 'Algebro', 'күч': 'Математика', 'чыныгы аты-жөнү':
'Al. G. Bro.'}

# 'algebro'  сөздүгүнө  'жаш' ачкычын кошуп, ага 42 деген маанини бериңиз
algebro['жаш'] = 42'

# Жаңы кошулган ачкыч-маани жупту көрсөтүү үчүн algebrону басып чыгарыңыз
print(algebro)
```

Бул кодду иштетсек, биз төмөнкүдөй жыйынтыкка жетишебиз:

```
{'коддун аталышы': 'Algebro', 'күч': 'Математика', 'чыныгы аты-жөнү':
'Al. G. Bro.', 'жаш': '42'}
```

Бул жерден биз "жаш" жана "42" жаңы ачкыч-маани жуп кошулгандыгын көрө алабыз.

Бул жерде сиз байкай турган көйгөй жаштын туруктуу сан эмес экендиги болуп саналат. Башкача айтканда, ал убакыттын өтүшү менен өзгөрүп турат. Биздин

каардуу Algebro каармanынын туулган күнү болгон сайын биз ушул ачкыч менен маани жубун жаңыртып турушубуз керек.

Эч кандай тынчсызданууунун кереги жок, анткени ачкычтын маанисин өзгөртүү, жаңы маани кошкондой эле жөпжөнөкөй:

```
# Биздин "жаш " ачкычынын маанисин жаңыртуу
algebro['жаш'] = 43

# 'жаш' ачкычынын жаңыланган маанисин көрүү үчүн algebro сөздүгүн басып чыгаруу
print(algebro)
```

Эми биз жаштын маанисин басып чыгара турган болсок, анда ал 43кө барабар болмок.

Сөздүктүн маанисин жаңыртуунун дагы бир жолу - dict.update() методун колдонуу. Мисалы, биз *кылмышкерТүрү* деп аталган жаңы ачкычты кошуп, dict.update() методун колдонуп, ага *мутант* маанисин бере алабыз.

Мисалы:

```
# algebro деген ат менен сөздүк түзүңүз жана аны ачкыч-маани жуптары менен толтуруңуз
algebro = {'коддун аталышы': 'Algebro', 'күч': 'Матемагика', 'чыныгы аты-жөнү':'Al. G. Bro.'}
# Биздин "algebro" сөздүгүбүзгө ачкыч-жуп маанисин кошуу үчүн #dict.update() колдонулат
# Фигуралык кашаа {} менен кашаанын () аралашып колдонулушуна көңүл #буруңуз
 algebro.update({'кылмышкерТүрү': 'мутант'})
# Жыйынтыктарын басып чыгаруу
print(algebro)
```

Эми бул кодду иштетсеңиз, төмөнкүдөй жыйынтык чыгышы керек:

```
{'коддун аталышы': 'Algebro', 'күч': 'Матемагика', 'чыныгы аты-жөнү':
'Al. G. Bro.', 'age': 43, 'кылмышкерТүрү': 'мутант'}
```

кылмышкерТүрү - мутант ачкыч-маани жубунун кошулушуна көңүл буруңуз. Ушул ыкманы, ошол эле кодду колдонуп, сөздүктөгү ачкыч-мааниси бар жуптарды жаңырта аласыз.

Буга чейин көргөн **del** ачкыч сөзүн колдонуп, сөздүктөн ачкыч-маани жубун алып салабыз. Мисалы, кандайдыр бир себептерден улам, Algebro өзүнүн супер

күчүн жоготуп алса, анда биз төмөнкүдөй ачкыч жуп түгөйүн толугу менен жок кыла алабыз:

```
# Ачкыч-маани жубун жок кылуу үчүн del ачкыч сөзүн колдонуу
del algebro['күч']

# Ачкыч-маани жубун туура алып салгандыгыбызды текшерүү үчүн algebrону басып чыгарыңыз.
print(algebro)
```

Бул бизге төмөндөгүнү берет:

```
{'коддун аталышы': 'Algebro', 'күч': 'Математика', 'чыныгы аты-жөнү':
'Al. G. Bro.', 'жаш': 43, 'кылмышкерТүрү ': 'мутант'}
```

Биз негизги күчтү жана ага байланыштуу маанини чындап эле ийгиликтүү жок кылгандыгыбызды текшеребиз.

Андан тышкары, эгер биз сөздүктү толугу менен жок кылгыбыз келсе, анда дагы **del** колдонсок болот:

```
# del ачкыч сөзүн колдонуп, algebro сөздүгүн жок кылуу
del algebro
# Жок кылынган algebrону басып чыгаруу, натыйжада, ката деген билдирүү келет

# Бул algebro жок болгондуктан пайда болот
print(algebro)
```

Эгер сиз ошол кодду иштетсеңиз, анда ката жөнүндө билдирүү аласыз, анткени сиз буга чейин жок кылган algebro сөздүгүн басып чыгарууга аракет кылып жатасыз:

```
Traceback (most recent call last):
File "C:/Users/ ... /Python/Python36-32/
DictionaryExamples.py", line 58, in <module>
print(algebro)
NameError: name 'algebro ' is not defined
```

Кээде сөздүктөгү бардык элементтерди же ачкыч-маани жуптарын өчүргүңүз келип, бирок сөздүктүн өзүн *жок кылбай* турган учур келиши мүмкүн. Бул үчүн **dict.clear()** ыкмасын колдонобуз:

```
# algebro деген ат менен сөздүк түзүңүз жана аны ачкыч-маани жуптары менен толтуруңуз
algebro = {'коддун аталышы': 'Algebro', 'күч': 'Математика', 'чыныгы аты-жөнү':'Al. G. Bro.'}
algebro.clear()
print(algebro)
```

Эгер сиз ушул коддун үзүндүсүн иштетсеңиз, анда төмөнкүдөй жыйынтык чыкмак:

```
{}
```

Негизинен, бош сөздүк деп атоого болот, же болбосо, бир эле натыйжага жетиш үчүн: algebra = {} деп терип жазсаңыз болот.

Башка сөздүк методдору

Сиздин колуңузда 26 сөздүк ыкмасы бар. Бул китепте алардын бардыгын чагылдырууга орун жок. Бирок, мен аларды өз алдынча изилдөө жүргүзүүгө чакырам. Алар менен тажрыйба жүргүзүп, күчүңүздү акылдуулук менен пайдаланыңыз!

Ушул ыкмалардын айрымдарын сиз буга чейин тизмелерде жана кортеждерде колдонгонсуз. Алардын катарына sum(), min(), max(), sorted(), ж.б.у.с. кирет.

Бул жерде сөздүктөрдүн башка методдорунун тизмеси берилген: Алга, тайманбаңыз, кеңири тажрыйба алыңыз!

- dict.clear(): Сөздүктөгү бардык элементтерди алып салат;
- dict.copy(): Сөздүктүн көчүрмөсүн кайтарып берет;
- dict.fromkeys(): Сөздүктү ырааттуу түзүү үчүн колдонулат;
- dict.get(): Көрсөтүлгөн ачкычтын маанисин кайтарып берет;
- dict.items(): Берилген сөздүктүн ачкыч / жуп маанилеринин көрүнүшүн кайтарып берет;
- dict.keys(): Сөздүктөгү ачкычтарды кайтарып берет;
- dict.popitem(): Сөздүктүн элементин кайтарып берет жана алып салат;
- dict.pop(): Белгиленген ачкычтагы элементти кайтарып берет жана алып салат;

- dict.setdefault(): Ачкычтын бар-жогун текшерип, эгер жок болсо, ачкычты киргизет (мааниси менен);
- dict.values(): Сөздүктөгү бардык маанилерди кайтарып берет;
- dict.update(): Сөздүктү жаңыртуу үчүн колдонулат;

Сөздүктө колдоно турган башка ыкмаларга төмөнкүлөр кирет:

- any(): Элементтин Туура экендигин текшерет;
- all():Эгер кайталоонун бардык элементтери Туура болсо, ***True*** кайтарылат;
- dict(): Сөздүк түзүү үчүн колдонулат;
- enumerate(): Эсептелүүчү объектти жаратат же кайтарат;
- iter(): Берилген объект үчүн итераторду кайтарат;
- len(): Объекттин узундугун кайтарып берет;
- max(): Эң чоң элементти кайтарып берет;
- min(): Эң кичинекей элементти кайтарып берет;
- sorted(): Сорттолгон тизмени кайтарып берет;
- sum(): Бардык элементтердин суммасын берет .

Сөздүк кодунун мисалы

Мында бул бөлүмдөгү бардык код файлдарынын үлгүсү берилген. Ар кандай өзгөртүүлөрдү киргизип, код менен (каалагандай) тажрыйба жүргүзүп көрүңүз. Кандайдыр бир каталарды байкап, ушул убакка чейин китептен алган билимиңизди колдонуп, аны кызыктуу жолдор менен өзгөртүүгө аракет кылыңыз.

Эсиңизде болсун, көңүл ачып, тобокелчил болуңуз (андан башка супер баатыр дагы эмне кылышы мүмкүн?):

```
# algebro сөздүгүн түзүп, ачкыч-маанилик жуп менен толтуруңуз
algebro = {'коддун аталышы': 'Algebro', 'күч': 'Математика', 'чыныгы аты-жөнү':'Al. G. Bro.'}
# algebro сөздүгүн басып чыгарыңыз
print(algebro)

# Чыныгы аты-жөнүнүн ачкычын гана басып чыгарыңыз
print(algebro['чыныгы аты-жөнү'])

# algebro сөздүгүндөгү ачкычтарды гана басып чыгарыңыз
print(algebro.keys())
```

```
# algebro сөздүгүндөгү маанилерди гана басып чыгарыңыз
print(algebro.values())

# Ачкыч-маани жуптарын басып чыгарыңыз
print(algebro.items())

# Сөздүгүбүздү кайталоо жана басып чыгаруу үчүн for циклин колдонуу
for key, value in algebro.items():
     print("Ачкыч: ", key, "жана анын мааниси: ", value)

# algebro сөздүгүнө 'жаш' ачкычын кошуп, ага '42' маанисин бериңиз
algebro['жаш'] = '42'

# Жаңы кошулган ачкыч-маани жубун көрсөтүү үчүн басып чыгарыңыз
print(algebro)

# Биздин "жаш" ачкычыбыздын маанисин жаңыртуу
algebro['жаш'] = 43

# 'жаш' ачкычынын жаңыланган маанисин көрүү үчүн algebro сөздүгүн басып
#чыгаруу
print(algebro)

# Биздин "algebro" сөздүгүбүзгө ачкыч – маани жубун кошуу үчүн
#dict.update() колдонуу
# Кашаанын() ичине аралаштырылган фигуралуу кашаанын {} колдонулушуна
#көңүл буруңуз
algebro.update({'кылмышкерТүрү': 'мутант'})

# Жыйынтыктарын басып чыгаруу
print(algebro)
# Ачкыч-маани жубун жок кылуу үчүн del ачкыч сөзүн колдонуу
del algebro['күч']

# Ачкыч-маани жубун туура алып салгандыгыбызды текшерүү үчүн algebrону
#басып чыгаруу
print(algebro)
##########################################################
# Коддун ушул бөлүмүнө комментарий берилген. Анткени ал бардык каталарга
#себеп болот
# del ачкыч сөзүн колдонуп, algebro сөздүгүн жок кылуу
# del algebra
```

```
# Жок кылынган algebro сөздүгүн басып чыгаруу, натыйжада, катага алып
#келет

# Бул algebro сөздүгү жок болгондуктан пайда болот

# print(algebro)

##############################################

# algebro деген ат менен сөздүк түзүңүз жана аны ачкыч-маани жуптары
менен толтуруңуз

algebro = {'коддун аталышы': 'Algebro', 'күч': 'Матемагика', 'чыныгы аты-
жөнү':

'Al. G. Bro.'}

algebro.clear()

print(algebro)
```

Бул эпизоддо

Ушул жерге жеткениңиз үчүн өзүңүз менен сыймыктануунуз керек! Бул эпизоддо эс-тутумуңузга эки жаңы маалымат структурасын - кортеж жана сөздүк, кошуу менен мээңиздин эс-тутумун кеңейттик!

Биз демейдегидей көп нерсени камтыдык. Демек, адаттагыдай эле, ушул бөлүмдөн алган билимдеribиздин көпчүлүгүн сүйктүү тизмебизге топтоо ар дайым жакшы идея болуп саналат. Ошентип, эмне деп ойлойсуз? Анда эмесе кеттик:

- Кортеждер жана сөздүктөр - бул өзгөрмөлөр жана тизмелер менен катар маалыматты камтыган маалымат структурасынын кошумча эки формасы;
- Кортеждер тизмелерге окшош, бир айырмасы кортеждер туруктуу, башкача айтканда, алардын маанилерин алмаштыруу же өзгөртүү мүмкүн эмес;
- Кортеждер мындай жол менен аныкталат;

  кылмышкер = ('Каш көтөрүүчү', 'Ачуулуу Хеклер', 'Анчалык бактылуу эмес адам', 'Каргышка калган Рейзер')

- Биз print(кылмышкер) кодун колдонуп кортежди басып чыгара алабыз;
- Биз элементти print (кылмышкер[0]) кодун колдонуп кортежди басып чыгарабыз, бул биздин кортеждеги биринчи элементти - же 0 - индексте катталган элементти - басып чыгарат;
- Элементтин диапазонун кортежге басып чыгаруу үчүн биз print (кылмышкер[0: 3]) колдонобуз; ал элементтерди 0, 1 жана 2 индекстеринде басып чыгарат; экинчи параметрде жайгашкан элементке чейин басып чыгарууну бүтүрөт (бул учурда, 3);
- Кортеж функциялары min(), max(), len(), sorted(), sum(), жана tuple() функцияларын камтыйт;

- del ачкыч сөзүн толугу менен кортежди жок кылуу үчүн колдонсо болот;
- count() кортежде элементер болгон учурда, алардын санын эсептейт;
- Биз кортеждин ичинде бир нерсе пайда болгонун текшерүү үчүн **in** колдонсок болот; ал логикалык True (**Чын)** же False (**Жалган)** маанисин кайтарып берет;
- Маалыматтарды сактоо үчүн сөздүктөр маппингди колдонушат;
- Сөздүктөрдө ачкыч-маани жубу же ачкыч-маани жуптар тобу бар, алар элементтер же маалымат элементтери катары белгилүү;
- Ачкычтар кош чекиттин сол жагында, ал эми маанилери оң жагында аныкталат;
- Сөздүктөр мындайча аныкталат:

```
  algebro = {'коддун аталышы': 'Algebro', 'күч': 'Математика',
'чыныгы аты-жөнү':
  'Al. G. Bro.'};
```

- print(algrebro) кодун колдонуп сөздүктү басып чыгара аласыз;
- print(algebro['чыныгы аты-жөнү']) кодун колдонуп белгилүү бир ачкычтын маанисин басып чыгара аласыз;
- Сөздүк методдору: dict.keys(), dict.items(), жана dict.update();
- del ачкыч сөзүн сөздүктү жок кылуу үчүн да колдонсо болот;
- dict.clear() чыныгы сөздүктү жок кылбай туруп, элементтерди сөздүктөн жок кылууга мүмкүндүк берет (жөн гана анын ачкычтарын жана маанилерин).

10-БӨЛҮМ

PYTHON ФАЙЛДАРЫ

Азырынча биз негизинен бир файлдын ичинде иштедик. Башкача айтканда, биздин бардык коддор бир гана .py файлында сакталган жана ошол эле файлдан иштетилген. Бирок, күндөлүк жашоодо, биздин көптөгөн программалар бир нече файлдарда сакталат. Андан тышкары, кээ бир сүйүктүү код үзүндүлөрүбүздү жана функцияларыбызды кийинчерээк колдонуу үчүн файлдарда сактасак болот. Биз - программисттер, алардын ичинде азыр сиз да барсыз, ушундай иштейбиз.

Коддун бир нече файлын колдонуунун көптөгөн себептери бар. Алардын айрымдары биздин коддогу натыйжалуулукту көбөйтүүнү жана каталарды азайтууну көздөйт. Жалпы милдеттерди аткарган программалардын бөлүктөрүн кийинчерээк башка программаларда кайра колдонуу үчүн сактап койгонубуз эсиңиздеби? Функциялар жана модулдар жөнүндө сөз болгондо биз аны терең талкуулаганбыз.

Ошондой эле, класстарды жана объектилерди, өзгөрмөлөрдү, маалыматтардын тизмесин жана биз ойлогон жалпы колдонулган коддордун дээрлик бардык түрүн сактай алабыз. Негизинен, программадан бир нерсени кийинчерээк колдоном деп ойлосоңуз жана убакытты үнөмдөп, колдонуучунун катасын азайтууга мүмкүнчүлүк бергиңиз келсе (б.а., кылмыштуулук менен күрөшүүдөн чарчаганыңызда кодду жазып жатканда), анда өзүңүзгө чоң жакшылык кылыңыз. Анын көчүрмөсүн кийинчерээк колдонуу үчүн өзүнчө файлга көчүрүп алыңыз.

Ооба, жана аны жакшылап документтештирип алыңыз, ошондо ал укмуштуу кодду эмнеге сактап койгонуңузду билип аласыз!

Кодду бир нече файлдардан колдонуунун дагы бир себеби - бул көбүнчө ири долбоорлорго тиешелүү - биз программанын үстүндө иштеген жалгыз кодер эмеспиз. Биз жалпы колдонмонун кичинекей бөлүгүн гана иштетип жатабыз. Ушул себептен улам, сиз көп сандаган файлдар менен алектенишиңиз мүмкүн.

Мисалы, эгер сиз супер баатырдын ролдук оюнунун кодун жазып жатсаңыз, анда сиз долбоорду толугу менен башкара аласыз. Сиздин досуңуз Пол Кодерман коддун согушка байланыштуу бөлүгүнө жооптуу болушу мүмкүн. Сиздин башка досуңуз - Ральф Программердудесон каармандарды жаратуу менен иштеши мүмкүн. Сиздин офистеги душманыңыз (ар бир адамга алардын бири керек) бурчта отуруп алып, эртеден-кечке сизди ачуулуу тиктеп, толгон токой зыяндуу тамактарды жей бериши мүмкүн.

Программаны бириктирүү үчүн Пол Кодермандын файлдарынан көптөгөн функцияларды чакырып, Ральф Программердудесондун кодго толгон папкасынан каармандарды жаратуу курамын киргизе аласыз. Акыр-аягы сиздин душманыңыз ачууланууу дозасын кошот. Айкалыштыра келгенде сизде ийгиликтүү ролдук оюн үчүн керектүү элементтердин бардыгы болот.

Python файлдары менен иштөө

Эгер сиз китептин ушул жерине чейин окуп жетишкен болсоңуз, анда файл тутуму эмне экендигин жакшы билесиз деп ойлоймун. Эгер андай болбосо, компьютериңиздин жумушчу столундагы документтерди, комикстерди, видео оюндарды жана көптөгөн селфилерди камтыган кичинекей папкалар жөнүндө ойлонуп көрсөңүз болот.

Алгач Python орнотуп жатканда аны демейки папкага орнотууга уруксат бергенбиз. Компьютериңизге, операциялык тутумуңузга жана дисктериңизди кантип орнотконуңузга жараша, сизде меникине окшош нерсе бардыр. Мисалы, менин Python нускам жана IDLE төмөнкү дарекке орнотулган:

C:\Users\James\AppData\Local\Programs\Python\Python36-32

Сиздики бир аз башкача болушу мүмкүн. Мисалы:

C:\Users\СиздинАтыңыз\Programs\Python

жана ушу сыяктуу.

Баса, IDLE менен түзүлгөн бардык .py же Python файлдары ушул жерге автоматтык түрдө сакталат. Чындыгында, мен программаны иштеткенде ал алгач файлдарды издеп, ошол папканы карайт. Эгерде мен түзгөн Python программасынан башка файлды чакырышым керек болсо, ал автоматтык түрдө ошол папканы издеп, ушул жерден табууга болот деп кайрылат.

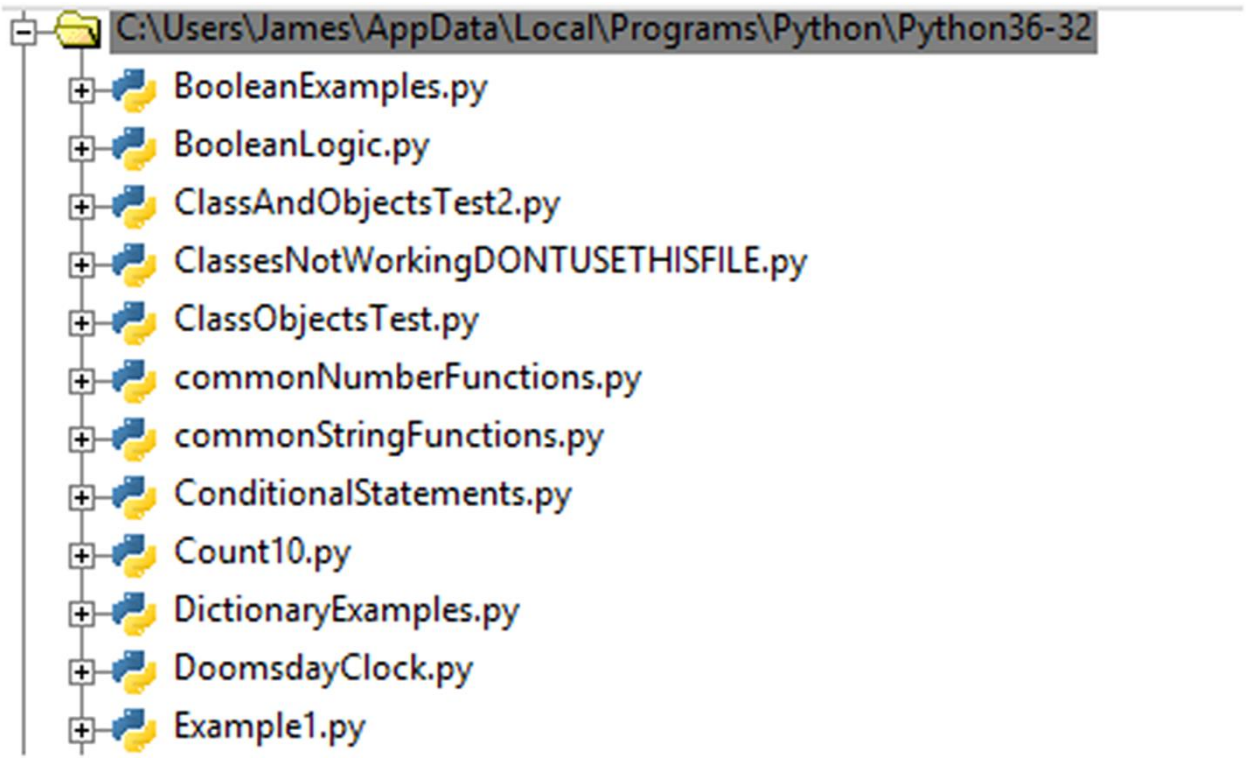
Python каталог папкасынын мисалында ушул китеп үчүн ушул кезге чейин жазган бардык файлдарым көрсөтүлгөн (10-1-сүрөттү караңыз):

Бул файлдар биздин негизги каталог папкасында болгондуктан, аларды Python программалардын бирине чакырганда каталогдорду алмаштыруу же башка папкаларды издөө сыяктуу өзгөчө бир нерсе жасоонун кажети жок. Болгону, мен программада файлдын атын атап, аны импорттойм. Бул аябай жеңил.

Күтө тур, мен жакында кайтып келем, мен сыр жеп келейин.

Мына, мен кайтып келдим.

Күндөлүк жашоодо ал дайыма эле жөнөкөй боло бербейт. Адатта, адашып калбашыбыз үчүн, же кокустан туура эмес файлды чакырып албаш үчүн ар бир программа боюнча программабыздын файлдарын белгилүү папкаларда сактайбыз. Элестетиңиз, бир жыл ичинде бир топ файлдарды топтосоңуз болот жана сөзсүз түрдө жумушту уюштуруунун жолу керек болот.



10-1-сүрөт. *Python каталог папкасынын мисалы*

Мисалы, сиз ролдук оюндардын үстүндө иштеп жаткан болсоңуз, анда РолдукОюндарБаатыры деп аталган негизги папкаңыз болушу мүмкүн. Андан кийин ошол папканын ичинде оюндун ар бир бөлүмү үчүн файлдарды камтыган папкалар топтому болот. Бул папканын түзүлүшүн карап көрөлү. Мисалы:

супербаатыр RPG;

- КаармандардаТүзүүМодулу;
- СогушМашинасы;
- ШумпайдынПрофили;
- ЖерФайлы;
- КапыстанЖолугушууФункциялары.
- ЭлементТизмелери;
- СуперКүчтөрдүнТизмеси;
- ШумпайларСөздүктөрү;
- БаатырКласстары;
- Мутанттар;
- Адамдар;

- Робот;
- Сыйкырчылар;
- ШакирттерПрофайлы жана дагы ушул сыяктуулар

Бул папкалардын ар биринде программанын ар бир бөлүгүнүн функцияларын аткарган программаңыздын бөлүктөрү болот. Мисалы, СогушМашинасы папкасында согуш сценарийлерин, зыяндын натыйжаларын ж.б.у.с. жооптуу функциялар жана код камтылат.

Бул файлдардын бардыгы түпкү папкадан *тышкары* сактала тургандыктан, файлды ушул программанын бир бөлүгү жайгашкан каталогдон биздин негизги программабызга чакырышыбыз керек.

Эгер учурда түшүнүксүз болуп калса, анда эч нерсе болбойт. Бул бөлүмдө кайсы жерде жайгашкандыгына карабастан, башка программанын ичинен программаны кантип чакыруу керектиги жөнүндө кеңири маалымат беребиз.

Эми сиз папканын түзүлүшү жана Python файлдарыңыз ар кайсы жерлерде сактала тургандыгы жөнүндө негизги түшүнүк менен таанышкандан кийин калганы оңой-олтоң болот.

Анда эмесе, бир кесим торт жеп келейин.

Файлдын түрлөрү

Ушул убакытка чейин биз .py файлдары менен гана иштеп келгенбиз. Чындыгында, биз көпчүлүк программисттер жасаган нерсени тексттик же .txt файлдарда жаза алабыз. Бул Windows Notepad сыяктуу программаларга же Notepad ++ деп аталган дагы бир таасирдүү текст редакторуна таянат.

Программалоону кеңейтип, өз программаңызды иштеп чыксаңыз же ишканада иштей баштасаңыз, анда файлдардын башка түрлөрүн да колдоно баштайсыз. Алардын ичинен эң кеңири тараган .txt файлдары: гипертекст белгилөө тили (HTML) (веб-баракчаларды иштеп чыгуу үчүн колдонулат) жана үтүр менен ажыратылган (CSV) файлдар - электрондук таблицанын маалыматтары.

Албетте, сиз дагы C же C ++ файлдары жана JSON сыяктуу башка тилдик файлдар менен иштейсиз. Бул Python колодонууну кеңейтүү деп аталат жана 13-бөлүмдө кыскача баяндала турган тема.

Бул бөлүмдөгү мисалдарда биз негизинен .txt жана .py файлдары менен иштейбиз. Бирок теориянын көпчүлүгү бардык жерде бирдей иштейт.

Python кодунда тексттик файлды түзүү

Бул китептин кийинки бөлүгүнө кайрылуунун бир нече жолу бар. Жаңы баштап жаткандар үчүн Notepad же тексттик редактор программасын ачып, жаңы тексттик файлды түзүп, андан кийин башка (учурда) .py жана Python программалоо файлдары сакталган каталогго сактасак болот. Бирок бул бир аз жай көрүнөт.

Тескерисинче, башкача ыкманы колдонуп көрөлү. Кээ бир Python коддорун колдонуп, жаңы текст файлын түзөлү.

Бул бөлүмдөн бир нече жаңы түшүнүктөрдү үйрөнгөнү жатабыз. Андыктан кээ бир нерселерди дароо түшүнбөсөңүз, көп тынчсызданбаңыз. Негизги түшүнүктөрдү алгандан кийин биз баарын кылдаттык менен чагылдырабыз. Мындан тышкары, сөзсүз түрдө комментарийленген кодду окуп чыгыңыз. Адаттагыдай эле, мындан ар бир сап эмнени билдирерин билсеңиз болот.

Эсиңизде болсун! Бул программаны үйрөтүүдөгү биздин максатыбыз – Python программалоо тилин колдонуп, жаңы текст файлын түзүү. Булар биз жараткан .py же Python файлдарынан айырмаланып турат. Бул жерде бир айла бар. Биз Python файлынын ичине тексттик файл түзөбүз. Андыктан биз кайсы файлдын үстүнөн иштеп жаткандыгыбызды билбей башыңызды оорутпаңыз.

Алгач FunWithFiles.py деген жаңы Python файлын түзүшүбүз керек. Ага төмөнкү кодду кошуңуз:

```
# Бул код файл ачуу үчүн колдонулат
# Бирок, файл мурунтан жок болгондуктан
#Анын ордуна  Python аны биз үчүн түзөт

жаңыФайл = open("ТүзүлгөнФайл.txt", 'w')

# Бул код print() операторуна окшош
# Бирок, колдонуучунун компьютеринин экранына текст жазуунун же
#чыгаруунун ордуна

# Аны файлга жазат
жаңыФайл.write("Мында биздин файл!")

# Close() функциясы биз иштеп жаткан файлды жаап, сактайт
# Файл менен иштөө аяктагандан кийин аны ар дайым жабуу маанилүү
# Эч кандай толуктоолорду киргизбешибиз же файлды бузуп албашыбыз үчүн

жаңыФайл.close()
```

Бул коддо бир нече нерсени белгилей кетүү керек. Биринчиден, *ТүзүлгөнФайл*.txt деген аталыштагы жаңы текст же .txt файлын түзүү үчүн код колдонууну көздөп жатабыз. Биз *жаңыФайл* деген өзгөрмө жасап, ага open() функциясын колдонуудан баштайбыз. Адатта open() функциясын колдонуп, мурда сакталган файлды ачып, ага кандайдыр бир өзгөртүүлөдү киргизип же аны колдоно алабыз. Бирок, мисалы, *ТүзүлгөнФайл*.txt деген файл жок болгондуктан, Python андай файл таба албады жана биздин жаңы .txt файлын түзүүнү көздөп жатканыбыздай түшүнүк менен жаңы файл түздү. Ушундай аталыштагы файл болгон күндө, ал мурунку файлдын үстүнөн жазып, ичин бош калтырып коёрун эске алыңыз. Андыктан ушул ыкманы колдонууда этият болуңуз!

Сапта:

```
open("ТүзүлгөнФайл.txt", 'w')
```

"ТүзүлгөнФайл.txt" - бул биз ачууну / түзүүнү каалаган файлдын аталышы. 'w' бөлүгү *аргумент* катары белгилүү жана ал - open() функциясын колдоно турган бир нече бөлүктүн бири.

Бул учурда, 'w' Python сиз файлды *жазуу* үчүн ачууну каалаганыңызды билдирет. Ошондуктан, Python файлды *жазуу режиминде* ачат. Бул режим бизге өзгөртүүлөрдү киргизүүгө же файлга башка нерселерди кошууга мүмкүндүк берет.

Биз жаңы файл түзүп, ага жазуу үчүн 'x' режимин да колдонсок болмок. Бирок, ал файлды эксклюзивдүү түрдө жаратат. Эгерде файлдын аталышы мурунтан эле бар болсо, ал иштебей калат же ката кетирет. Муну колдонуу үчүн биз жөн гана кодду өзгөртөбүз:

```
open("ТүзүлгөнФайл.txt", 'x')
```

Биздин жаңы түзүлгөн файлыбызга бир нерсе кошкубуз келди. Албетте, кошпой койсок деле болот. Биз аны жөн эле калтырып койсок болмок. Бирок, биз аны ачып жатканда ага бир нерсе салып койгонбуз.

Саптагы .**write** методу:

```
жаңыФайл.write("Мында биздин файл!")
```

Жаңы түзүлгөн *ТүзүлгөнФайл*.txt файлы `Мында биздин файл!`"[2] текстин сактоо же жазуу үчүн колдонулган.

Аягында, анын бузулбашы, өзгөрүлбөшү же биз ойлобогон кандайдыр бир таасирлерге кабылбашы үчүн биз ар дайым ачкан же түзгөн файлды жаап салууну каалайбыз. Ал үчүн биз төмөнкүдөй **close**() функциясын колдондук:

```
жаңыФайл.close()
```

Python файлдарын окуу

Файлдарды түзүүдөн жана аларга жазуудан тышкары, аларды окуй да алабыз. Эми *ТүзүлгөнФайл*.txt деген жаңы файлды түзүп алгандан кийин аны окуй турган программа түзөлү. FunWithFiles.py файлыңызга төмөнкү кодду кошуңуз:

```
# ТүзүлгөнФайл.txt файлын ачыңыз
окуу = open("ТүзүлгөнФайл.txt", 'r')
```

[2]Китепте маанини толук берүү максатында, текст кыргыз тилине которулду. Бул текстти кыргыз тилинде жазгандыгыбыз үчүн, *UnicodeEncodeError: 'charmap' codec can't encode character* катасын берип калышы мүмкүн. Эгер ката берилсе, тексттеги айрым тамгаларды (ң, ү ж.б.) өзгөртүп же текстти латын тамгалары менен жазып, сактап, программаңызды иштетиңиз.

```
# Файлдын мазмунун окуп чыгыңыз
print(окуу.read())
```

Бул жерде, мурун түзүлгөн файлды ачуу үчүн open() колдонуп, аны 'r' ачкычына өткөрүп бердик же аргумент / параметрди окудук. Андан кийин, print() функциясын .read методун колдонуп текстти экранга чыгарып, файлдын мазмунун көрө алабыз.

Файлда бир сап текст болгондо бул жакшы иштейт. Бирок бизде бир нече сап болсочу?

FunWithFiles.py файлындагы кодду ушул мисалга дал келгидей кылып өзгөртүңүз:

```
# Бул код файлды ачуу үчүн колдонулат.
# Бирок, файл азырынча жок.
# Анын ордуна Python аны биз үчүн жаратат.
# Файлдын аты мурунтан эле бар болсо, программа анын үстүнө жазып, ичиндеги мурдагы материалдарды өчүрүп саларын унутпаңыз.
жаңыФайл = open("ТүзүлгөнФайл.txt", 'w')
# Бул код print() операторуна окшош
# Бирок, колдонуучунун компьютеринин экранына текст жазуунун же чыгаруунун ордуна
#   Аны файлга жазат
жаңыФайл.write("Мында биздин файл!")  жаңыФайл.write("Мында тексттин экинчи сабы!")
# Close () функциясы биз иштеп жаткан файлды жаап, сактайт
# Файл үстүндө иштөө аяктагандан кийин аны ар дайым жабуу маанилүү
# Эч кандай толуктоолорду киргизбешибиз же файлды бузуп албашыбыз үчүн
жаңыФайл.close()
# ТүзүлгөнФайл.txt файлын ачыңыз
окуу = open("ТүзүлгөнФайл.txt", 'r')
# Файлдын мазмунун окуп чыгыңыз
print(окуу.read())
```

Биздин кодго ушул сапты гана коштук:

```
жаңыФайл.write("Мында тексттин экинчи сабы!")
```

Файлды иштеткенде биз тексттин эки сабын көрөбүз деп күтөбүз. Мисалы*:*

```
Мында биздин файл!
Мында тексттин экинчи сабы!
```

Бирок, андай эмес. Тескерисинче, биз мындай натыйжа алабыз*:*

```
Мында биздин файл!Мында тексттин экинчи сабы!
--------------------------------------------------------------------------------------------------------------------
```

Бирок эмне үчүн андай?

Эки жооп бар, экөөнү тең талкуулайбыз. Биринчиден, жаңы түзүлгөн файлга алгач текст жазганда, ага эч кандай формат берген эмеспиз, .write тексттин аягында кайтуу кареткасын же жаңы сап белгисин кабыл албайт (ал сиз сүйлөмдү тергенден кийин Enter баскычын басканыңызга барабар).

Ошондуктан, биздин саптар бириктирилбеши үчүн биз тексттин аягына \n жаңы сап кошуп алышыбыз керек. Негизинен, эки .write билдирүүсүн төмөнкүдөй өзгөртүүгө болот:

```
жаңыФайл.write("Мында биздин файл!\n")
жаңыФайл.write("Мында тексттин экинчи сабы!\n")
```

Кана анда, FunWithFiles.py файлыңызды ошол өзгөрүүлөргө дал келгидей кылып өзгөртүңүз. Эми программаны дагы бир жолу иштетип көрүңүз. Бул жолу сизде төмөндөгүдөй натыйжа болушу керек.

```
Мында биздин файл!
Мында тексттин экинчи сабы!
--------------------------------------------------------------------------------------------------------------------
```

readline() жана readlines()

Тексттик файлда белгилүү бир сапты же конкреттүү бир нече гана сапты окууну каалаган учурлар болот. .read методу файлдын мазмунун толугу менен окуйт. Андыктан ал бул сценарийде иштебей калат. Анын ордуна readlines() модулун колдонушубуз керек.

Муну иш жүзүндө көрүү үчүн, кодубузду өзгөртүп, алмаштыралы:

print(окуу.read())

кодун мындай жазалы:

print(окуу.readline())

Эми программаны иштеткенде, натыйжаңыз төмөнкүчө болот:

Караңыз, биз Python кодун колдонуу менен жаңы бренд файлын түздүк!

Себеби readline() тексттин саптарын бир эле учурда окуйт. Файлдагы тексттин кийинки сабын окуу үчүн readline() дагы бир нускасын кошсоңуз болот. Уланта бериңиз, FunWithFiles.py учурдагы көчүрмөсү ушул код менен дал келгенин текшериңиз:

```
# Бул код файлды ачуу үчүн колдонулат
# Бирок, файл азырынча жок болгондуктан
#Анын ордуна Python аны биз үчүн түзөт
жаңыФайл = open("ТүзүлгөнФайл.txt", 'w')
# Бул код print() операторуна окшош
# Бирок, текст жазуунун же колдонуучунун компьютер экранында көрсөтүүнүн ордуна
#   Аны файлга жазат
жаңыФайл.write("Мында биздин файл!\n")
жаңыФайл.write("Мында тексттин экинчи сабы!\n")
# Close () функциясы биз иштеп жаткан файлды жаап, сактайт
# Файл үстүндө иштөө бүткөндөн кийин аны ар дайым жабуу маанилүү
# Эч кандай толуктоолорду киргизбешибиз же файлды бузуп албашыбыз үчүн
жаңыФайл.close()
# ТүзүлгөнФайл.txt файлын ачыңыз
окуу = open("ТүзүлгөнФайл", 'r')
# Файлдын мазмунун окуп чыгыңыз
# txt файлындагы биринчи сапты окуп чыгыңыз
print(окуу.readline())
# txt файлындагы экинчи сапты окуп чыгыңыз
print(окуу.readline())
```

```
# Файлды кайрадан жабыңыз
окуу.close()
```

readline() модулунан тышкары, окшош көрүнгөнүнө карабастан, бир аз башкача иштеген readlines() деп аталган функция дагы бар. Эгерде биз кодубузду басып чыгаруу үчүн өзгөртө турган болсок (өзгөртпөңүз), print(окуу.readlines()) txt файлынан тексттин саптарын басып чыгаруунун ордуна, файлдагы саптардын тизмесин басып чыгармак. Натыйжада, мындай нерсе болмок:

```
['Мында биздин файл!\n', Мында тексттин экинчи сабы!\n']
```

Файлдарга окуу жана жазуу жөнүндө эскертүү

Мындан ары алга жылуудан мурун, файлдарга жазуу кандайча иштээрин талкуулашыбыз керек. Файлга биринчи жолу жазганыңызда баары жайында болот. Бирок, биз файлды ачып, "w" параметрин колдонуп ага экинчи жолу жазууга аракет кылсак, жазууга аракет кылып жаткан файлдагы мурда жазылган нерсенин үстүнө жазасыз.

Мисалы, сиз код жазган болсоңуз:

```
# Бул код файл ачуу үчүн колдонулат
# Бирок, файл мурунтан эле жок болгондуктан
# Анын ордуна Python аны биз үчүн түзөт
жаңыФайл = open("ТүзүлгөнФайл.txt", 'w')
# Бул код print() операторуна окшош
# Бирок, колдонуучунун компьютеринин экранына текст жазуунун же чыгаруунун ордуна
#  Аны файлга жазат
жаңыФайл.write("Мында биздин файл!\n")
жаңыФайл.write("Мында тексттин экинчи сабы!\n")
# Көбүрөөк текст кошуу үчүн Файл ачуу
файлгаКошуу = open("ТүзүлгөнФайл.txt", 'w')
# Көбүрөөк текст жазуу
файлгаКошуу.write("Бул да текст.\n")
файлгаКошуу.close()
```

Жыйынтыгын басып чыгарууга аракет кылдыңыз, натыйжасы кандай болот деп ойлойсуз?

Сиз бул жерде төмөндөгүдөй бир нерсе болот деп күтсөңүз болот:

```
Мында биздин файл!
Мында тексттин экинчи сабы!
Бул текст.
```

Бул туура эмес болмок. Чындыгында, файлды экинчи жолу ачып, ага жаза баштасак, мурунтан бар болгон тексттин үстүнө жазабыз жана анын ордуна жаңы саптарды киргизебиз. Бул сценарийдеги чыныгы жооп төмөндөгүдөй:

```
Бул да текст.
```

Демек, окуянын жыйынтыгы жөнөкөй: файлдар менен кайсы режимде иштегениңизди ар дайым эсиңизден чыгарбаңыз.

Файлдарга тиркөө

Файлдагы бир дагы тексттин үстүнө жазбастан, файлга кантип жазуу керек деген дилемманы чечүү үчүн, биз жөн гана 'w' параметринен 'a' - же append - параметрине өтөбүз.

FunWithFiles.py файлына дагы бир тексттин сабын кошкубуз келди дейли. Болгону, файлды ачып, кошуу режимине өтүү керек. Келгиле, программабызды төмөнкүлөргө дал келиши үчүн өзгөртөлү:

```
# Бул код файлды ачуу үчүн колдонулат
# Бирок, файл азырынча жок болгондуктан
#Анын ордуна  Python  аны биз үчүн түзөт
жаңыФайл = open("ТүзүлгөнФайл.txt", 'w')

# Бул код print() операторуна окшош
#  Бирок, колдонуучунун компьютеринин экранына текст жазуунун же чыгаруунун ордуна
#  Аны файлга жазат
жаңыФайл.write("Мында биздин файл!\n")
жаңыФайл.write("Мында тексттин экинчи сабы!\n")

# Close () функциясы биз иштеп жаткан файлды жаап, сактайт
# Файл менен иштөө бүткөндөн кийин аны ар дайым жабуу маанилүү
```

```
# Эч кандай толуктоолорду киргизбешибиз же файлды бузуп албашыбыз үчүн
жаңыФайл.close()
# ТүзүлгөнФайл.txt файлын ачыңыз
окуу = open("ТүзүлгөнФайл.txt", 'r')
print("ФАЙЛДАГЫ ОРИГИНАЛ ТЕКСТ")
print(окуу.readline())
print(окуу.readline())
# Файлды жабуу
окуу.close()
# Файлды ага текст жазуу үчүн кайрадан ачуу
файлгаКошуу = open("ТүзүлгөнФайл.txt", 'a')
# Файлга текст кошуу
файлгаКошуу.write("Бул да текст.")
# Файлды жабуу
файлгаКошуу.close()
# Окуу үчүн файлды дагы бир жолу ачуу
# Эми биз бир сапты коштук
print("БИЗ КОШУМЧА ТЕКСТ ТИРКЕГЕНДЕН КИЙИНКИ ФАЙЛ")
кошумчаланганФайл = open("ТүзүлгөнФайл.txt", 'r')
# Бул файлдан басып чыгаруунун дагы бир жолу
# Бул жерде for циклын колдонуп жатабыз
# Жана ар бир сапты басып чыгаруу үчүн in жана line ачкыч сөздөрүн
колдонуу
# Текст файлында тиркелгенФайл сап үчүн: print(line)
# Файлды кайрадан жабуу
кошумчаланганФайл.close()
```

Ушундай жол менен бир нече толуктоолорду киргиздик. Документтерди бекемдөө практикасынын аркасында кандай өзгөрүүлөр болуп, алардын эмне жасаганы көрүнүп турушу керек.

Ушундай болгонуна карабастан, кошулган коддун айрымдарын талкуулайлы.

Алгач, коддун кыскача обзору жана анын максаты тууралуу айталы. Коддун максаты:

- Жаңы txt файлын түзүү;
- Файлга эки сап текст жазуу;
- Файлды окуу үчүн файлды окуу режиминде ачуу;
- Файлдагы саптарды басып чыгаруу;
- Файлды кошуу режиминде ачуу;
- Файлга тексттин жаңы сабын кошуу;
- Өзгөртүлгөн файлдын мазмунун басып чыгаруу.

Ушул кадамдардын ар биринин аралыгында биз файлды жаптык. Ошондуктан, окуу, жазуу же кошуу үчүн ачылган ар бир учур үчүн биз ар дайым жакшы код жазганыбызга ишенимдүү болууну кааладык жана файлды жаптык. Бул файлды коддоонун эффективдүү жолу болбосо да, биздин максат үчүн жөнөкөй тилди, коддоо принциптерин жана теорияны үйрөнүү үчүн эң жакшы иштейт.

Акыры, сиз кодубуздун аягында кичине **for** циклын кошконубузду байкадыңыз. Бул тексттик файлдын саптарын басып чыгаруунун дагы бир жолу.

Папкалар менен иштөө

Жогоруда талкуулагандай, ушул кезге чейин баштапкы Python орнотулган папканын ичинде гана иштедик. Бул бөлүмдө, компьютердеги башка папкалардан же каталогдордон файлдарды ачууну үйрөнөбүз.

Бирок, муну жасоодон мурун, учурда кайсы папкада экенибизди табуунун оңой жолун карап көрөлү. WorkingWithDirectories.py аттуу жаңы Python түзүп, ушул кодду киргизиңиз:

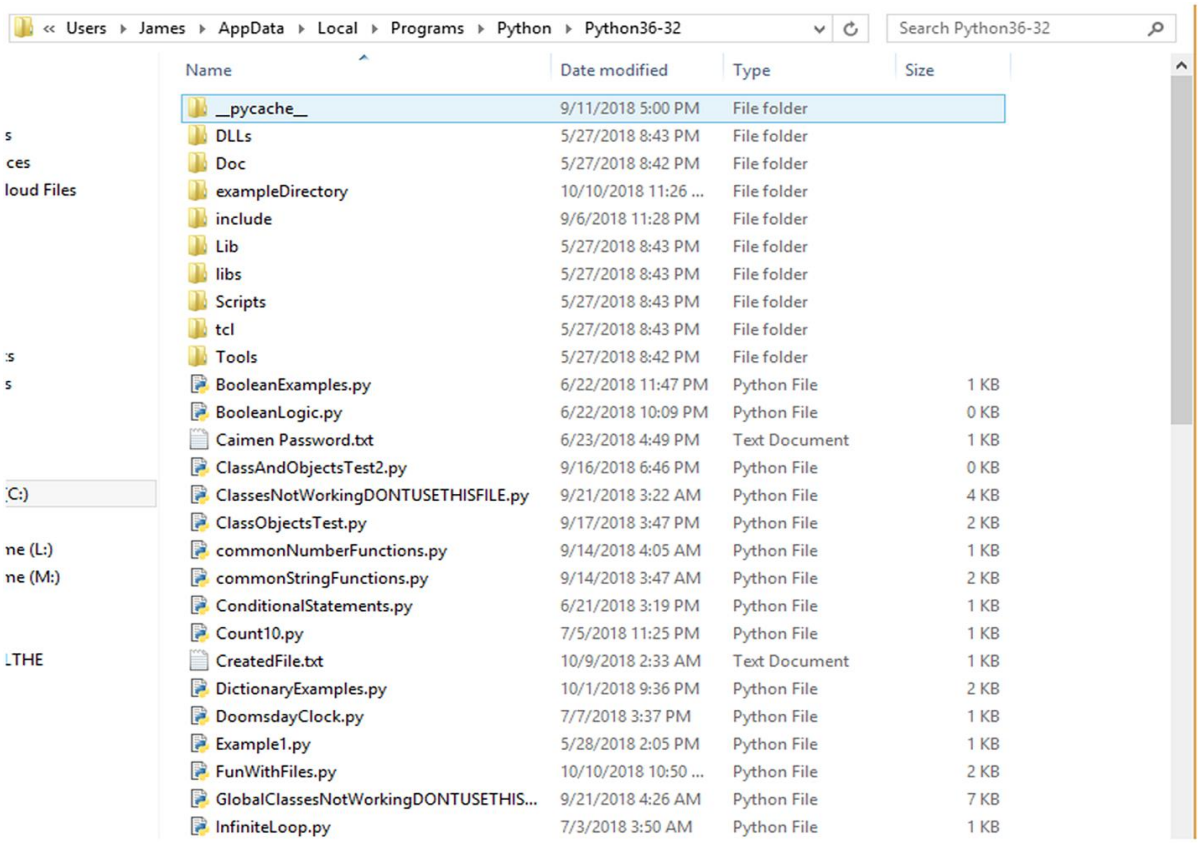
```
# os модулун импорттоо
# Бул операциялык системанын маалыматы менен иштөө үчүн колдонулат
import os
# os модулунун getcwd() методун колдонуңуз
# Биз кайсы папкада экенибизди көрүү үчүн
os.getcwd()
```

Бул кодду иштеткенде сиз меникине окшош жыйынтыкка ээ болосуз. Ал башкача болот. Анткени биздин компьютер тутумдарыбыз жана орнотууларыбыз ар башкача. Бирок төмөнкүдөй көрүнүш пайда болушу керек:

C:\Users\James\AppData\Local\Programs\Python\Python36-32

Бул колдо болушу керек болгон маанилүү маалымат. Анткени файлдар ар кандай папкаларда жайгашкан. Эгерде биз учурдагы тутумубуздан файлды ачууга аракет кылсак жана ал жок болсо, биз ката билдирүүсүнө туш болобуз же кокустан файлдын жаңы версиясын түзүп алабыз. Ар кандай тутумдарда файлдын бир нече көчүрмөсү болсо, албетте, түшүнүксүз болмок.

Эми учурдагы тутмубузду билип, башка папкага өтө алышыбыз *мүмкүн.* Ошол жерден файлды ача алабыз. Мен "*мүмкүн*" деп айтам. Анткени биз аны сынап көрүүдөн мурун өзгөртө турган жаңы тутум түзүшүбүз керек. Эсиңизде болсо, ушул бөлүмдүн башында мен азыркы Python каталогум кандай болгонун көрсөткөм. Ал сүрөт бир аз адаштырды. Анткени анда мендеги туттумдардын же папкалардын бардыгы жок болчу. Чындыгында, ал кандай көрүнөт (10-2-сүрөттү караңыз):



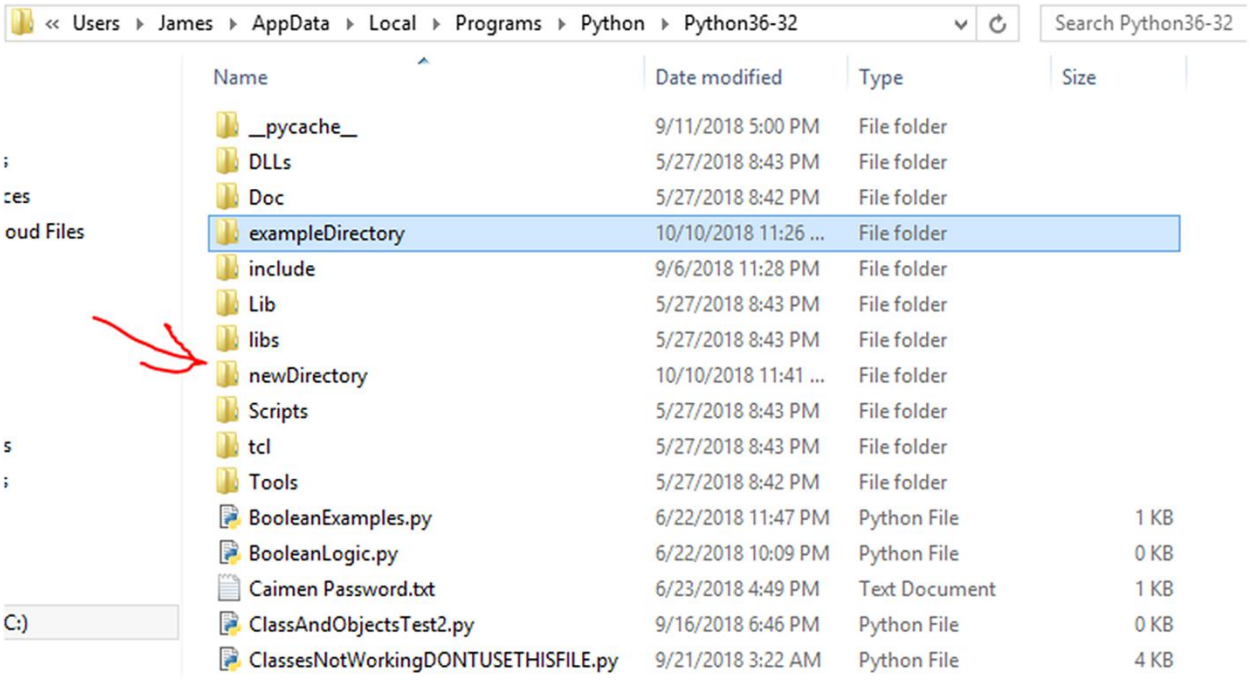
Сүрөт - 10-2. *Файлдарды жана папкаларды көрсөткөн менин чыныгы Python каталогумдун көрүнүшү*

Эгер сиз ушул китепти ээрчип, файлдарды сунушталгандай кылып түзүп келе жаткан болсоңуз, сиздин папкаңыздын бир-эки файлы эле жок болбосо, ушуга окшош көрүнүшү керек.

Келгиле, os каталогунун mkdir() методу менен newDirectory деп аталган жаңы папка түзөлү. Бул кодду WorkingWithDirectories.py файлыңызга кошуңуз:

```
# Жаңы тутум же папка түзүү
os.mkdir("newDirectory")
```

Эми newDirectory деп аталган жаңы папканы түзө турган кодду иштетиңиз. Эгерде, сиз Python каталог папкаңызды ачсаңыз, анда ал меники сыяктуу тизмеде кошулганын көрүшүңүз керек (10-3-сүрөттү караңыз):



Сүрөт - 10-3. *Жаңы түзүлгөн "newDirectory" папкасы*

Эми кийинки бөлүгү маанилүү! Көрсөтмөлөрдү алмаштырардан мурун жаңы эле кошкон кодубузду комментарийлеп алалы. Биз ага комментарий жазышыбыз керек. Антпесек, ката жөнүндө билдирүү алабыз. Эмнеге? Анткени ал бар болсо, анда Python каталог түзбөйт. Ал эми биз аны жаңы гана түздүк – түшүнүктүү болду го дейм!

Кодуңузду меники менен дал келгидей кылып өзгөртүп, андан кийин программаны иштетиңиз.

Эскертүү. *("C:/Users/James/AppData/Local/Programs/Python/Python36-32/newDirectory") менин эмес, сиздин тутум менен дал келгенин текшериңиз; os.getcwd() колдонууну үйрөнгөн ушул бөлүмдүн биринчи мисалында кайтарылган маанини колдонсоңуз болот. Эгер андай болбосо, ката жөнүндө билдирүү аласыз. Ошондой эле, тутум жолундагы \ белгилерин / белгилерине өзгөртүүнү унутпаңыз, антпесе дагы ката жөнүндө билдирүү аласыз.*

Бул жерде код:

```
# os модулун импорттоо
# Бул операциялык системанын маалыматы менен иштөө үчүн колдонулат

import os

# os модулунун getcwd() методун колдонуңуз
# Биз кайсы папкада экенибизди көрүү үчүн

os.getcwd()

# Жаңы тутум же папка түзүү
# Биз папканы мурда түзгөнүбүз үчүн, мында комметарийлейбиз
# Эгер биз түзбөгөн болсок, Python аны түзөт
# Ката билдирүүсүн алуу

# os.mkdir("newDirectory")

# Тутумдарды өзгөртүү үчүн chdir() методун колдонуу

os.chdir("C:/Users/James/AppData/Local/Programs/Python/Python36-32/
newDirectory")

print(os.getcwd())
```

Эскертүү! Эгер сиз ката жөнүндө билдирүү алган болсоңуз, анда анын себеби сиз өзгөртүүгө аракет кылган тутум туура эмес деп билиңиз. Эгер сиз кодуңузду мениki менен дал келүү үчүн жазган болсоңуз, анда сөзсүз түрдө ушундай болот. Эсиңизде болсун, биздин папкалар айырмаланат. Андыктан сиз өзүңүздүн тутумду киргизишиңиз керек.

Мисалы:

```
os.chdir("C:/Users/James/AppData/Local/Programs/Python/Python36-32/
newDirectory")
```

Менин тутумумду ушундайча өзгөртө алам. Сиздики төмөндөгүгө окшош болушу мүмкүн:

```
os.chdir("C:/Users/СиздинАтыңыз/Programs/Python/Python36-
32/newDirectory")
```

Бул бөлүмдүн биринчи os.getcwd() мисалындагы жыйынтыкка кошумча /newDirectory кошулган болушу керек.

Мындан тышкары, \ белгилерин / белгилерине алмаштырууну унутпаңыз. Мисалы, менин баштапкы тутумум:

C:\Users\James\AppData\Local\Programs\Python\Python36-32

бирок биз аны коддо жазганда, мындай болушу керек:

C:/Users/YourName/Programs/Python/Python36-32/

Сиздин кодуңуз иргелип, аны иштеткенден кийин которулган тутумду көрсөткөн төмөнкүгө окшогон натыйжаны аласыз:

```
C:\Users\James\AppData\Local\Programs\Python\Python36-32\newDirectory
```

Тутумдун акыркы бөлүгүндө \newDirectory деп жазылганда биз коддун иштегенин билебиз.

Эми жаңы документтик тутумду кантип түзүүнү жана башка папкага кантип өзгөртүү керектигин билгенден кийин, баштапкы папкага кайтып келели, ошондо китепте ушул убакка чейин түзүлгөн код менен иштөөнү уланта алабыз.

Артка кайтуу үчүн биз жөн гана chdir() ыкмасын колдонуп, бул жолу аны баштапкы папкабызгы багыттайбыз. Эсиңизде болсун, мен жазган нерсенин ордуна өзүңүздүн баштапкы папкаңызды колдонуңуз:

```
# Документтик тутумду алмаштыруу үчүн chdir() методун колдонуңуз

print("NewDirectory папкасына өзгөртүү: ")

os.chdir("C:/Users/James/AppData/Local/Programs/Python/Python36-32/
newDirectory")

# Тутутмдун өзгөртүлгөндүгүн текшерүү үчүн басып чыгарыңыз
print(os.getcwd())

# Түпнуска папкага которулуу
# Меникин эмес, өзүңүздүн тутумуңузду колдонуңуз!

os.chdir("C:/Users/James/AppData/Local/Programs/Python/Python36-32")

# Баштапкы папкага кайтып келгенибизди текшерүү

print("Баштапкы папкага кайтуу: " )
print(os.getcwd())
```

Бул жерде биз тутумдун кайсы баскычында экенибизди көрсөтүү үчүн бир нече print() функциясын коштук. Ошондой эле, баштапкы папкабызгы кайтуу үчүн

тутумдагы акыркы өзгөртүүлөрдү коштук. Натыйжада, иштетилгенде төмөнкүлөргө окшош болушу керек:

```
NewDirectory папкасына өзгөртүү:
C:\Users\James\AppData\Local\Programs\Python\Python36-32\newDirectory
Баштапкы тутумга кайтуу:
C:\Users\James\AppData\Local\Programs\Python\Python36-32
```

Тутум түзүү жана алардын ортосунда которуштуруу маселесин талкуулоодон мурун акыркы нерсеге көңүл буралы. Келечекте башаламандыкка жол бербеш үчүн newDirectory папкасын жок кылалы.

Жөн гана Python папкасын ачып, папканы басып, өчүрүүнү тандасаңыз болот. Бирок, биз азыр программистпиз. Программисттерге белгилүү болгондой, биз үчүн оор жумушту жасаш үчүн кодду колдонушубуз керек!

Папканы өчүрүү үчүн ушул кодду файлга кошуңуз:

```
# NewDirectory папкасын өчүрүү
# rmdir() метолун колдонуу
os.rmdir('newDirectory')
```

Бул кодду иштеткенден кийин эгер сиз Python папкаңызды карасаңыз, анда newDirectory папкасы жок болуп калганын көрө аласыз. Папканы табуу үчүн толук Python жолун колдонуунун кажети жок экенине көңүл буруңуз. Себеби, папка учурдагы Python издеген түпнуска папкасында бар (мисалы, C:\Users\James\AppData \ Local \ Programs \ Python \ Python36-32). Эгер сиз папкаларды "newDirectory" кылып өзгөртсөңүз, анда mkdir жана chdir колдонулганда дагы ушундай болот.

Бонустук раунд

Бул бөлүмдөн файлдар менен иштөө жана документтерди навигациялоо жөнүндө көп нерселерди билдик. Бирок дагы бир нече нерсени үйрөнүшүбүз керек. Мен ашыкча маалымат менен сиздин башыңызды ооруткум келбейт. Андыктан ушул өзгөчө супер жашыруун бонусту кыска жана кызыктуу кылайын деп жатам.

Акыркы бөлүмдөн папкаларды кантип жок кылууну билдик. Бирок файлдарды жок кылуу жөнүндө эмнени айтууга болот? Файлдарды жок кылуу өтө жөнөкөй. Биз бул жерде көрсөтүлгөндөй remove() ыкмасын колдонобуз:

```
# os модулун импорттоо
import os
```

```
# Remove() ыкмасын колдонуп файлды алып салуу
os.remove('тест.txt')
```

Бул код учурдагы папкадан тест.txt файлын алып салат. Эгерде файл учурдагыдан башка папкада жайгашкан болсо, биз ошол тутумга өтүп, андан кийин remove() ыкмасын колдонсок болот. Же болбосо, файлдын жолун жана атын remove() методу менен мындайча берсек болот:

```
 # os модулун импорттоо
import os
# Remove() ыкмасын колдонуп файлды алып салуу
# Эгер файл newDirectory папкасында бар болсо
os.remove('C:\Users\James\AppData\Local\Programs\Python\Python36-32\
newDirectory\тест.txt')
```

Акыры файлдын атын өзгөртүүнү каалаган учур келиши мүмкүн. Биз муну башка ыкманы колдонуп жасай алабыз: ал rename():

```
# os модулун импорттоо
import os
# rename() ыкмасын колдонуп файлдын атын өзгөртүү
# Атын өзгөртүү үчүн эки аргумент талап кылынат
# Файлдын учурдагы аты жана жаңы аты
os.rename('тест.txt', 'кийинкиТест.txt')
```

Бул код биздин учурдагы папкабыздагы тест.txt файлын алып, анын атын кийинки Тест.txt деп өзгөртөт.

FunWithFiles.py коду

Бул жерде биздин FunWithFile.py файлынан алынган бардык коддордун көчүрмөсү берилген. Өзүңүздүн өзгөртүүлөрүңүздүн натыйжаларын көрүү үчүн аны улам иштетип, ушул кодду өзгөртүп, аны менен тажрыйба алып көрүңүз!

```
# Бул код файл ачуу үчүн колдонулат
# Бирок, файл мурунтан жок болгондуктан
# Анын ордуна Python файлды биз үчүн түзөт
жаңыФайл = open("ТүзүлгөнФайл.txt", 'w')
# Бул код print() операторуна окшош
```

```
# Бирок, колдонуучунун компьютеринин экранына текст жазуунун же
чыгаруунун ордуна
#  Аны файлга жазат
жаңыФайл.write("Бул биздин файл!\n")
жаңыФайл.write("Мында тексттин экинчи сабы!\n")

# Close() функциясы биз иштеп жаткан файлды жаап, сактайт
# Файлды менен иштөө бүткөндөн кийин аны ар дайым жабуу маанилүү
# Биз эч кандай толуктоолорду киргизбешибиз же файлды бузуп албашыбыз
үчүн

жаңыФайл.close()

# ТүзүлгөнФайл.txt файлын ачыңыз

окуу = open("ТүзүлгөнФайл.txt", 'r')

print("ФАЙЛДАГЫ ОРИГИНАЛ ТЕКСТ ")
print(окуу.readline())
print(окуу.readline())

# Файлды жабуу
окуу.close()

# Текст жазуу үчүн файлды кайрадан ачуу
файлгаКошуу = open("ТүзүлгөнФайл.txt", 'a')

# Жазуу режиминде файлга бир аз текст кошуу
файлгаКошуу.write("Бул да текст.")

# Файлды жабуу
файлгаКошуу.close()

# Окуу үчүн файлды дагы бир жолу ачуу
# Эми биз бир сапты коштук

print("БИЗ КОШУМЧА ТЕКСТ ТИРКЕГЕНДЕН КИЙИНКИ ФАЙЛ")

кошумчаланганФайл = open("ТүзүлгөнФайл.txt", 'r')

# Бул файлдан басып чыгаруунун дагы бир жолу
# Бул жерде for циклин колдонуп жатабыз
# Текттик файлдагы ар бир сапты басып чыгаруу үчүн
# саптагы ачкыч сөздөрдү колдонуу
```

```
for line in кошумчаланганФайл:
print(line)

# Файлды кайрадан жабуу
кошумчаланганФайл.close()
```

WorkingWithDirectories.py

Мында WorkingWithDirectories.py файлынан алынган толук код берилген. Эскерте кетүүчү нерсе, айрым коддорго комментарийлер берилет. Анткени аны бир нече жолу колдонууда ката кетиши мүмкүн. Бул жаңы папкаларды түзгөндө жана өчүргөндө байкалат. Эгер мурунтан бар папканы түзүүгө аракет кылсак, анда ал ката билдирүүсүн берет.

Дагы бир жолу, ушул код менен тажрыйба жасап, көңүл ачыңыз. Кантсе да, кодду бузуп, андан кийин аны оңдосок, анда биз, чындыгында, күчтүү программист супер баатырларга айланабыз!

```
# os модулун импорттоо
# Бул операциялык система менен иштөө үчүн колдонулат

import os

# os модулунун getcwd() ыкмасын колдонуу
# кайсы папкада экенибизди көрүү үчүн
os.getcwd()

# Жаңы папка түзүү
# Биз папканы мурда түзгөнүбүз үчүн, мында комметарийлейбиз
# Эгер биз түзбөгөн болсок, Python аны түзөт
# Ката билдирүүсүн алуу

# os.mkdir("newDirectory")

# Тутумдарды өзгөртүү үчүн chdir() ыкмасын колдонуу

print("NewDirectory папкасына өзгөртүү: ")

os.chdir("C:/Users/James/AppData/Local/Programs/Python/Python36-32/
newDirectory")
# Учурдагы тутумдун өзгөртүлгөндүгүн текшерүү үчүн басып чыгарыңыз

print(os.getcwd())

# Түпнуска каталогго которулуу
```

```
# меникин эмес, өзүңүздүн тутумуңузду колдонуңуз!

os.chdir("C:/Users/James/AppData/Local/Programs/Python/Python36-32")

# Баштапкы каталогго кайтып келгенибизди текшерүү
print("Баштапкы каталогго кайтуу: " )
print(os.getcwd())

# NewDirectory каталогун жок кылуу үчүн
# rmdir () ыкмасын колдонуу
os.rmdir('newDirectory')
```

Бул эпизоддо

Сиз бул укмуштуу окуяда, чындыгында, кайраттуулук көрсөттүңүз, жаш баатыр! Сиз желмогуздун да ичин күйгүзүү үчүн жетиштүү билимди үйрөндүңүз. Баса, бул жигиттин маңдайын көрүшүңүз керек. Ал абдан чоң!

Канчалык акылдуу жана таланттуу болсоңуз дагы, бир аз үйрөнгөн нерсеңизди кайталап, жаңыртып алуу пайдалуу. Анда көп ойлонбостон, төмөндө бул эпизоддун кыскача баяндамасына көз чаптыр:

- *Python .py, .txt, .html, .c, CSV жана JSON сыяктуу көптөгөн файл түрлөрүн иштете алат;*
- *open() файлды ачуу үчүн колдонулат. Ошондой эле мурунтан мындай аталыштагы файл жок болгон шартта, open() менен файл түзсө болот;*
- *Мисалы; open(“ТүзүлгөнФайл.txt”, 'w')*
- *'w' параметри файлды жазуу режиминде ачуу үчүн колдонулат;*
- *"x" параметри файлды түзүү / жазуу режиминде ачуу үчүн колдонулат;*
- *.write методу файлга текст кошууга мүмкүнчүлүк берет;*
- *.write методун колдонуунун мисалы: жаңыФайл.write (“Мында текст бар.”);*
- *Файл менен иштеп бүткөндө, ар дайым close() функциясын колдонуп жабыңыз;*
- *Мисалы; жаңыФайл.close();*
- *"r" аргументи файлды окуу үчүн ачканда колдонулат;*
- *.read() методу файлдагы тексттин бардыгын окуйт;*
- *.read() ыкмасын колдонуунун мисалы: print (readMe.read ());*
- *Файлдын бир сабын окуу үчүн readline() ыкмасын колдонобуз;*
- *Мисалы; print(readMe.readline ());*

- *кошуу режими - "a" - бар файлга жазуу үчүн колдонулушу керек. "w" ачкычын колдонуу менен, мурунку файлдын мазмунуна жазуу жүргүзүлөт;*
- *Папкалар менен иштөө үчүн, биз **os** модулун импорттошубуз керек;*
- *Учурдагы кайсы папкада экенибизди көрүү үчүн getcwd() ыкмасын колдонобуз;*
- *Мисалы: os.getcwd();*
- *Жаңы каталог түзүү үчүн mkdir() колдонобуз. Мисалы; os.mkdir(“newDirectory”);*
- *Тутумду өзгөртүү үчүн chdir() колдонобуз.*

  Мисалы: os.chdir

  ("C:/Users /СиздинАтыңыз/");
- *rmdir() жардамы менен папканы алып салабыз же жок кыла алабыз. Мисалы:*

  os.rmdir(“newDirectory”).
- *Файлдарды os.remove() ыкмасын колдонуп өчүрө алабыз. Мисалы; os.remove*

  ('тест.txt').
- *os.rename() аркылуу файлдардын атын өзгөртө алабыз. Мисалы: os.rename (‘тест.txt ', 'кийинкиТест.txt').*

11 - БӨЛҮМ

PYTHON ОЮНДАР ҮЧҮН

Бизде Python программалоо тилинде видео оюндарды түзүүнү талкуулаган бөлүмдүн болгону жөндүү. Кантсе да дал ушул көп жылдар мурунку кызыгуум мени кичинекей кезимден программалоону үйрөнүүгө түрткү берген экен. Ошондон бери иш бир топ илгериледи. Ал кезде компьютердик оюндар текстке негизделип, өтө сапатсыз графикадан турган сүрөттөрдөн турчу. ASCII белгилеринен жасалгандары, андан да сапатсыздары бар болчу.

Алтургай, үндөр да абдан жөнөкөй болгон: бир үндүү санариптик биип, баап жана бооптор, элестетип көрсөңүз. Анимацияларчы? Ооба, алар техникалык жактан бар болчу. Ошентип, менин доорумдун чындап эле жогорку технологиялуу компьютер оюнунун мыкты мисал катары YouTube платформасынан *Where in the World is Carmen San Diego* жана мен эң жакшы көргөн *The Oregon Trail* компьютердик оюндарын карап көрсөңүз болот.

Алга! Мен күтөм. Күлүп бүттүңбү? Жакшы, анда эмесе, уланта берели.

Бул сапаттуу видео оюндар болгон эмес дегенди билдирбейт. Ошол учурда Atari көп жылдар бою ойнолуп келген. Nintendo Entertainment System (NES), Sega жана Commodore бар да болчу. Мен Nintendo үлгүсүнө ээ болуп, жогорку технологиялуу 8-биттик графикага жана мыкты үнгө суктанчумун.

Ал оюндар ошол кезде сонун болсо да, алардын айрымдары бүгүнкү күндө да бар. Менин компьютердик оюн борборлорунда ойногон көптөгөн оюндарыма караганда көңүлдүү. Ошол консолдук оюндарда менин азыр комьютердеги оюндарымда бар нерсе жок болчу. Мен аларды бузуп, андан да кызыгы, компьютерде өзүмдүн версияларымды түзүп алганмын.

Азыр абал башкача. Кааласаңыз, ири консолдор үчүн иштеп чыгуучунун консолун сатып алып, керектүү ресурстар жана көндүмдөр менен өз оюнуңузду иштеп чыгууну баштасаңыз болот. Бул оюндар кайсы бир оюн дүкөндөрүндө сатылабы же жокпу, аны ким билет? Бирок, чындыгында, техникалык жактан сиз ушул күндөрү консоль оюндарын түзө аласыз.

Ал кездеги менин курагымда болсоңуз, сиз муну жасай алмак эмессиз.

Видео оюндар - компьютердик программалоо көндүмдөрүн үйрөнүүнүн мыкты жолу. Татаал видео оюнун эске алганда сиз кодер булчуңдарыңызды чыңап алсаңыз болот. Сиз кодду адатта аларды колдонууну ойлобогон ыкмалар менен колдоно баштайсыз жана оюн жазардан мурун оюн үчүн кодуңузду алдын ала пландаштырышыңыз керек. Айрыкча, анын аркасында сюжеттик сызык менен оюн жарата турган болсоңуз, бул бөлүк абдан маанилүү болот.

Бирок, мен үчүн баарынан маанилүүсү, видео оюндар адамдын эргүүсүн өстүрө алат. Эгерде, бул китептеги маалымат сиздин фантазияңызды чын эле

кызыктырбаса же программалоонуу каалабасаңыз, анда өзүңүздүн оюнуңузду жаратып жаткан кезде кызыктуу болот деп үмүттөнөм.

Оюндарды жаратууну каалабасаңыз да, коопсуздукка, иш-такта колдонмолорго, маалымат илимине же веб-платформалары менен иштөөгө кызыктар болсоңуз да, мен ушул бөлүмдөгү көрсөтмөлөрдү аткарууну сунуштайм. Бул жерде аябагандай терең коддоо болбосо дагы, оюндардан тышкары колдонула турган кээ бир түшүнүктөрдү камтыйбыз. Мисалы, үн, сүрөт, алтургай, анимация менен иштөө да бар.

Мындан сырткары, ар бир баатыр өз күчтөрү жөнүндө мүмкүн болушунча көбүрөөк билиши керек. Менин айтайын дегеним, Супермен бийик имараттардан секире алат дешет. Бирок, анын секиргенин канча жолу көрдүң эле?

Ошентсе да, бир күнү ал мындан ары уча албай калган жерде (балким, ал пилоттук күбөлүгүнөн айрылып калышы мүмкүн) бир нерсе болуп кетиши мүмкүн. Анан кантип ал Метрополистин чокусуна чыгып, ал жерде өтүп жаткан кечеге чаң салып, катышуучуларды төмөн карай ыргытууну каалаган мээси чоң дөөдөй маймылды токтотуп калат?

Python оюндар үчүн

Python видео оюндарын программалоону ойлогондо, биринчи кезекте ойго келе турган тил эмес. Айтмакчы, ал - айрым ири оюндарда, мисалы Battlefield, компьютерде жана кээ бир Python колдонгон консолдордо болгон оюндун мыкты үлгүсү.

Эгер сиз чындап эле оюн иштеп чыгууну кааласаңыз, анда C ++ жана JAVA жөнүндө мүмкүн болушунча көбүрөөк билгиңиз келет. Алар учурда көпчүлүк оюндарда колдонулган эң мыкты эки тил болуп саналат. Башкалар, мисалы, Unity үчүн C # (Python менен жайылтылат) дагы колдонулат. Бирок, чындыгында, сиз консоль жана компьютер оюндарга умтулсаңыз, айрыкча, C ++ тилине көңүл бурууну каалайсыз.

Эгерде сиз веб-оюндарды программалоону пландаштырсаңыз, анда HTML5, CSS3, JavaScript жана SQL (маалымат базалык тили) керек болот.

Албетте, оюндарды өнүктүрө турган көптөгөн тилдер бар. Бирок бул жерде көрсөтүлгөндөр - алардын эң күчтүүсү, башкача айтканда, оор артилериясы.

Python тилин колдонуу эгер сиз негизги түшүнүктөрдү үйрөнүп, алтургай, өзүңүздүн оюндарыңызды түзүүнү кааласаңыз, көңүл ачуу үчүн болобу, досторуңуз менен бөлүшүү үчүн болобу же портфолиоңуздун бир бөлүгү болсо да, сиз үчүн абдан жакшы чечим болот. Python үйрөнүү C ++ тилине караганда оңойураак, эгерде сиз китепти тереңирээк түшүнүп алсаңыз, анда код жазуунун негиздерин жакшы билип калдыңыз.

Дагы айта кетчү нерсе, Python китептин башында орноткон абдан ыңгайлуу pygame модулуна ээ. Бул - чындыгында, Python программалоо тилинде өз

оюнуңузду жана анимацияңызды түзүүгө мүмкүнчүлүк берген ар кандай модулдардын топтому.

Колуңуздагы Python китеби болгондуктан, биз Python менен оюндарды түзүүгө көңүл бурабыз. Бирок мен сиз Python колдонууну өздөштүргөндөн кийин башка тилдерди да үйрөнүү керек экендигин унутпасаңыз экен дейм.

Python программалоо тилинде жарата ала турган оюндардын түрлөрү

Чындыгында, Python менен түзө турган оюндардын *түрүнө* чек жок. Жок дегенде теория жүзүндө. Сиз роль ойной турган оюндарды (RPG), биринчи адам аткычтарды (FPS), платформаларды, табышмак оюндарын ж.б. жасай аласыз. Бул оюндар текстке негизделген, жөнөкөй графиканын аралашмасы болушу мүмкүн, үндөр жана тексттик, анимацияланган, 2D капталдагы жылдыргычтар (NES оюнундагы Contra сыяктуу оюндарды ойлонуңуз), жадакалса, 3D оюндар болушу да мүмкүн.

Эгер сиз 3D оюндарын жасоону кааласаңыз, анда Panda3D (www.panda3d.org/) сыяктуу кошумча технологияларды үйрөнүшүңүз керек. Биз бул жерде оюндун өнүгүшүнө анчалык көп сүңгүбөйбүз. Бирок ушундай мүмкүнчүлүк бар экендигин билип алыңыз.

Python сизге мыкты оюндарды түзүүгө жардам берсе, чындыгында, ресурстарды көп талап кылган оюндар, эс-тутумду жана иштетүү күчүн талап кылган оюндар, C ++ менен иштелип чыкса жакшы болот. Бул иштетүүгө жана графикалык жабдыктарга көбүрөөк мүмкүнчүлүк берет.

Python программалоо тилинде кандай оюн түрлөрүн иштеп чыгууга боло тургандыгын билүү үчүн жана ушул бөлүмдө карала турган pygame модулун колдонуп кайсы түрүн программаласаңыз болорун билүү үчүн расмий Pygame веб-сайтынын долбоордук китепканасына кирип, ал жердеги көптөгөн оюндарды www .pygame.org/tags/all шилтемеси аркылуу карап чыгыңыз.

Python программисттери тарабынан иштелип чыккан оюндардын түрлөрү, колдонулган китепканалар жана башкалар боюнча маалымат ала аласыз. Бул жер - кандайдыр бир идеяларды алуу, ошондой эле оюндарды ойноп, көңүл ачуу үчүн эң сонун жер!

Pygame модулуна киришүү

Эгерде сиз бул китепти жакшылап окуган болсоңуз, анда pygame модулун орнотуп койгонбуз. Эгер сиз ал бөлүктү өткөрүп жиберген болсоңуз же дагы бир жолу кантип жасоону билгиңиз келсе, анда аны кайрадан орното беребиз. Буга кабатыр болбоңуз.

Бирок, алгач Pygame тарыхы жана анын эмне экендиги жөнүндө бир аз сөз кылышыбыз керек.

Pygame бул модуль деп айтсак болот, чындыгында, бул - видео оюнун иштеп чыгуу үчүн атайын түзүлгөн модулдардын жыйындысы. Ал Пит Шиннерс тарабынан иштелип чыгып, анын биринчи версиясы 2000-жылдын октябрь айында чыккан. Модуль Python, C жана Assembly аралашмасынын жардамы менен жасалган.

Компьютерлердеги оюндардан тышкары, Pygame оюну PGS4A деген аталыштагы топтомду колдонуп, Android түзмөктөрү үчүн оюндарды иштеп чыгуу үчүн колдонсо болот. http://pygame.renpy.org/ дарегине кирип, ушул атайын топтом менен мобилдик тиркеме иштеп чыгууга арналган программалоо оюндары жөнүндө көбүрөөк биле аласыз.

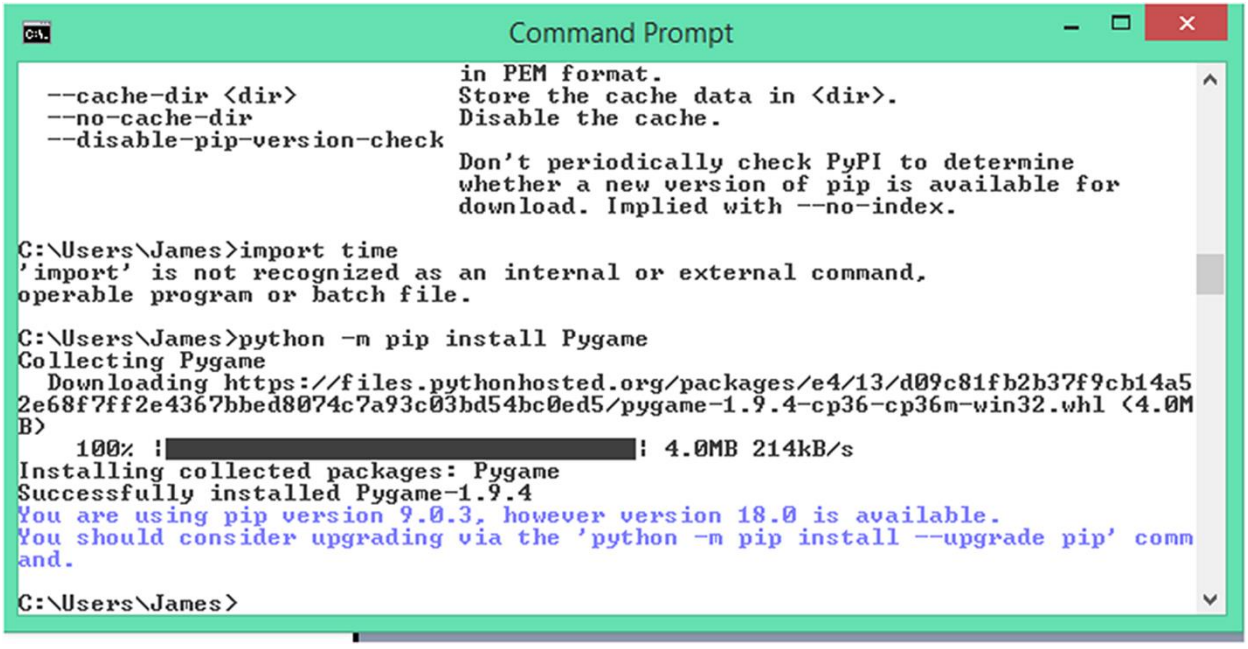
Pygame орнотуу

Жогоруда талкууланган дай, биз pygame модулун орнотуп койгонбуз. Бирок, айкындуулук үчүн 7-бөлүмдүн бир нече барактарын баракта чыккыңыз келбесе, аны кантип орнотсо болорун кайрадан карап көрөлү.

Модулду орнотуу үчүн жана өзгөчө pygame - буйрукту же CMD терезесин ачып, буйрук тилкесине төмөнкүлөрдү киргизиңиз:

python -m pip install Pygame

Эгер сизде pygame орнотула элек болсо, анда CMD терезесинде бир нече мүнөттөн кийин топтомду жүктөө жана орнотуу процесси башталарын көрө аласыз. Билдирүү 11-1 сүрөтүнө окшош болот:



Сүрөт 11-1. *Pygame орнотулууда*

Бул абдан эле жөнөкөй!

Оюнга pygame негизги элементтерин орнотуу

Биринчи кезекте, биздин Pygame модулун куруу үчүн структура керек. Бул үчүн биз жөнөкөй скелеттердин кыймылдаткычын колдонсок болот. Ал анда мындай көрүнүшкө ээ:

```
import pygame
from pygame.locals import *
import sys
# Pygame модулдарынын бардыгын орнотуңуз, ошондо аларды кийинчерээк #колдоно алабыз
pygame.init()

# Оюн экранын түзүп, 800 x 600 пикселге коюңуз
экран = pygame.display.set_mode((800, 600))

# Оюндун иштешин камсыз кыла турган цикл түзүңүз
# колдонуучу чыгууну чечкенге чейин

while True:
# Оюнчудан аракеттер тууралуу пикир алуу
    for аракет in pygame.event.get():
        # Эгерде оюнчу кызыл "x" баскычын баскан болсо, анда ал оюндан чыгуу #болуп эсептелет
         if аракет.type == QUIT:
              pygame.quit()
              sys.exit()
```

Бул - pygame модулун колдонгон оюндун өтө жөнөкөй версиясы. Бул коддо техникалык жактан ойной турган оюн жок болсо дагы, ал биздин оюндарыбызды куруу тутумун түзүп берет. Код төмөнкүдөй иштейт.

Бизге керектүү модулдарды импорттогондон кийин - pygame жана sys - ошондой эле pygame камтылган бардык кошумча модулдарды импорттойбуз. Pygame импорту жетиштүү болушу керек, бирок кээде сиздин системаңызга жараша pygame менен келген бардык модулдар жүктөлбөйт, ошондуктан биз төмөндөгүнү колдонобуз:

```
from pygame.locals import *
```

Биз баарын импорттоп жаткандыгыбызды текшерип алыңыз.

Эми биздин модулдар жүктөлгөндүктөн, биз бардык pygame модулдарын башташыбыз керек. Биз мында pygame.init() кодун колдонуп жасайбыз.

Азырынча программаларыбызда IDLE колдонуп кодду иштетип, натыйжаларын Python Shell баракчасында көрсөттүк. Бирок, оюндарды Pygame модулун колдонуп жазганда, графикага байланыштуу болгондуктан, программаларыбызды көрсөтүү үчүн чыныгы экранды жаратышыбыз керек. Терезе же экран түзүү үчүн .display.set_mode() колдонобуз. Бул сап: экран = pygame.display. set_mode ((800, 600)) туурасы жана бийиктиги 800 x 600 пиксель болгон экранды же терезени жаратат. Дал ушул экранга биз кийинчерээк сүрөтүбүздү, графикабызды жана текстибизди тартабыз же көчүрөбүз.

Бул коддун акыркы бөлүгү, сиз бардык оюндар үчүн түзүшүңүз керек болгон нерсе оюн цикли деп аталат. Бул түзүмдүн максаты эң эле жөнөкөй. Ал - колдонуучудан чычканды басуу жана баскычтарды басуу аркылуу *окуялар* катары белгилүү болгон маалыматты алуу.

Интерактивдүү оюнду түзгөндө, колдонуучу оюнду ойноп бүткөнүн айтып, чыгып кетишине жол керек. Оюн цикли да ушул максатты көздөйт.

while True менен башталган while цикли оюн циклин баштайт. Андан кийин программа окуя түзүү үчүн колдонуучунун кандайдыр бир иш-аракет жасашын күтөт. Дал ушул учурда, биз QUIT окуясын гана издешибиз керек.

QUIT окуясы колдонуучу терезенин оң бурчундагы кызыл X белгисин колдонуп, терезени жапканын билдирет. Мындай болгондон кийин, биз эки маанилүү функцияны колдонобуз. Ошондой эле, бардык Pygames pygame.quit() жана sys.exit() камтышы керек. Бул эки окуя Pygame оюнун аяктап, оюндан чыгат. Сизде экөө тең чогуу болушу керек. Эгер жок болсо, терезеңиз тоңуп же катып калат.

Эгер сиз азыр ушул программаны иштете турган болсоңуз, анда кара фон менен терезе чыкмак. Башка эч нерсе болбойт. Сиз кызыл X белгисин басканда терезе жабылып, программа аяктайт.

Биздин оюн скелетибизге кошуу

Эми биздин Pygame оюнубуздун скелети даяр болгондуктан, бир аз өзгөчөлүктөрдү кошкубуз келсе, аны кошуп, бир аз жандандыра алабыз. Кантсе да, биз супер баатырбыз да, анча-мынча шык-жөндөмсүз кандай баатыр болмок эле?

Алгач, pygameExample.py аттуу жаңы файл түзөлү. Ага төмөнкү кодду кошуңуз:

```
import pygame
from pygame.locals import *
import sys
# RGB (Кызыл, Жашыл Көк) маанилерин сактоо үчүн кортеж түзүү
# Ошентип, кийинчерээк экраныбызды көк түскө боёп алсак болот
көк = (0, 0, 255)
```

```
# Pygame модулдарынын бардыгын баштаңыз, ошондо аларды кийинчерээк колдоно алабыз
pygame.init()

# Оюн экранын түзүп, 800 x 600 пикселге коюңуз
экран = pygame.display.set_mode((800, 600), 0, 32)

# Биздин терезеге тема жазуу орнотуңуз
pygame.display.set_caption("Super Sidekick: Sophie the Bulldog!")

# Экраныбызды / терезебизди көккө боёңуз
экран.fill(көк)

# Азыр көк терезени экранга тартыңыз(жаңылаңыз)
pygame.display.update()

# Оюндун аякташы керекпи же жокпу деген маанини сактоо үчүн өзгөрмө #түзүңүз
иштөө = True

# Колдонуучу чыгууну чечкенге чейин иштей турган цикл түзүңүз
# Ал чечкенде, иштетүүнүн маанисин False деп өзгөртүп, оюнду бүтүрөт

while True:
# Оюнчудан окуялар түрүндө пикир алуу
    for аракет in pygame.event.get():
        # Эгерде оюнчу кызыл түстөгү "x" баскычын басса, анда ал окуяны бүтүрөт деп эсептелет
        if аракет.type == QUIT:
            pygame.quit()
            sys.exit()
```

Бул код сизге мурун көрсөткөн мисалга окшош. Терезени иретке келтирип, бир аз жакшыртсам деген ниет менен дагы бир нече саптарды кодго коштум.

Мен кошкон биринчи код:

```
көк = (0, 0, 255)
```

Комментарийлерден көрүнүп тургандай, бул - RPG (кызыл, жашыл, көк) маанилерин чагылдырган кортеж. Биз кийинчерээк экраныбызга түс берүү үчүн колдонобуз. Биз бул маанилерди экран объектисине / өзгөрмөсүнө кортеж мааниси катары өткөрүп беришибиз керек. Анткени бул - ал кабыл алган маалыматтын түрү.

RGB маанилеринин негизин түзгөн теория мындай: кызыл, жашыл жана көк түстөрдүн айкалышын колдонуп, адамдын көзүнө каалаган түстү көрсөтө аласыз.

Биздин учурда, мындагы биринчи 0 мааниси нөл кызыл түс дегенди билдирет. Экинчи 0 мааниси биздин түсүбүздө нөл жашыл болот дегенди билдирет. Акырында, үчүнчү мааниси 255 - бул биз кошо турган көк түстүн максималдуу суммасы. Эгер биз анын ордуна (0,0,0) мисал келтирсек, анда биз кара түскө ээ болобуз. Демек, башка түстөр жок. Башка жагынан алганда, (255,255,255) ак түскө барабар болот. Анткени ак түс - бардык түстөрдүн айкалышы.

Андан кийин, биз өзүбүз түзгөн терезеге аталышын же коштошуу жазуусун кошууну каалайбыз жана .display.set_caption() колдонуп, төмөнкү саптагыдай кылабыз:

pygame.display.*set_caption("Super Sidekick: Sophie the Bulldog!")*

Бул код 11-2-сүрөттө көрсөтүлгөндөй, терезенин жогору жагында жайгашканга окшош жазууну жаратат.

Сүрөт -11-2. *Терезенин аталышынын мисалы*

Андан кийин, биз иш жүзүндө көк түс менен фон / экранды толтургубуз келет. Бул үчүн биз .fill() колдондук:

```
экран.fill(көк)
```

Бул терезеге азырынча эч нерсе кошпой тургандыгын эске алыңыз. Көк түс тартылардан мурун .display.update() аркылуу дисплейди жаңыртыш керек:

```
pygame.display.update()
```

Кана эмесе, аракеттенип карап көрүңүз. Программадан чыгуу үчүн кызыл X баскычын басууну унутпаңыз.

Pygame оюнуна сүрөттөрдү жана спрайттарды кошуу

Эми оюн терезебизди кандайча форматтап, негизги оюн циклин орнотууну билип алгандан кийинки иш - сүрөттөр менен иштөөнү үйрөнүү. Кантсе да, pygame модулун колдонуунун максаты - видео оюндарды түзүү да, туурабы?

Эки өлчөмдүү же 2D форматындагы видео оюндардагы сүрөттөрдү талкуулаганда, биз аларды *спрайт* деп атайбыз. Бул спрайт түшүнүгүнүн жөнөкөй аныктамасы. Бирок ал биздин максат үчүн жакшы иштейт.

Видео оюндардагы спрайт көбүнчө оюнчуларга тиешелүү каармандарды, душмандарды же сүрөттөрдү билдирет. Спрайттар - бул оюндагы нерселер. Мисалы, ок, бак, таш жана башка ушул сыяктуу нерселер.

Мындай спрайттар кыймылсыз же кыймылдуу болушу мүмкүн. Бул бөлүмдө биз жөн гана статикалык спрайтты талкуулайбыз. Биздин терезе коштомо жазуусунун / аталышы: *Super Sidekick: Sophie the Bulldog* (Супер Сайдкик: Софи Бульдог). Бул кокустан болгон жок!

Көптөгөн супер каармандардын жаныбарлардан турган достору бар. Сотко кайрылып, эбегейсиз байлыгымды жоготуп алуудан жана бир аз эски мопеддердин коллекциясынан айрылып калуудан коркуу мага бирөөнүн атын атоого мүмкүнчүлүк бербейт. Бирок, алардын саны өтө көп экенине ишенем.

Эмне үчүн сен экөөбүз башкача болушубуз керек? Биз дагы кан достуу болууга татыктуу эмеспизби? Менин досум - бул Софи аттуу бульдог. Анын супер күчү - кекирүү, уктоо, бутумдун манжаларын тиштөө, аябай катуу коңурук тартуу.

Биздин коддун ушул бөлүгү үчүн мен оюн терезесине Софи Бульдогдун сүрөтүн кошом. Эгер сизге жакса, анда ээрчисеңиз болот. Андан да жакшысы - эгер сиз досуңуз болсо деп ойлогон жаныбар болсо, чамдаңыз! Сүрөттү pygameExample.py файлыңыз жайгашкан папкага эле сактаңыз. Эгер антпесеңиз, анда сиздин программа аны таба албайт.

Акыркы эскертүү: программада мен жазып жаткан атты эмес, өзүңүздүн файлыңыздын атын колдонуп жатканыңызды текшерип алыңыз. Мисалы, мен колдонуп жаткан сүрөт “SophieTheBullDog.jpg” деп аталат. Сиздики башкача аталышы мүмкүн.

pygameExample.py файлыңызга экран.fill колдонгон бөлүмдүн ылдый жагына жана pygame.display.update() колдонуудан мурун төмөнкү кодду кошуңуз:

```
фигура = pygame.Rect(100,100, 200, 200)
бульдог = pygame.image.load('SophieTheBullDog.jpg')
эскиз_бульдог = pygame.transform.scale(бульдог, (200,200))
экран.blit(эскиз_бульдог, фигура)
```

Толугу менен жаңыланган кодду ушул бөлүктү түшүндүргөндөн кийин жайгаштырам. Ушундан кийин файлыңызды меники менен салыштырсаңыз болот.

Эң алгач биздин сүрөтүбүз чагылдырыла турган же боёлгон башка бетти түзүү керек. Биз буга төмөнкү сап менен жетишебиз: фигура = pygame.Rect (100,100, 200, 200).

Бул код сызыгы экрандын 100дөн 100гө чейинки XY *координатасында* жайгашкан. Ал бийиктиги жана туурасы 200 x 200 пикселди түзгөн тик бурчтуу экранды түзөт.

XY координаттары объекттин экрандагы абалына байланыштуу. Pygame модулунда жараткан беттерибиз пикселден турат. Ар бир пиксел XY абалына байланыштуу тордо жайгашкан. Терезенин жогорку сол бурчу XY позициясында жайгашкан (0, 0). Демек, биз тик бурчтукту (100, 100) позицияга тартканда, чындыгында, ал 100-пиксель горизонталдык жана 100-пикселдик тигинен жайгашат деп айтабыз.

Эгер бул түшүнүксүз болсо, көп тынчсызданбаңыз. Бир нече мүнөттөн кийин программаны иштеткенденде ал мааниге ээ болот.

Коддун кийинки сабы:

```
бульдог = pygame.image.load('SophieTheBullDog.jpg')
```

бульдог өзгөрмөсүндө 'SophieTheBullDog.jpg' деген сүрөттү сактайт. Мындан сырткары, сиздин сүрөтүңүздүн аталышы меникинен айырмаланат. Андыктан жөн гана менин сүрөтүмдүн атын сиздики менен алмаштырыңыз.

Менин 'SophieTheBullDog.jpg' сүрөтүм чоң болгондуктан, ал 1400 x 1400 өлчөмүндө болот. Оюн терезесинде анын азыркы өлчөмүндө көрсөтүү өтө чоң болмок. Биз ал үчүн түзгөн тик бурчтук бетинде да чоң көрүнөт. Ошондуктан, биз аны каалаган өлчөмгө чейин кичирейтүүбүз керек.

Биз муну .transform.scale() жардамы менен жасайбыз. Ал сүрөттү биз белгилеген өлчөмгө чейинки масштабга которот.

Биздин сап:

```
эскиз_бульдог = pygame.transform.scale(бульдог, (200,200))
```

Сүрөттү 200 x 200 пикселге чейин кичирейтет, бул биз түзгөн фигура объектинин тик бурчтук бетинин өлчөмү менен бирдей. Эгер биз аны тик бурчтук бетинен көбүрөөк масштабда көрсөткөн болсок, анда биз бүт сүрөттү көрө албай калмакпыз. Ошондуктан, ар дайым сүрөттүн өлчөмдөрү аны көрсөтүү үчүн сиз жараткан беттин өлчөмдөрүнө дал келгенин текшериңиз.

Акыр-аягы, акыркы кадам боёк же блит болот. Эсиңизде болсун! Сүрөттүн өлчөмү биз түзгөн фигура тик бурчтугуна чейин өзгөрүлөт. Бул үчүн биз мындай деп тердик:

```
экран.blit(эскиз_бульдог, фигура)
```

Кашаанын ичиндеги биринчи аргумент - биз которгубуз келген объекттин аталышы. Экинчи аргумент - бул сүрөттү көчүрүп алгыбыз келген объект (анын жайгашкан жерин камтыйт).

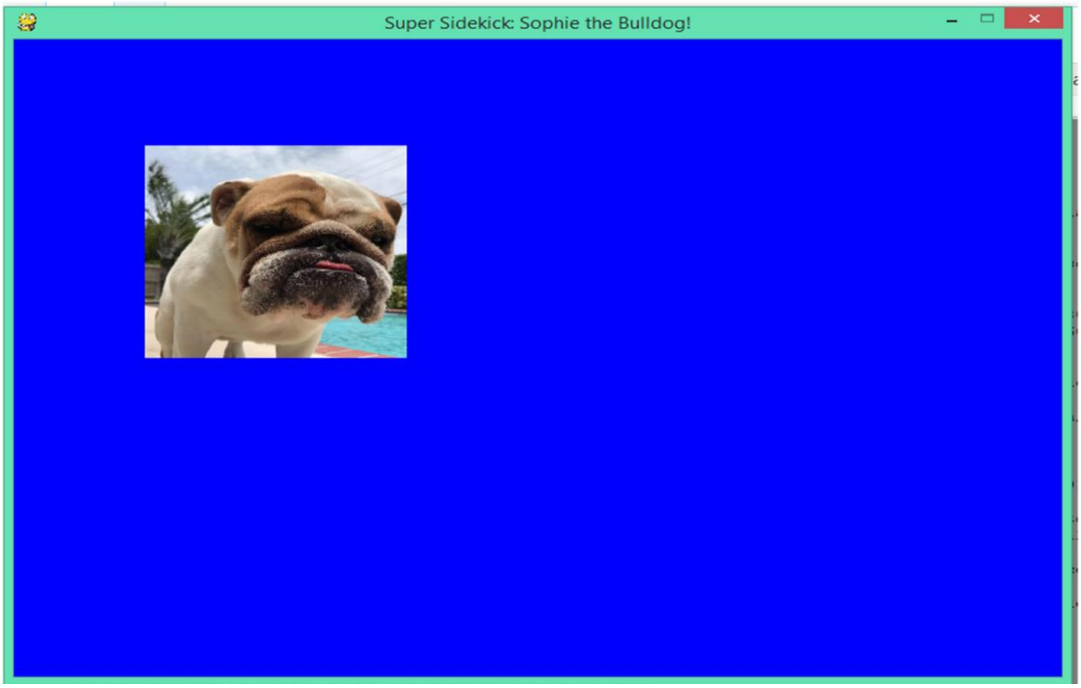
Акыркы код төмөндөгүдөй болушу керек. Кодуңузду меникине окшош кылып өзгөртүңүз, сүрөтүңүздүн аталышы кандай болсо, ошондой кылып өзгөртүңүз. Сүрөтүңүздү pygameExample.py файлы менен бир папкага салганыңызды унутпаңыз, антпесе ал иштебей калат:

```
import pygame
from pygame.locals import *
import sys

# RGB (Кызыл, Жашыл Көк) маанилерин сактоо үчүн кортеж түзүү
# Ошентип, кийинчерээк экраныбызды көк түскө боёп алсак болот
```

```
көк = (0, 0, 255)
# Pygame модулдарынын бардыгын баштаңыз, ошондо аларды кийинчерээк колдоно алабыз
pygame.init()
# Оюн экранын түзүп, 800 x 600 пикселге коюңуз
экран = pygame.display.set_mode((800, 600), 0, 32)
# Биздин терезеге тема жазуу орнотуңуз
pygame.display.set_caption("Super Sidekick: Sophie the Bulldog!")
# Экраныбызды / терезебизди көккө боёңуз
экран.fill(көк)
# Биздин сүрөтүбүздү коюу үчүн бир бет түзүңүз
фигура = pygame.Rect(100,100, 200, 200)
# биздин сүрөттү жүктөө үчүн объект түзүү
бульдог = pygame.image.load('SophieTheBullDog.jpg')
# Сүрөтүбүздүн көлөмүн өзгөртүп, аны боёй турган болдук
эскиз_бульдог = pygame.transform.scale(бульдог, (200,200))
# Экранга сүрөттү көчүрүү же боё
экран.blit(эскиз_бульдог, фигура)
# Азыр көк терезени экранга тартыңыз (жаңылаңыз)
pygame.display.update()
# Оюндун аякташы керекпи же жокпу деген маанини сактоо үчүн өзгөрмө түзүңүз
иштөө = True
# Колдонуучу чыгууну чечкенге чейин оюндун иштей турган циклин түзүңүз
# Ал чечкенде, иштетүүнүн маанисин False өзгөртүп, оюнду бүтүрөт
while True:
    # Оюнчудан окуялар тууралуу пикир алуу
        for аракет in pygame.event.get():
            # Эгерде оюнчу кызыл "x" баскычын басса, анда ал оюндан чыгуу болуп эсептелет
            if аракет.type == QUIT:
                pygame.quit()
                sys.exit()
```

Кодду сактап, аны иштетиңиз. Сиздин натыйжаңыз меникине караганда башкача көрүнөт. Анткени мен сизден башка сүрөттү колдонуп жатам. Бирок сиздин натыйжаңыз 11-3-сүрөттө окшош көрүнүшү керек.



Сүрөт 11-3. *Pygame оюн терезесине сүрөт кошуу*

Улантуудан мурун, XY координаттарынын кандай иштей тургандыгын билүү үчүн фигура тик бурчтугунун бетинин XY координаттарын өзгөртүүнү унутпаңыз. Мисалы, сапты өзгөртүңүз:

```
фигура = pygame.Rect(100,100, 200, 200)
```

координатын төмөнкүдөй өзгөртүңүз

```
фигура = pygame.Rect(200,200, 200, 200)
```

Биздин pygame оюн терезебизге текст кошуу

Биздин оюнга сүрөттөрдү кошуу сонун. Бирок текст жөнүндө эмне айтууга болот? Биз текстти дагы кошо алабыз. Бул бөлүмдө биз так ошондой кылабыз.

Pygame оюн терезесине текст кошуу сүрөттөрдү кошууга окшош процесс. Башкача айтканда, аларды тартуу үчүн алгач бетин түзүшүңүз керек. Андан кийин ага текстти бөлбөй туруп, сиз ошол беттин терезеде кайда пайда болорун көрсөтүңүз.

Төмөнкү кодду pygameExample.py файлынын алдына, биз экран.fill(көк) колдонгон жердин астына кошуңуз:

```
# Текстке сиздин шрифтиңизди даярдаңыз
шрифт = pygame.font.SysFont('None', 40)

# Текст объектисин түзүү
текст = шрифт.render("Sophie the Bulldog", True, кызыл, көк)

# Биздин текстти жана анын ордун аныктоо үчүн бет түзүңүз
текстАянты = текст.get_rect()
текстАянты.left = 100
текстАянты.top = 75

# биздин текстти терезеге жайгаштыруу
экран.blit(текст, текстАянты)
```

Ошондой эле, биздин текст менен колдоно турган кызыл деген жаңы түстү аныктайбыз. Бул текстти көк түсүн аныктаган жерге коюңуз:

```
кызыл = (255, 0, 0)
```

Кодду учурдагы оңдоолор менен көрсөтөм. Ошондуктан алардын акыркы версияларын түшүндүргөндөн кийин аларды салыштырып көрсөңүз болот.

Баштоо үчүн биз шрифтибизди сактоо үчүн объект түздүк. Аны түзгөндөн кийин тексттик объектибизге колдонобуз. Биз муну сапта жасайбыз:

```
шрифт = pygame.font.SysFont('None', 40)
```

Pygame.font.SysFont() аргументтери - 'None' жана 40. Биринчи аргумент Pygame модулуна кандай шрифт колдонуу керектигин билдирет. Биз "Arial" сыяктуу шрифт аталышын колдонсок болмок. Бирок мен Pygame модулуна 'None' баскычын тандап, анын демейки системанын шрифтин колдонууга уруксат бердим. Аргумент 40 Pygame биздин текстти которууда (же тартууда) кандай өлчөмдө шрифт колдонушу керектигин айтат.

Андан кийин биз тексттик объектибизди түзөбүз:

```
текст = шрифт.render("Sophie The Bulldog", True, кызыл, көк)
```

Бул мисалда шрифт.render() командасынын эки аргументи бар. Биринчиси, биз экранга кандай текстти басып чыгарууну каалайбыз. Экинчи аргумент - *True* – Pygame модулуна текстиниздин өзгөрүүгө/лакап ат коюуга каршы болушун каалап-каалабаганыңызды айтып берет. Бул анын бир калыпта болушун каалайсызбы же каалабайсызбы дегенди билдирет. *True* бир калыпта дегенди билдирет, *False* бир калыпта эмес дегенди билдирет.

Үчүнчү параметр (кызыл) - бул тексттин биз каалаган түсү. Ал программанын башында жарыялаган түс кортежибиздин негизинде түзүлгөн. Төртүнчү жана акыркы аргумент - биздин тексттин фонунун түсүн билдирет. Терезебиздин түсүнө дал келип, айкалышып турушу үчүн аны көк кылып койдум.

Андан кийин, биз тексттик объектини басып чыгара турган төрт бурчтук бетти аныктайбыз. Андан кийин биз сүрөтүбүздүн кайда пайда болорун тандагандай эле, тексттик объектини басып чыгара турган төрт бурчтук бет кандай жана кайда болорун аныктайбыз.

текстАянты.left = 100 Pygame модулуна экрандын сол жагынан 100 пикселдик тик бурчтуктун бетин тартууну айтат. текстАянты.top = 75 Pygame модулуна тик бурчтуктун бетин экрандын жогору жагынан 75 пиксель төмөн түшүрүүнү айтат.

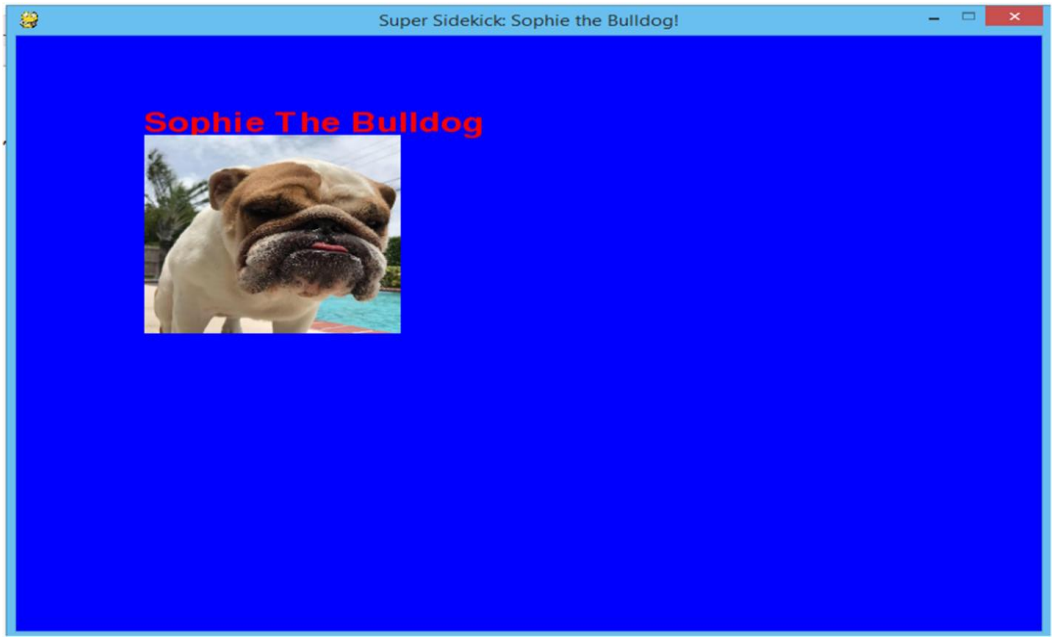
Мурда тартылган сүрөтүбүз текстти жайгаштырган жерге салыштырмалуу кайда экендигин унутпоо керек.

Мисалы, эсиңизде болсо керек, биздин сүрөт жогорудан 100 пиксель төмөн, сол жактан 100 пиксель аралыкта болгон.

Биздин тексттин бетин сол тараптан 100 пиксель аралыкка орнотуп, анын сүрөтүбүзгө туура келгендигин камсыздайбыз. Текстибиздин жогору жагын 75 пикселге орноттук, ошондо ал биздин сүрөтүбүздүн жогору жагында жайгашкан.

Акырында, биз текстти экранга боёп берүү үчүн экран.blit (текст, текстАянты) командасын колдонобуз.

Менин сүрөтүм жаңы кодду колдонгондон кийин кандайча көрүнөт? Сиздики 11-4-сүрөткө окшош болушу керек.



Сүрөт-1-4. *Pygame оюн терезебизге текст кошуу*

Мында сүрөттү жана текстти кошкондон кийин коддун учурдагы нускасы. Кодуңуз меники менен дал келгенин текшериңиз:

```
import pygame
from pygame.locals import *
import sys

# Кийинчерээк экраныбызды көк, текстти кызыл түскө боёгондой кылып,
# RGB (Кызыл, Жашыл Көк) маанилерин сактоо үчүн кортеж түзүү
көк = (0, 0, 255)
кызыл = (255, 0, 0)

#  Pygame модулдарынын бардыгын баштаңыз, ошондо аларды кийинчерээк #колдоно алабыз
pygame.init()

# Оюн экранын түзүп, 800 x 600 пикселге коюңуз
экран = pygame.display.set_mode((800, 600), 0, 32)

# Биздин терезеге тема жазуу орнотуңуз
pygame.display.set_caption("Супер Шерик: Софи Бульдог!")

# Экраныбызды/терезебизди көккө боёңуз
экран.fill(көк)

# Текстке сиздин шрифтиңизди даярдаңыз
шрифт = pygame.font.SysFont('None', 40)

# Текст объектисин түзүү
текст = шрифт.render("Sophie The Bulldog", True, кызыл, көк)

# Биздин текстти жазуу жана анын орду үчүн бет түзүңүз
текстАянты = текст.get_rect()
текстАянты.left = 100
текстАянты.top = 75

# биздин текстти терезеге көчүрүңүз
экран.blit(текст, текстАянты)

# Биздин сүрөтүбүздү кармап туруу үчүн бир бет түзүңүз
фигура = pygame.Rect(100,100, 200, 200)

# биздин сүрөттү жүктөө үчүн объект түзүү
бульдог = pygame.image.load('SophieTheBullDog.jpg')
```

```
# Сүрөтүбүздүн көлөмүн өзгөртүп, андагы сүрөтүбүздү көчүрүп, боёй турган болдук
эскиз_бульдог = pygame.transform.scale(бульдог, (200,200))

# көчүрүү же сүрөтүн боё
экран.blit(эскиз_бульдог, фигура)

# Көк терезени экранга тартыңыз (жаңылаңыз)
pygame.display.update()

# Оюндун аякташы керекпи же жокпу деген маанини сактоо үчүн өзгөрмө түзүңүз
иштөө = True
# Колдонуучу чыгууну чечкенге чейин оюн иштей турган цикл түзүңүз
# Аны аткарганда, иштетүү False маанисине өзгөрүп, оюнду бүтүрөт
while True:
    # Оюнчудан окуялар боюнча пикир алуу
    for аракет in pygame.event.get():
        # Эгерде оюнчу кызыл "x" баскычын басса, анда ал оюндан чыгуу болуп эсептелет
        if аракет.type == QUIT:
            pygame.quit()
            sys.exit()
```

Pygame модулунда фигураларды тартуу

Pygame оюндарыңызга сүрөттөрдү жана спрайттарды киргизүү - декорацияларды, каармандарды жана нерселерди кошуунун эң сонун жолу. Бирок оюн жаратууда бул сиздин жалгыз гана мүмкүнчүлүгүңүз эмес. Ошондой эле сиздин жөнөкөй коддорду колдонуп фигураларды тартуу мүмкүнчүлүгүңүз бар.

Программабызга дагы бир нече түстөрдү кошуудан баштайлы. Мурунку түстөрдү аныктаган жердин астына төмөнкү кодду кошуңуз:

```
кызгылт = (255,200,200)
жашыл = (0,255,0)
кара = (0,0,0)
ак = (255,255,255)
сары = (255,255,0)
```

Андан кийин биз алгачкы бир нече фигураларыбызды чиебиз. Биз үч тегеректин сүрөтүн тартабыз. Алардын ар бири бири-биринен өзгөчө жол менен

айырмаланат. Pygame.display.update() сабынан мурун файлыңызга төмөнкү кодду кошуңуз:

```
# Айлана тартуу
pygame.draw.circle(экран, кызыл, (330, 475), 15, 1)
pygame.draw.circle(экран, сары, (375, 475), 15, 15)
pygame.draw.circle(экран, кызгылт, (420, 475), 20, 10)
```

.draw.circle методу бир нече аргументти камтыйт. Биринчиден, тегеректи кайсы бетке түшүрөрүбүздү аныктайт. Мында биз аны беттеги объектти камтыган жогорудагы *экран* өзгөрмөсүнө тартабыз.

Кийинки аргумент - бул кызыл аркылуу аныкталган түс. Андан кийин программага тегеректин кайсы пиксель аралыкта же XY координатасында жайгашышын каалаганыбызды айтабыз. Бул учурда, айлананын борбору жайгашкан жерге маани беребиз.

Акыркы эки аргумент биздин айлананын радиусун, бул мисалда 15, жана сызыктын калыңдыгын белгилейт.

Бул жерде белгилей кетчү эң кызыктуу нерсе - эгерде биз биринчи тегеректи жараткандан кийин эле программаны иштетсек, анда түскө боёлбогон тегерекчени көрө алабыз. Себеби, биз акыркы аргументти, сызыктын туурасын, 1ге коюп жатабыз. Эгер биз айлананы толугу менен түскө боегубуз келсе, анда сызыктын калыңдыгын радиуска барабар кылмакпыз.

Мунун мисалын, салыштыруу үчүн, экинчи радиуста тартабыз, анын радиусу - 15 жана калыңдыгы - 15.

Акыры, үчүнчү айлананы чийип, бул жолу анын калыңдыгын радиусунун жарымына чейин азайтып, кандай болорун көрө алабыз. Болжол менен, биздин акыркы айланабыз май нанга окшош болот деп айткым келет. Менин туура экенимди тактап көрөлү. Программаны сактап, аны иштетиңиз. Сиз 11-5-сүрөттө окшош натыйжаны көрүшүңүз керек.

Сиздин натыйжаңыз меникине окшош болушу керек.

Бул жерде сиз тарта турган ар кандай фигуралардын бир нече мисалдары бар (аларды файлыңызга кошпой эле коюңуз):

- Айлана : pygame.draw.lines (бети, түсү, (x, y), радиусу, жоондугу)

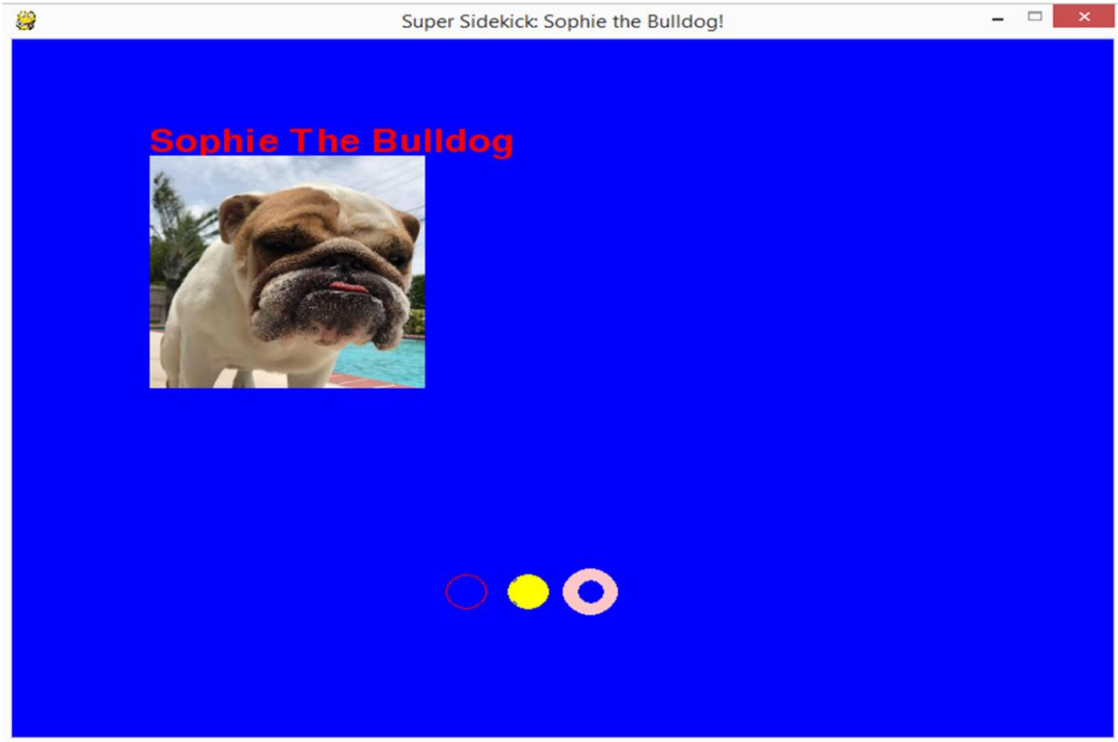
  Мисалы: pygame.draw.circle(экран, сары, (375, 475), 15, 15)

- Тик бурчтук: pygame.draw.rect (бети, түсү, (x, y, туурасы, бийиктиги), жоондугу)

  Мисалы: pygame.draw.rect(экран, сары, (455, 470, 20, 20), 4)

- Сызык: pygame.draw.line (бети, түсү, (X, Y координаттары үчүн.) Саптын башталышы), (саптын аягында X, Y координаттары), жоондугу)

Мисалы: pygame.draw.line(экран, кызыл, (300, 500), (500,500),1)

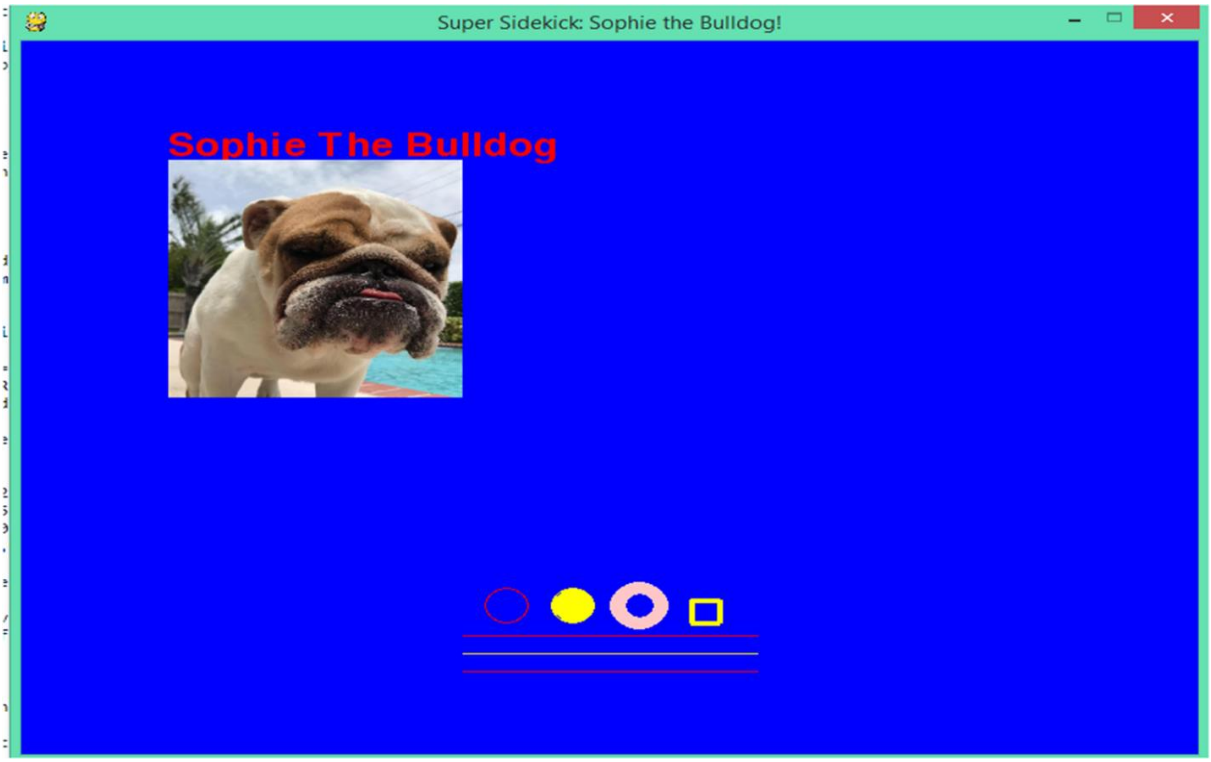


Сүрөт -11-5. *Pygame оюн терезебизге фигураларды тартуу*

Алга жөнөңүз жана биздин айланалардын кодунун астына файлыңызга төмөнкү код саптарын кошуңуз:

```
pygame.draw.rect(экран, сары, (455, 470, 20, 20), 4)
pygame.draw.line(экран, кызыл, (300, 500), (500,500),1)
pygame.draw.line(экран, сары, (300, 515), (500,515),1)
pygame.draw.line(экран, кызыл, (300, 530), (500,530),1)
```

Эгер сиз ушул кодду иштетсеңиз, анда сиздин натыйжаңыз 11-6-сүрөткө окшош болот:



Сүрөт -11-6 *Биздин Pygame оюн терезесине айрым саптарды кошуу*

Кошумча окуяларды кошуу

Колдонуучуга жооп бербесе, оюндан эмне пайда болмок? Ошондой эле, кайсы программа колдонуучуларга экрандын жогорку оң бурчундагы кызыл "X" баскычын басуу менен гана чыгууга мүмкүнчүлүк берет, бул өтө тууралай көрүнбөйт, ошондойбу?

Pygame программалары ар кандай аракеттерге жооп берүүгө жөндөмдүү. Бул аракеттер - чычканды басуу, клавиатурадагы жебе баскычын же клавиатурадагы кандайдыр бир баскычка басуу. Биз булардын бир нечесин гана атадык.

Биздин pygameExample.py файлынан алыстаардан мурун окуялардын иштешин жакшыраак сезишибиз үчүн, дагы бир нече окуяларды программага кошолу.

Эгер сиз эч нерсени калтырбай үйрөнүп жүргөн болсоңуз, pygameExample.py кодуңуз төмөнкүлөргө дал келиши керек. Эгер андай болбосо, анда анын аткарылышын камсыз кылуу үчүн бир аз убакыт бөлүңүз:

```
import pygame
from pygame.locals import *
import sys
```

```
import random

# Кийинчерээк экраныбызды көк, текстти кызыл түскө боёгондой кылып,
# RGB (Кызыл, Жашыл Көк) маанилерин сактоо үчүн кортеж түзүү
көк = (0, 0, 255)
кызыл = (255, 0, 0)
кызгылт = (255,200,200)
жашыл = (0,255,0)
кара = (0,0,0)
ак = (255,255,255)
сары = (255,255,0)
# Pygame модулдарынын бардыгын баштаңыз, ошондо аларды кийинчерээк колдоно алабыз
pygame.init()

# Оюн экранын түзүп, 800 x 600 пикселге коюңуз
экран = pygame.display.set_mode((800, 600), 0, 32)

# Биздин терезеге коштомо жазуу орнотуңуз
pygame.display.set_caption("Super Sidekick: Sophie the Bulldog!")
# Экраныбызды / терезебизди көк түскө боёңуз
экран.fill(көк)

# Текстке сиздин шрифтиңизди даярдаңыз
шрифт = pygame.font.SysFont('None', 40)

# Текст объектисин түзүү
текст = шрифт.render("Sophie The Bulldog", True, кызыл, көк)

# Биздин текстти жазуу үчүн бет түзүңүз
текстАянты = текст.get_rect()
текстАянты.left = 100
текстАянты.top = 75

# биздин текстти терезеге көчүрүңүз
экран.blit(текст, текстАянты)

# Биздин сүрөтүбүздү кармап туруу үчүн бир бет түзүңүз
фигура = pygame.Rect(100,100, 200, 200)

# биздин сүрөттү жүктөө үчүн объект түзүү
бульдог = pygame.image.load('SophieTheBullDog.jpg')

# Сүрөтүбүздүн көлөмүн өзгөртүп, андагы сүрөтүбүздү көчүрүп, боёй турган болдук
```

```
эскиз_бульдог = pygame.transform.scale(бульдог, (200,200))

# Көчүрүү же экранга сүрөтүн тартуу
экран.blit(эскиз_бульдог, фигура)

# Фигураларды тартуу
pygame.draw.circle(экран, кызыл, (330, 475), 15, 1)
pygame.draw.circle(экран, сары, (375, 475), 15, 15)
pygame.draw.circle(экран, кызгылт, (420, 475), 20, 10)
pygame.draw.rect(экран, сары, (455, 470, 20, 20), 4)
pygame.draw.line(экран, кызыл, (300, 500), (500,500),1)
pygame.draw.line(экран, сары, (300, 515), (500,515),1)
pygame.draw.line(экран, кызыл, (300, 530), (500,530),1)

# Азыр көк терезени экранга тартыңыз (жаңылаңыз)
pygame.display.update()

# Оюн аякташы керекпи же жокпу деген маанини сактоо үчүн өзгөрмө түзүңүз
иштөө = True

# Колдонуучу чыгууну чечкенге чейин оюн иштей турган цикл түзүңүз
# Аны аткарганда, иштетүү маанисин False деп өзгөртүп, оюнду бүтүрөт

while True:
      # Оюнчудан окуялар тууралуу пикир алуу
      for аракет in pygame.event.get():
            # Эгерде оюнчу кызыл "x" баскычын басса, анда ал оюндан чыгуу болуп
            эсептелет
            if аракет.type == QUIT:
                  pygame.quit()
                  sys.exit()
```

Окуяларыбызга кошула турган коддун бир бөлүгү биздин билимибизди жаңыртуу үчүн биздин оюн циклибиз болуп саналат:

```
# Оюндун аякташы керекпи же жокпу деген маанини сактоо үчүн өзгөрмө
түзүңүз
иштөө = True

# Колдонуучу чыгууну чечкенге чейин оюн иштей турган цикл түзүңүз
# Аны аткарганда, иштетүү маанисин False деп өзгөртүп, оюнду бүтүрөт

while True:
# Эгерде оюнчу кызыл "x" баскычын басса, анда ал оюндан чыгуу болуп
```

```
# эсептелет
    for аракет in pygame.event.get():
        # Эгерде оюнчу кызыл "x" баскычын басса, анда ал оюндай чыгуу
        #болуп эсептелет
        if аракет.type == QUIT:
            pygame.quit()
            sys.exit()
```

Оюн циклдери жүрүп жатканда бул абдан жөнөкөй көрүнөт. Белгиленгендей, анын бир эле окуясы бар. Биринчи кезекте колдонуучунун тиркемеден чыгуусунун дагы бир жолун кошуу керек. Бул үчүн эки ыкманы колдонобуз. Биринчиден, колдонуучу клавиатурасында 'q' баскычые басса, анда колдонмо жабылат. Экинчиден, колдонуучу ESC баскычын басса да оюн жабылат. Биздин код жаңыртылгандан кийин, колдонуучу расмий түрдө биздин тиркемеден чыгуунун үч жолуна ээ болот.

Коддун оюн цикл бөлүгүн төмөнкүлөргө дал келгидей кылып өзгөртүңүз.

Эскертүү: Тийиштүү чегинүүнү унутпаңыз!

```
# Оюн аякташ керекпи же жокпу деген маанини сактоо үчүн өзгөрмө түзүңүз
иштөө = True
# Колдонуучу чыгууну каалаганга чейин оюн иштей турган цикл түзүңүз
# Аны аткарганда, иштетүү маанисин False деп өзгөртүп, оюнду бүтүрөт
while True:
# Эгерде оюнчу кызыл "x" баскычын басса, анда ал оюндан чыгуу болуп
эсептелет
    for аракет in pygame.event.get():
        # Эгерде оюнчу кызыл "x" баскычын басса, анда ал оюндан чыгуу
        болуп эсептелет
        if аракет.type == QUIT:
            pygame.quit()
            sys.exit()
        if аракет.type == pygame.KEYDOWN:
            if аракет.key == pygame.K_q:
                pygame.quit()
                sys.exit()
        if аракет.type == pygame.KEYDOWN:
            if аракет.key == pygame.K_ESCAPE:
                pygame.quit()
                sys.exit()
```

Кодду сактап, программаңызды бир нече жолу иштетип көрүңүз. Программаны кайра иштетип жатканда "q" баскычын басып, кызыл "X" баскычын басып, ESC баскычын басып, ар бир чыгуу параметринин иштешине ынаныңыз.

Жаңы коддо эки жаңы окуянын түрү бар экендигин эске алыңыз. Биринчиси - колдонуучу клавиатурасындагы баскычты басышын күтүп жатканда колдонулган pygame. KEYDOWN.

Биздин pygame.KEYDOWN окуясынан кийин аракет.key чегинүүсү бар, ал программанын *так* ачкычын кандай кабыл аларын аныктайт. Клавиатурадагы тамгалар менен сандардын көпчүлүгү pygame.K_, андан кийин тамга же сан киргизүү менен аныкталат.

Мисалы, "a"ны көрүү үчүн, сиз төмөнкүлөрдү колдонуңуз:

```
if аракет.type == pygame.KEYDOWN:
    if аракет.key == pygame.K_a:
       бир нерсе жасаңыз...
```

www.pygame.org/docs/ref/key.html шилтемесине кирип клавиатуранын туруктуу тизмесин толугу менен көрө аласыз. Мындан сырткары, кеңири тараган клавиатура константтарынын өзгөрбөс чоңдуктарынын тизмеси бар. Алар:

- Жогору жебе: K_UP;
- Төмөнкү жебе: K_DOWN;
- Оң жебе: K_RIGHT;
- Сол жебе: K_LEFT;
- Бош орун: K_SPACE;
- Кирүү же кайтуу: K_RETURN;
- Сандар: K_0, K_1, K_2 ж.б.;
- Тамгалар: K_a, K_b, K_c, K_d ж.б.

Кийинки бөлүмгө өтүүдөн мурун pygameExample.py файлын дагы бир аз толуктап коёлу. Дагы бир окуяны - клавиатуранын " b " тамгасын көрөлү. Колдонуучу ушул баскычты басканда программа текстти экранда көрсөтөт.

Бул нерсени ишке ашыруу үчүн оюн циклибизге төмөнкүнү, биздин акыркы if блогубуздун төмөн жагында кошолу:

```
if аракет.type == pygame.KEYDOWN:
    if аракет.key == pygame.K_b:
        үрүү = шрифт.render("Гав-гав!", True, кызыл, көк)
        үрүүАянты = үрүү.get_rect()
        үрүүАянты.left = 300
        үрүүАянты.top = 175
```

```
            экран.blit(үрүү, үрүүАянты)
            pygame.display.update()
```

Ушул убакка чейин, сиз бул коддун кандай жумуш аткарарын жакшы түшүнүп калсаңыз керек. Эгер андай болбосо, анда эч нерсе эмес. Биз аны этап-этабы менен карап чыгабыз.

Алгач биз KEYDOWN окуясынын түрүн карайбыз. Башкача айтканда, кимдир бирөө клавиатурасында баскычты басат. Андан кийин Pygame модулуна кайсы баскычты көрүп жатканыбызды айтып беребиз:

```
if аракет.key == pygame.K_b:
```

Бул сап биз "b" баскычын издеп жаткандыгыбызды билдирет. KEYDOWN окуясы менен KEYUP окуясынын айырмасын белгилей кетүү маанилүү. Белгиленгендей, KEYDOWN окуясы колдонуучу өзүнүн клавиатурасында көрсөтүлгөн баскычты басканда пайда болот; KEYUP окуясы ушул баскычты коё бергенде пайда болот. Эгер KEYUP окуясы байкалбаса, колдонуучу ачкычты коё бергенден кийин эч нерсе болбойт.

Андан кийин колдонуучу b баскычын басса эмне болорун аныктайбыз. Алгач, үрүү деген текст объектиси түзүлөт. Текст объектиси үчүн аргументтерди белгилейбиз. Анда текст эмне деш керек, тексттин түсү жана тексттин артындагы фонунун түсү кандай болорун айтат.

Кийинки, сапта:

```
үрүүАянты = үрүү.get_rect()
```

Биз текстибиз турган бетти аныктайбыз. Ошол жерден биз төмөнкүлөрдү колдонуп, беттин абалын көрсөттүк:

```
үрүүАянты.left = 300
үрүүАянты.top = 175
```

Акырында биз текст объектисин жана анын бетин экрандын үстүнө бөлүп, жаңы түзүлгөн текстти көрсөтүү үчүн дисплейин жаңыртабыз.

Эгерде сиз ушул кодду сактап, иштетип көрсөңүз, анда колдонмо жүктөлгөндөн кийин 'b' баскычын басуу менен, 11-7-сүрөткө окшош жыйынтыкка ээ болосуз.

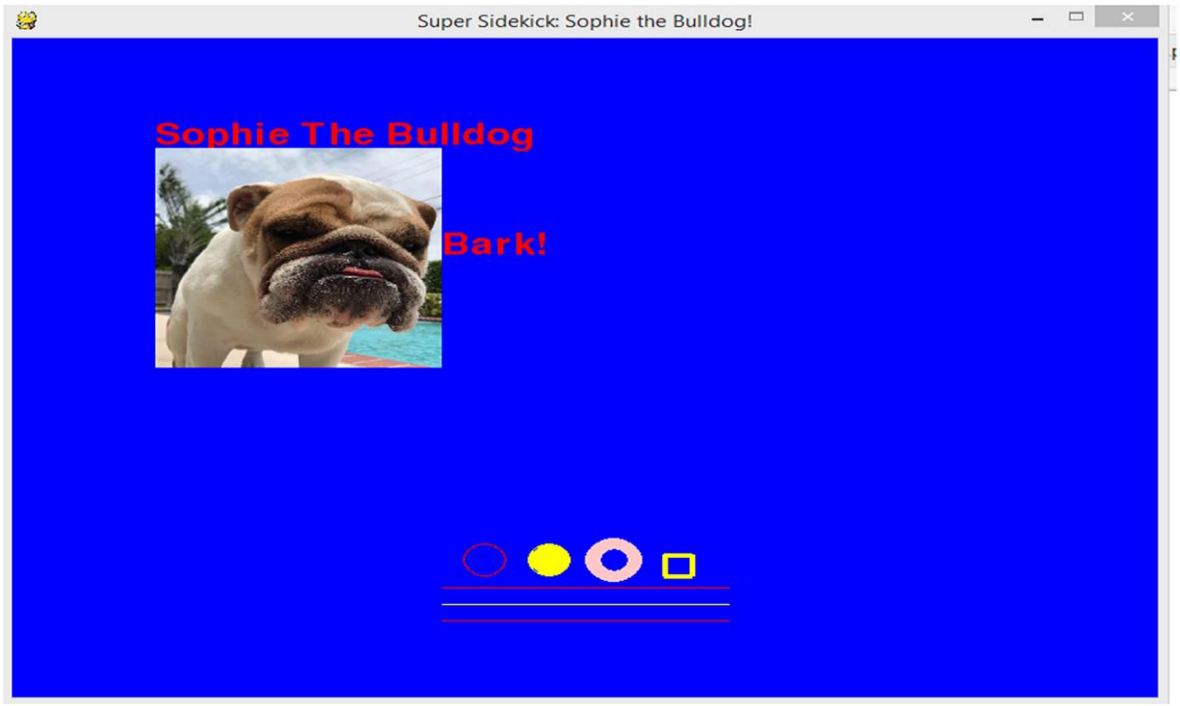
"b" баскычы окуясындагы "Bark!" – "Үрүү!" тексти биздин оюн циклибизге жайгаштырылгандыктан, техникалык жактан "b" баскычын баскан сайын текст экранга кайра жүктөлөт. Бирок, сиз муну көрө албайсыз. Анткени "Bark!" текстин алмаштыруу бат, көлөмү, формасы жана түсү бирдей.

Колдонуучу 'b' баскычын баскан сайын тексттин пайда болушунун бир катар жолдору бар. Эң жөнөкөй ыкмалардын бири - иллюзияны колдонуу. Арамза *Математик* сыймыктана турган нерсе!

Бул иллюзияны алып салуу үчүн биз дагы бир баскыч окуясын кошкону жатабыз. Ачкыч дагы деле "b" болот. Бирок KEYDOWN окуясынын ордуна биз KEYUP окуясын кошобуз.

Коддун кийинки бөлүгүнүн идеясы жөнөкөй: колдонуучу 'b' баскычын коё бергенден кийин "Bark!" жок болот.

Чындыгында, биз жөн гана "Bark!" деген сөздүн түсүн фонддун түсүндөй кылып өзгөртөбүз. Ал жок болуп кеткендей көрүнөт. Бирок ал жөн гана фондун артында жашынып калат.



Сүрөт - 11-7. *“Sophie The Bulldog” үрдү!*

Колдонуучу 'b' баскычын кайра басканда түс дагы бир жолу кызылга айланып, кайрадан көрүнүп калат. Бул цикл колдонуучу колдонмодон чыккыча 'b' баскычын баскан сайын улана берет.

Бул кодду акыркы KEYDOWN окуяңызды аныктаган кодго кошуңуз. Сактап, андан кийин программаны иштетиңиз. "Софи үрөт - Sophie Bark!" көргөндөн тажаганча ‘b’ баскычын бир нече жолу басканды унутпаңыз!

Эскертүү: Сиздин биринчи if операторуңуз мурунку if оператору менен дал келиши үчүн, сиз туура чегинип жатканыңызды аныктаңыз.

```
if аракет.type == pygame.KEYUP:
      if аракет.key == pygame.K_b:
          үрүү = шрифт.render("Bark!", True, көк, көк)
          үрүүАянты = үрүү.get_rect()
          үрүүАянты.left = 300
          үрүүАянты.top = 175
          экран.blit(үрүү, үрүүАянты)
          pygame.display.update()
```

Эгер сиздин кодуңуз иштебей жатса, анда анын төмөнкү код менен дал келишин текшерип, убакыт бөлүңүз. Биздин акыркы чыгарылыштарыбыз менен pygameExample.py толук коду:

```
import pygame
from pygame.locals import *
import sys
import random

# Кийинчерээк экраныбызды көк, текстти кызыл түскө боёгондой кылып,
# RGB (Кызыл, Жашыл Көк) маанилерин сактоо үчүн кортеж түзүү
көк = (0, 0, 255)
кызыл = (255, 0, 0)
кызгылт = (255,200,200)
жашыл = (0,255,0)
кара = (0,0,0)
ак = (255,255,255)
сары = (255,255,0)

# Pygame модулдарынын бардыгын баштаңыз, ошондо аларды кийинчерээк
#колдоно алабыз
pygame.init()

# Оюн экранын түзүп, 800 x 600 пикселге коюңуз
экран = pygame.display.set_mode((800, 600), 0, 32)

# Биздин терезеге коштомо жазуу орнотуңуз
pygame.display.set_caption("Super Sidekick: Sophie the Bulldog!")

# Экраныбызды / терезебизди көккө боёңуз
экран.fill(көк)
```

```
# Текстке сиздин шрифтиңизди даярдаңыз
шрифт = pygame.font.SysFont('None', 40)

# Текст объектисин түзүү
текст = шрифт.render("Sophie the Bulldog!", True, кызыл, көк)

# Биздин текстти жана анын ордун аныктоо үчүн бет түзүңүз
текстАянты = текст.get_rect()
текстАянты.left = 100
текстАянты.top = 75

# биздин текстти терезеге көчүрүңүз
экран.blit(текст, текстАянты)

# Биздин сүрөтүбүздү кармап туруу үчүн бир бет түзүңүз
фигура = pygame.Rect(100,100, 200, 200)

# биздин сүрөттү жүктөө үчүн объект түзүү
бульдог = pygame.image.load('SophieTheBullDog.jpg')

# Сүрөтүбүздүн көлөмүн өзгөртүп, андагы сүрөтүбүздү көчүрүп, боёй турган
болдук
эскиз_бульдог = pygame.transform.scale(бульдог, (200,200))

# көчүрүү же экранда сүрөтүн боё
экран.blit(эскиз_бульдог, фигура)

# Фигураларды тартуу
pygame.draw.circle(экран, кызыл, (330, 475), 15, 1)
pygame.draw.circle(экран, сары, (375, 475), 15, 15)
pygame.draw.circle(экран, кызгылт, (420, 475), 20, 10)
pygame.draw.rect(экран, сары, (455, 470, 20, 20), 4)
pygame.draw.line(экран, кызыл, (300, 500), (500,500),1)
pygame.draw.line(экран, сары, (300, 515), (500,515),1)
pygame.draw.line(экран, кызыл, (300, 530), (500,530),1)

# Азыр көк терезени экранга тартыңыз (жаңылаңыз)
pygame.display.update()

# Оюн аякташ керекпи же жокпу деген маанини сактоо үчүн өзгөрмө түзүңүз
иштөө = True

# Колдонуучу чыгууну каалаганга чейин оюн иштей турган цикл түзүңүз
```

```
# Аны аткарганда, иштетүү маанисин False деп өзгөртүп, оюнду бүтүрөт

while True:

# Эгерде оюнчу кызыл "x" баскычын басса, анда ал оюндай чыгуу болуп эсептелет
      for аракет in pygame.event.get():
          # Эгерде оюнчу кызыл "x" баскычын басса, анда ал оюндай чыгуу болуп эсептелет
          if аракет.type == QUIT:
                pygame.quit()
                sys.exit()
          if аракет.type == pygame.KEYDOWN:
               if аракет.key == pygame.K_q:
                     pygame.quit()
                     sys.exit()
          if аракет.type == pygame.KEYDOWN:
               if аракет.key == pygame.K_ESCAPE:
                     pygame.quit()
                     sys.exit()
          if аракет.type == pygame.KEYDOWN:
               if аракет.key == pygame.K_b:
                    үрүү = шрифт.render("Bark!", True, кызыл, көк)
                    үрүүАянты = үрүү.get_rect()
                    үрүүАянты.left = 300
                    үрүүАянты.top = 175
                    экран.blit(үрүү, үрүүАянты)
                    pygame.display.update()
          if аракет.type == pygame.KEYUP:
              if аракет.key == pygame.K_b:
                    үрүү = шрифт.render("Гав-гав!", True, көк, көк)
                    үрүүАянты = үрүү.get_rect()
                    үрүүАянты.left = 300
                    үрүүАянты.top = 175
                    экран.blit(үрүү, үрүүАянты)
                    pygame.display.update()
```

Бул эпизоддо

Ой, кандай гана кызыктуу! Бирок акылга сыйбаган бөлүм! Эгерде сиз бул бөлүмдү башыңызды столго ургулап, майда жарааттар жана тактар менен өткөн болсоңуз, анда жакшы! Бул бөлүмдө камтылган темаларды өздөштүрүү эң кыйын болгон болушу мүмкүн. Жок дегенде алар класстар жана обьектилер сыяктуу татаал болуп, Python тилинен үйрөнө турган эң кыйын нерселердин бири болушу мүмкүн.

Мыкты жумуш!

Бирок жетишилген ийгиликтер менен азырынча эс алып отуруп калбаңыз. Кийинки бөлүм Pygame оюнун талкуулоону улантат жана өзүңүздүн оюндарыңызды жаратуунун дагы эки татаал, бирок күчтүү жана пайдалуу аспектилерине: анимация жана кагылышууну аныктоого арналат. Эгер сиз оюндун иштеп чыгуучусу болгуңуз келсе же программалоодогу кыйынчылыкты кааласаңыз, анда кийинки бөлүмдү өткөрүп жиберүүнү каалабасаңыз керек!

Бул бөлүм жана кийинки бөлүм биригип, ушунчалык кенен темалар болгондуктан, биз ар бир бөлүмдүн аягында аткарган кадимки кыскача баяндаманы берип турабыз. Маанилүү сөздөрдү жалпылоо адилеттүүлүккө жатпайт.

Андан көрө, ушул бөлүмдө үйрөнгөн жана кийинки бөлүмдө үйрөнө турган көндүмдөрүңүздү иш жүзүндө колдонуп, керектүү учурларда кайра-кайра окуп чыгыңыз.

Ошондой эле, адат болгон тажрыйба жана да тажырыйбаны уланта берели.

Бул түзмө-түз оюндун аталышы!

12-БӨЛҮМ

ОЮН АНИМАЦИЯЛОО

Жаңы билимге ээ болуу үчүн кайтып келгениңизди көрүп турам. Азаматсыз! Мурдагы бөлүм, чынын айтсам, аябай кызыктуу болду. Оюндарды өнүктүрүүнүн айрым негизги теорияларын жана тажрыйбаларын үйрөнүп гана тим болбостон, кодду өзүңүз жаздыңыз!

Эң негизгиси, сиз Софи менен тааныштыңыз. Ал - уктап жатканда укмуш ит жана сонун үй жаныбары. Албетте, ал бүткүл оюн учурунда уктай берет. Бирок сизге тамактын бардыгын жеп туруп, артынан кекирип койгон бирөө керек болсо, анда андан өткөн жакшы өнөктөш таба албайсыз.

Өткөн бөлүмдө оюндарыбызга фигураларды тартууну жана сүрөттөрдү киргизүүнү үйрөндүк. Ошондой эле, оюн циклдери жана колдонуучунун биздин оюндар менен иштешүүсүнө мүмкүндүк берүүчү окуяларды түзүү жөнүндө билдик.

Бул бөлүмдүн жүрүшүндө биз оюн жасоону өнүктүрүүнүн дагы эки маанилүү аспектисин үйрөнөбүз. Биринчиси – анимация. Башкача айтканда, ал объектилердин экрандагы кыймыл-аракетин көргөзөт. Экинчиси кагылышууну аныктоо деп аталат. Ал - эки же андан көп объектилер бири-бирине тийгенде же сиздин оюн терезеңиздин чек араларына тийгенде боло турган көрүнүш.

Мен сизди узак киришүү кылып тажатпайын. Ооба, кеч болуп калды деп жатпайсызбы?

Анда эмесе баштадык, акылдуум!

Pygame оюнунда анимацияларды түзүү

Оюнду иштеп чыгуунун айрым негизги түшүнүктөрүнүн жана Python программасында pygame модулунун жардамы менен өз оюндарыбызды кантип жаратууну үйрөнүү үчүн узак жолду басып өттүк. Ушул кезге чейин биз өз тарыхыбызды түзүүнү, сүрөттөрдү же спрайттарды кошууну, текстти киргизүүнү жана баскычтарды басуу сыяктуу окуяларды байкоону, андан да маанилүүсү - жооп берүүнү үйрөндүк.

Визуалдык 2D оюнун жаратуунун чыныгы өзөгү анимацияга байланыштуу. Бул жөнүндө кийинки бөлүмдө талкуулайбыз. Бардык Python сыяктуу эле, биздин Pygames модулунда анимация жасоонун көптөгөн жолдору бар. Бирок бул жаңыдан баштагандардын китеби болгондуктан, биз эң жөнөкөй ыкмаларды гана карап көрөлү.

Биздин акыркы тиркеме pygameExample.py абдан чоң файл болуп калды. Башаламандыкка жол бербөө үчүн жана бир аз мейкиндикти үнөмдөө үчүн, келгиле, pygameAnimations.py деген жаңы файлды түзөлү.

Биз кээ бир кодубузду pygameExample.py файлынан кайра иштетебиз. Андыктан айрым коддор бир аз тааныш окшойт деп капаланбаңыз. Эсиңизде болсун: биз ар дайым кодубузду мүмкүн болушунча жана ылайыктуу учурларда кайра колдонууну каалайбыз. Атап айтканда, биздин кээ бир түстөрдүн өзгөрмөлөрү жана модулдун импорттоо аркылуу баштапкы маани берүү бөлүктөрүн кайра колдоно алабыз.

Биз оюндун структурасына, ошондой эле оюн циклине бир аз өзгөртүү киргизебиз. Анимациялар менен иштөө бир аз татаалыраак болгондуктан, мен анын кандайча иштээрин мыкты түшүндүрүп берүү үчүн түз эле файлды ачкым келди. Мындан тышкары, анимация үчүн структуралар статикалык сүрөттөрдөн жана тексттен айырмаланып турат.

Платформаны орнотуу үчүн pygameAnimation.py файлыңызга төмөнкү кодду кошуңуз:

```
# Биздин модулдарды кошуңуз
import pygame
from pygame.locals import *
import sys
import random

# Биздин pygame модулдарын жүктөңүз
pygame.init()

# Биздин түстөр үчүн кортеждерди түзүңүз
ак = (255,255,255)
кара = (0,0,0)
кызыл = (255,0,0)

# Оюндун негизги терезесин түзүңүз - акыркы жолу аны экран деп атаганбыз
# Бул жолу башкача ат берели
оюнТерезеси = pygame.display.set_mode((800,600))

# Биздин анимацияга коштомо жазууну / аталышты коюңуз
pygame.display.set_caption('Box Animator 5000')
```

Бул коддун нускасын мурунку тиркемебизде жазгандыктан, аны дагы бир жолу карап чыгуунун кажети жок. Биздин экранды орнотуучу, сүрөттөрүбүздө жана текстибизде колдонула турган түстөрдү аныктап, импорттогон жана модулдарды жөнгө салуучу скелеттердин коду экендигин билип коюңуз. Ошондой эле терезебиздин жазуусун "Box Animator 5000" деп өзгөрттүк.

Андан кийин, биз дагы бир нече өзгөрмө түзгүбүз келет:

```
оюнданЧыгуу = False

x = 300
y = 300
```

Биринчи өзгөрүлмө – оюнданЧыгуу. Ал биздин оюн циклибиз текшерип көргөн программанын аякташы керекпи же жокпу деген маани сактайт. ОюнданЧыгуу True болбосо, оюн улана берет. Анын мааниси True болуп өзгөрсө гана, оюн бүтөт.

Кийинки эки өзгөрмө - x жана y - биз тарта турган тик бурчтук объектинин баштапкы абалын орнотуу үчүн колдонулат. Бул маанилерди түздөн-түз тик бурчтуктун аргументтеринде аныктоонун ордуна өзгөрмө менен аныктайбыз. Анткени кийинчерээк объектибиздин XY координаттарынын маанисин өзгөртөбүз.

x объектинин X абалын билдирет, ал y анын Y абалы үчүн колдонулат.

Кийинкиде анимацияланган оюндун оюн цикли болот. Биз бир нече жаңы окуяларды коштук. Аларды кеңири талкуулайбыз. Кодуңузга төмөнкүнү кошуңуз:

```
# Оюн цикли
while not оюнданЧыгуу:
    for аракет in pygame.event.get():
        if аракет.type == pygame.QUIT:
            оюнданЧыгуу = True
            pygame.quit()
            sys.exit()
        # Эгерде оюнчу 'q' баскычын басса, анда ал чыгуу аракети деп эсептелет
        if аракет.type == pygame.KEYDOWN:
            if аракет.key == pygame.K_q:
                pygame.quit()
                sys.exit()
        # Эгерде оюнчу 'ESC' баскычын басса, анда ал токтотуу деп эсептелет
        if аракет.type == pygame.KEYDOWN:
            if аракет.key == pygame.K_ESCAPE:
                pygame.quit()
                sys.exit()
        # Эгерде жебе баскычы солго басылса, объектти солго жылдырыңыз
        if аракет.type == pygame.KEYDOWN:
            if аракет.key == pygame.K_LEFT:
```

```
            x -= 10
    # Эгерде жебе баскычы оңго басылса, объектти оңго жылдырыңыз
        if аракет.key == pygame.K_RIGHT:
            x += 10
    # Эгерде жебе баскычы жогору жакка басылса, объектти жогору
    жылдырыңыз
    if аракет.type == pygame.KEYDOWN:
        if аракет.key == pygame.K_UP:
            y -=10
    # Эгерде жебе баскычы төмөн жакка басылса, объектти төмөн
    #жылдырыңыз
        if аракет.key == pygame.K_DOWN:
            y +=10
```

Бул оюн циклинин көпчүлүгү сизге тааныш болушу керек. Бизде колдонуучунун кантип чыгарын сүрөттөгөн бир нече окуялар бар. Алар ESC же "q" же кызыл "X" баскычтарын баса алышат.

Андан кийин биз СОЛ, ОҢ, ЖОГОРУ жана ТӨМӨН жебелери үчүн баскычтарды басуу окуяларын түздүк. Ушул баскычтардын бирин бассаңыз, төмөнкүлөр болот:

- Эгерде СОЛ баскычы басылса, объекти солго 10 пикселге жылып, x мааниси 10го төмөндөйт.
- Эгерде ОҢ баскычы басылса, объекти оңго 10 пикселге жылып, x мааниси 10го көбөйөт.
- ЖОГОРУ баскычы басылса, объекти 10 пикселге көтөрүлүп, y мааниси 10го төмөндөйт.
- ТӨМӨН баскычы басылса, объектини 10 пикселге ылдый жылдып, y мааниси 10го көбөйөт.

Эми биздин оюн циклибиз жана окуяларыбыз ордунда болгондуктан, ишибиздин акыркы бөлүгү болгон терезебизди түзүп, аны түскө толтуруп, формабызды көчүрүп, дисплейди жаңыртабыз:

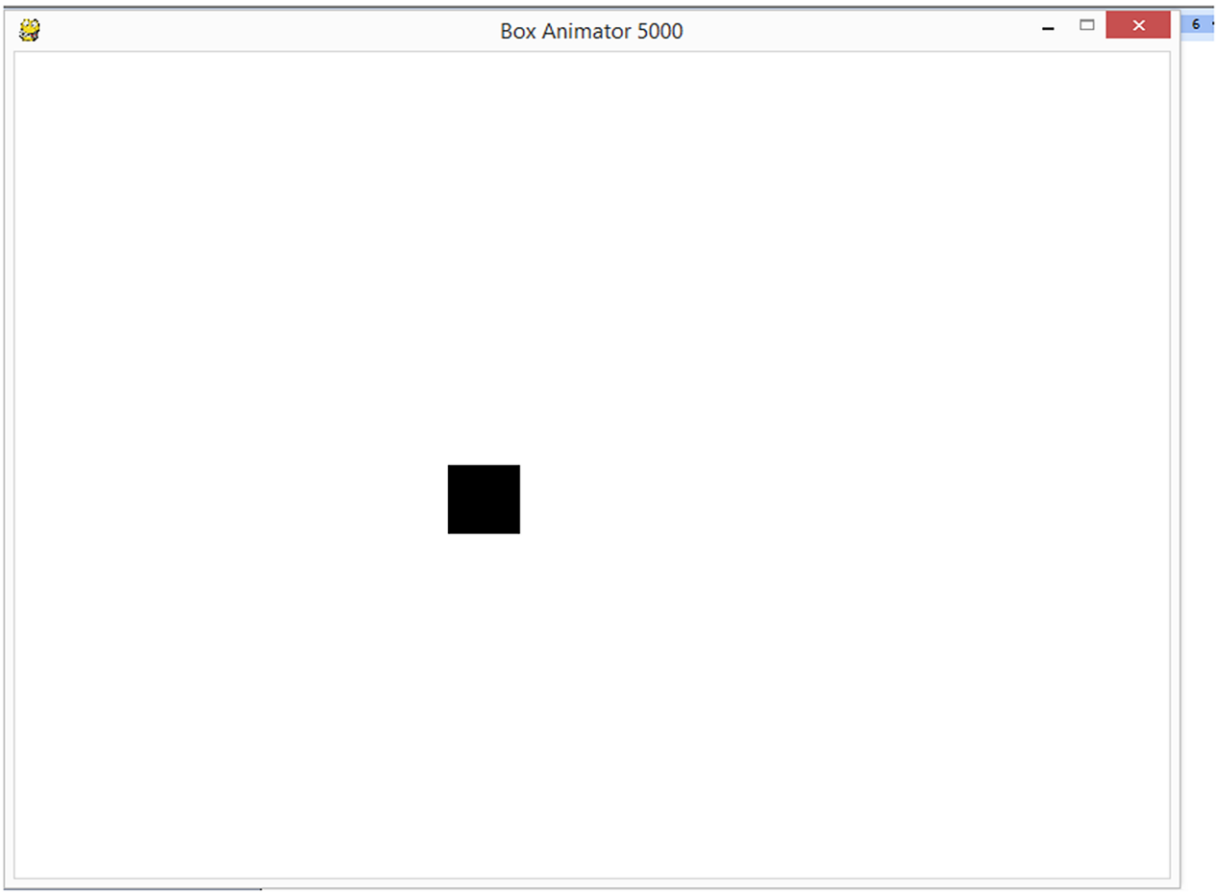
```
# Оюн терезесин (оюнТерезеси) ак түс менен толтуруңуз
оюнТерезеси.fill(ак)

# Кара тик бурчтуу объектини чыгар
pygame.draw.rect(оюнТерезеси, кара, [x,y,50,50])

# Экраныңызды жаңыртыңыз
pygame.display.update()
```

Бул жерде биздин биринчи анимациялык оюн бар! Программаны иштетип, аны сынап көрүңүз. Жебе баскычтарынын ар бирин басып, андан кийин дагы бир нече жолу иштетип, "чыгуу" окуяларынын ар бирин сынап көрүңүз.

Сиздин экран 12-1- сүрөтүнө окшош болушу керек:



12-1-сүрөт. *Чыгуу окуяларын текшерүү*

Бул укмуштай сонун да, туурабы? Бул түрдөгү анимация логикасын ар кандай оюндарга колдонсо болот. Мисалы, сиз автоунаа менен көчөлөрдө боло турган жарыш оюнун, каарман тактада айланып мушташуучу оюнду ж.б. оюндарды жарата аласыз.

Албетте, биздин оюн азыр абдан кызыксыз. Бирок, бул жерде үйрөнө турган негизги түшүнүк объектти жылдыруу.

Бул кодго бир нече нерсе жетишпейт. Бир нерсени байкаган болушуңуз керек, эгер сиз кутучаны кандайдыр тарапка ордунан узак жылдырсаңыз, ал экрандан чыгып, акыры жок болуп кетет. Эгер сиз аны жылдырсаңыз, ал техникалык жактан карама-каршы багытка кайтып келет. Бул биздин оюнда кандайча көйгөйлөрдү жаратарын көрө аласыз.

Муну оңдоонун бир нече жолдору бар. Аларды кийинки бөлүмдө талкуулайбыз. Азырынча тик бурчтукту жылдыруунун дагы бир жолун - кокустук телепортациясын кошолу! Келгиле, супер күчтөр жөнүндө сүйлөшөлү!

Калган окуялардын аягына төмөнкү коддун үзүндүсүн кошуңуз:

```
# эгер 't' баскычы басылса, объектини туш келди телепорттоңуз
if аракет.type == pygame.KEYDOWN:
     if аракет.key == pygame.K_t:
          y = int(random.randint(1,600))
          x = int(random.randint(1,600))
```

Кыраакы байкоочулар программабыздын башталышында random импорттогонубузду байкаса керек. Эмне үчүн экени төмөндө айтылат. Биз колдонуучу 't' баскычын басканда тик бурчтуу объектинин XY координаттарын туш келди түзгүбүз келет. Бул үчүн, биз мурунку мисалдагыдай random.randint() колдонобуз. Ал экрандан эч качан толугу менен жоголуп кетпеши үчүн биз ага 1ден 600гө чейин пикселди беребиз.

Азыр сиздин кодуңуз төмөндөгүдөй болушу керек. Эгер андай эмес болсо же сиздин кодуңуз иштебей жатса, анда бардыгы дал келгенин жана сиздин чегинүү туура орнотулгандыгын текшериңиз:

```
# биздин модулдарды импорттоо
import pygame
from pygame.locals import *
import sys
import random

# pygame модулдарын жүктөңүз
pygame.init()

# Биздин түстөр үчүн кортеждерди түзүңүз
ак = (255,255,255)
кара = (0,0,0)
кызыл = (255,0,0)

# Оюндун негизги терезесин түзүңүз - акыркы жолу ага экран деп ат
койгонбуз
# Бул жолу башкача ат берели
оюнТерезеси = pygame.display.set_mode((800,600))

# Биздин анимацияга коштомо жазууну/аталышты коюңуз
pygame.display.set_caption('Аниматор 5000')
оюнданЧыгуу = False

x = 300
```

```
y = 300

# Оюн цикли
while not оюнданЧыгуу:
      for аракет in pygame.event.get():
            if аракет.type == pygame.QUIT:
                  оюнданЧыгуу = True
                  pygame.qui()
                  sys.exit()
            # Эгерде оюнчу 'q' баскычын басса, анда ал чыгуу аракети деп
            эсептелет
            if аракет.type == pygame.KEYDOWN:
                  if аракет.key == pygame.K_q:
                        pygame.quit()
                        sys.exit()
            # Эгерде оюнчу 'ESC' баскычын басса, анда ал иш-чараны
            токтотуу деп эсептелет
            if аракет.type == pygame.KEYDOWN:
                  if аракет.key == pygame.K_ESCAPE:
                        pygame.quit()
                        sys.exit()
            # Эгерде жебе баскычы солго басылса, объектти солго
            жылдырыңыз
            if аракет.type == pygame.KEYDOWN:
                  if аракет.key == pygame.K_LEFT:
                        x -= 10
            # Эгерде жебе баскычы оңго басылса, объектти оңго жылдырыңыз
                  if аракет.key == pygame.K_RIGHT:
                        x += 10
            # Эгерде жебе баскычы жогору жакка басылса, объектти жогору
            жылдырыңыз
            if аракет.type == pygame.KEYDOWN:
                  if аракет.key == pygame.K_UP:
                        y -=10
            # Эгерде жебе баскычы төмөн жакка басылса, объектти төмөн
            жылдырыңыз
                  if аракет.key == pygame.K_DOWN:
```

```
                y +=10
    # эгер 't' баскычы басылса, объектини туш келди
телепорттоңуз
    if аракет.type == pygame.KEYDOWN:
        if аракет.key == pygame.K_t:
            y = int(random.randint(1,600))
            x = int(random.randint(1,600))

# Оюн терезесин (оюнТерезеси) ак түс менен толтуруңуз
оюнТерезеси.fill(ак)

# Кара тик бурчтуу нерсени чыгар
pygame.draw.rect(оюнТерезеси, кара, [x,y,50,50])

# Экраныбызды жаңыртыңыз
pygame.display.update()
```

Кодуңузду иштетип, мээңиз жарылып кеткиче телепорт кылыңыз!

Кагылышууну аныктоо: дубалдардан секирүү

Оюндарыбызда көбүрөөк объектилерди жаратканда, биз объектилер бири-бирине тийишкенде, алардын жүрүм-турумун көзөмөлдөө көйгөйүнө туш болобуз. Мисалы, терезенин ортосуна жылыш үчүн эки кыймылдуу тик бурчтук бар болсо, кайсы бир учурда алардын жолдору кесилишет.

Бул кесилишке көңүл бурбай койсок болот. Кээ бир учурларда бул эң жакшы чечим болушу мүмкүн. Эң негизгиси - биз объектилерибиз бул кагылышууну байкап, кандайдыр бир реакция жасашын каалайбыз.

Кагылышууну аныктоо - бул объектилерди башка объект менен кагылышканда "билип", андан кийин ошого жараша реакция жасашы үчүн программалоо чеберчилиги. Кээ бир учурларда, биз өзүбүздүн объектибиздин ошол тарапка жылышын токтотушун каалашыбыз мүмкүн. Башка учурларда, биз алардын кубаттуу күч талаасына туш болгондой бир нече кадам артка секиришин каалашыбыз мүмкүн.

Кагылышуулар башка себептерден улам да болушу мүмкүн. Мисалы, сиз каарман өтүшү керек болгон лабиринтти жараткансыз. Эгер биз кагылышууну аныктоону орнотпосок, анда каарман түз эле биздин дубалдарыбызды сүзүп кетиши мүмкүн. Ал эшиктерге дагы тиешелүү.

Чындыгында, экранда дубалдар, эшиктер же башка нерселер болбосо да, кагылышууну аныктоону орнотуу жакшы идея болуп саналат. Эмне үчүн дебейсизби? Биз төрт бурчтукту кыймылга келтиргенде буга кыскача токтолдук. Биздин кыймылдай алган объекттер биз аныктай турган экрандын чегинен чыгып кетиши мүмкүн жана чыгып кете алышат.

Терезенин дубал чеги болбосо да, чындыгында, анын чеги болот. Бул учурларда терезенин бийиктиги жана туурасы чектелген. Мисалы, бизде 800 x 600 пикселдик терезе бар деп коёлу. Колдонмонун чектерин ушул терезенин капталдарына, өйдө жана ылдый жагына орнотуп, объектибиз чектен ашып кетсе, алардан секиртип алабыз.

Акырында кагылышуунун дагы бир түрү тууралуу кеп кылалы. Ал биз билгибиз келген - атайын кагылышуу. Душманга ок атып жаткан оюнду ойлоп көрсөңүз. Ошол октор бутага тийгенде же кагылышкан сайын биз алардын зыян келтирүү, упай топтоо же кандайдыр бир реакцияны жаратуу сыяктуу аракеттерди жасашын каалайбыз.

Жөнөкөй сөз менен айтканда, кагылышуу эки же андан көп объект атайын же капысынан бири-бирине тийишкенде болот.

Кагылышууну аныктоо: терезенин чектерин аныктоо

Pygame тиркемелерибизди түзүүдө терезебиздин чектерин эсибизден чыгарбашыбыз керек. Көп учурда объектилерибиз терезебиздин же оюн экранынын туурасы жана узуну ичинде болушун каалайбыз. Кээде андай болбой калган учурлар болот. Ошондуктан биздин объектилер оюнчунун көз алдында болуу үчүн эмне кылышыбыз керектигине токтолобуз.

Коддун кийинки бөлүмүндө биз тик бурчтуу объект терезебиздин туурасынан же узунунан чыкпагандыгын текшеребиз. Бул үчүн биз төрт бурчтук аянтчанын айланасында жүргөндө анын кайда экендигин текшерип, тик бурчтук чек араларга тийип калса, биздин программанын ага карата болгон жообун жасашыбыз керек.

Муну ишке ашыруу үчүн биз оюнТерезеси.fill(ак) түзүүдөн мурун оюн циклинин жана окуянын угармандарынын астында турган бир катар if операторлорун колдонобуз.

Чегинүүлөргө көңүл бөлүп, төмөнкү кодду кошуңуз:

```
# Экрандын оң чеги менен кагылышуубузду текшериңиз
     if x > 750:
        x -= 50
        pygame.display.set_caption('Оң кагылышуу')
     if x < 1:
        x += 50
# Экрандын сол чеги менен кагылышуубузду текшериңиз
        pygame.display.set_caption('Сол кагылышуу')
# Экрандын төмөн чеги менен кагылышуубузду текшерип алыңыз
     if y > 550:
        y -= 50
```

```
        pygame.display.set_caption('Төмөнкү кагылышуу')
# Экрандын жогору чегине туш келип калганыбызды текшериңиз
    if y < 1:
        y += 50
        pygame.display.set_caption('Жогорку кагылышуу')
```

Мындай кичинекей код үчүн бул, албетте, көп. Ар бир кадамын карап көрөлү.

Биздин биринчи if операторубузда *эгерде* биздин тик бурчтук объектибиз 750 пиксель же андан чоңураак мейкиндикте жайгашкан болсо, анда тик бурчтук объектисибизди тескери багытта 50 пикселге артка жылдырып, секирүү эффектин жаратабыз. Бул x өзгөрмөсүнөн 50 пиксель (- = 50) алып салуу менен жасалат. Эсиңизде болсо, бул биздин тик бурчтук объектинин X координатын билдирет.

Сиз биздин объект 800дөн жогору пикселдик координатада экендигин текшерүүчү программа жок экендигин байкадыңызбы? Эмне үчүн ушундай? Мунун жообу жөнөкөй: биз кагылышууну аныктап жаткан нерсенин көлөмүн ар дайым эсибизден чыгарбашыбыз керек. Биз бул учурда анын өлчөмү 50нү эң жогорку координаталык мааниден кемитишибиз керек. Демек, биздин тик бурчтук 50 пиксель болсо, экраныбыз 800 пиксель болсо, тик бурчтук чек арага тийип, андан ашып кетпеши үчүн X координатынын 750 же андан жогору экендигин текшеришибиз керек.

Биздин коддун кийинки бөлүгү кайрадан X координатасында иштейт. Бул жолу экрандын сол тарабы менен кагылышуу болгонун текшерип жатабыз. Бул жерде биз 1ден төмөн маанилерди текшерүүнү каалайбыз (эсиңизде болсун: экрандын сол жагындагы чек X координатасында 0дө жайгашкан). Дагы бир жолу эгер тик бурчтук ушул "дубалга" туш келсе, анда ал тескери багытта 50 пикселге секирет.

Бул логиканы Y координаттары үчүн дагы улантып, терезебиздин жогорку жана төмөнкү кагылышууларын текшеребиз. Эгерде дагы бир жолу кагылышуу байкалса, анда биздин тик бурчтук 50 пикселге карама-каршы багытта секирет. Бул жолу абалга жараша жогору же төмөн болот.

Акыры программага дагы бир аз көбүрөөк шык кошуу үчүн, ар бир if билдирүүсү кагылышуу болгонун билип калса, эскертип, жогору, ылдый, солго же оңго деп терезенин жазуусун өзгөртөт.

Мына ошентип, биздин биринчи кагылышууну аныктоонун өзгөчөлүгүн билдик!

Программаны сактап, аны сынап көрүңүз, программанын талаптагыдай иштешине ынануу үчүн ар бир тарапка секиртип алыңыз. Эгер андай болбосо, анда аны төмөнкү аяктаган программанын кодуна дал келгенин текшериңиз:

```
# биздин модулдарды импорттоо
import pygame
from pygame.locals import *
```

```
import sys
import random

# pygame модулдарын жүктөңүз
pygame.init()

# Биздин түстөр үчүн кортеждерди түзүңүз
ак = (255,255,255)
кара = (0,0,0)
кызыл = (255,0,0)

# Оюндун негизги терезесин түзүңүз - акыркы жолу ага экран деп ат койдук
# Бул жолу башкача ат берели
оюнТерезеси = pygame.display.set_mode((800,600))

# Биздин анимацияга коштомо жазууну/аталышты коюңуз
pygame.display.set_caption('Box Animator 5000')

оюнданЧыгуу = False

x = 300
y = 300

# Оюн цикли
while not оюнданЧыгуу:
    for аракет in pygame.event.get():
        if аракет.type == pygame.QUIT:
            оюнданЧыгуу = True
            pygame.quit()
            sys.exit()
        # Эгерде оюнчу 'q' баскычын басса, анда ал чыгуу аракети деп
        эсептелет
        if аракет.type == pygame.KEYDOWN:
            if аракет.key == pygame.K_q:
                pygame.quit()
                sys.exit()
        # Эгер оюнчу 'ESC' баскычын басса, анда ал окуяны токтотуу
        деп эсептелет
        if аракет.type == pygame.KEYDOWN:
            if аракет.key == pygame.K_ESCAPE:
```

```
                pygame.quit()
                sys.exit()
          # Эгерде сол жебе баскычы  басылса, объектти солго
жылдырыңыз
          if аракет.type == pygame.KEYDOWN:
               if аракет.key == pygame.K_LEFT:
                    x -= 10
          # Эгерде оң жебе баскычы басылса, объектти оңго жылдырыңыз
               if аракет.key == pygame.K_RIGHT:
                    x += 10
          # Эгерде жогору жебе баскычы басылса, объектти өйдө
          #жылдырыңыз
          if аракет.type == pygame.KEYDOWN:
               if аракет.key == pygame.K_UP:
                    y -=10
          # Эгерде төмөн жебе баскычы басылса, объектти төмөн
          жылдырыңыз
               if аракет.key == pygame.K_DOWN:
                    y +=10
          # Эгерде 't' баскычы басылса, объектини кокустан телепорттоңуз
          if аракет.type == pygame.KEYDOWN:
               if аракет.key == pygame.K_t:
                    y = int(random.randint(1,600))
                    x = int(random.randint(1,600))
          #  Экрандын оң жак чети менен кагылышуубузду текшериңиз
          if x > 750:
               x -= 50
               pygame.display.set_caption('Right Collision')
          if x < 1:
               x += 50
          # Экрандын сол жак чети менен кагылышуубузду текшериңиз
              pygame.display.set_caption('Left Collision')
          # Экрандын төмөн жагында кагылышуубузду текшериңиз
          if y > 550:
               y -= 50
               pygame.display.set_caption('Bottom Collision')
```

```
            # Экрандын жогорку жагында кагылышуубузду текшериңиз
            if y < 1:
                  y += 50
                  pygame.display.set_caption('Top Collision')

    # Оюн терезесин ак түс менен боёңуз
    оюнТерезеси.fill(ак)

    # Кара тик бурчтуктун объектисин көчүрүп алыңыз
    pygame.draw.rect(оюнТерезеси, кара, [x,y,50,50])

    # Экраныңызды жаңыртыңыз
    pygame.display.update()
```

Эки объекттин кагылышуусу

Эми чектерди аныктоону жөндөп, кагылышууну аныктоонун дагы бир маанилүү түрүнө - эки объект бири-бири менен кагылышуусун аныктоого өтсөк болот. Жогоруда айтылгандай, бир нече объектинин кагылышуусун текшерүүгө көптөгөн себептер бар. Эки каармандын кагылышуусун же бутага тийген куралды байкоодон сырткары, кагылышууну аныктоодо, же объекттин мейкиндикте жайгашкандыгын аныктоодо пайдалуу.

Мисалы, сизде кандайдыр бир каарман объектилердин үстүнөн секириши керек болгон оюн болсо, платформалык оюндагыдай эле сиздин каарманыңыз чөптүн үстүндө турабы же кутунун үстүндө экенин кайдан билесиз? Ушундай максатта кагылышууну аныктоону же сокку урууну аныктоону колдонсоңуз болот.

Кийинки мисалда биз objectCollisionExample.py деп аталган жаңы Python файлын жаратканы турабыз. Биз кээ бир коддорду pygameAnimations.py программасынан алабыз. Коддун ар бир бөлүмүн карап чыгуунун ордуна мен программаны толугу менен алып, андан кийин эски кодубузга киргизилген жаңы толуктоолорду жана өзгөртүүлөрдү ка дам сайын карап чыгам.

Бир аз убакыт бөлүп, жаңы файлды түзүңүз жана ага төмөнкү кодду көчүрүңүз. Программанын максатын жана анын кандайча иштээрин мен түшүндүрүп бергенден мурун комментарийлерди окуп чыгууну унутпаңыз. Адаттагыдай эле, кодуңузду туура чегиндирүүнү унутпаңыз. Антпесе каталар пайда болот:

```
# биздин модулдарды импорттоо
import pygame
from pygame.locals import *
import sys

# pygame модулдарын жүктөңүз
pygame.init()
```

```
# Биздин түстөр үчүн кортеждерди түзүңүз
ак = (255,255,255)
кара = (0,0,0)
кызыл = (255,0,0)

# Оюндун негизги терезесин түзүңүз - акыркы жолу ага экран деп ат койдук
# Бул жолу башкача ат берели
оюнТерезеси = pygame.display.set_mode((800,600))

# Биздин анимацияга коштомо жазууну/аталышты коюңуз
pygame.display.set_caption('Colliding Objects')

оюнданЧыгуу = False

# Спрайттын тик бурчтук объектилерин сактай турган эки өзгөрмө түзүңүз
тик1 = pygame.sprite.Sprite()
тик1.rect = pygame.Rect(300,300,50,50)
тик2 = pygame.sprite.Sprite()
тик2.rect = pygame.Rect(100,100, 100,150)

# Оюн цикли
while not оюнданЧыгуу:
    for аракет in pygame.event.get():
        if аракет.type == pygame.QUIT:
            оюнданЧыгуу = True
            pygame.quit()
            sys.exit()
        # Эгерде оюнчу 'q' баскычын басса, анда ал чыгуу аракети деп
        эсептелет
        if аракет.type == pygame.KEYDOWN:
            if аракет.key == pygame.K_q:
                pygame.quit()
                sys.exit()
        # Эгерде оюнчу 'ESC' баскычын басса, анда ал иш-чараны
        токтотуу деп эсептелет
        if аракет.type == pygame.KEYDOWN:
            if аракет.key == pygame.K_ESCAPE:
                pygame.quit()
                sys.exit()
```

```
# Эгерде жебе баскычы солго басылса, объектти солго жылдырыңыз
if аракет.type == pygame.KEYDOWN:
    if аракет.key == pygame.K_LEFT:
        тик1.rect.x = тик1.rect.x - 10
# Эгерде жебе баскычы оңго басылса, объектти оңго жылдырыңыз
    if аракет.key == pygame.K_RIGHT:
        тик1.rect.x = тик1.rect.x +10
# Эгерде жебе баскычы жогору жакка басылса, объектти жогору
жылдырыңыз
if аракет.type == pygame.KEYDOWN:
    if аракет.key == pygame.K_UP:
        тик1.rect.y = тик1.rect.y -10
# Эгерде жебе баскычы төмөн жакка басылса, объектти төмөн
жылдырыңыз
    if аракет.key == pygame.K_DOWN:
        тик1.rect.y = тик1.rect.y +10
# Биздин эки rect объектилердин кагылышуусун collide_rect
колдонуп текшериңиз
# Эгер кагылышуу аныкталса, анда анын у жана х
# координаттарын өзгөртүү аркылуу тик1 ордун которобуз
if pygame.sprite.collide_rect(тик1, тик2):
    тик1.rect.y = 400
    тик1.rect.x = 400

# Экрандын оң жак чеги менен кагылышуубузду текшериңиз
# Эгер ошондой болсо, анда биз тик1 объектисин X координатасына
740 пикселге жылдырабыз
    if тик1.rect.x > 750:
        тик1.rect.x = 740
        pygame.display.set_caption('Right Collision')
    if тик1.rect.x < 1:
        тик1.rect.x = 51

    # Экрандын сол жак чеги менен кагылышуубузду текшериңиз
        pygame.display.set_caption('Left Collision')
# Экрандын төмөн чегине кагылышуубузду текшериңиз
    if тик1.rect.y > 550:
        тик1.rect.y = 540
```

```
                pygame.display.set_caption('Bottom Collision')
        # Экрандын жогору чегине кагылышуубузду текшериңиз
            if тик1.rect.y < 1:
                тик1.rect.y = 50
                pygame.display.set_caption('Top Collision')
# Оюн терезесин ак түс менен боёңуз
оюнТерезеси.fill(ак)

# Биздин тик бурчтуу нерселерди көчүрүү
pygame.draw.rect(оюнТерезеси, кара, тик1)
pygame.draw.rect(оюнТерезеси, кызыл, тик2)

# Экраныңызды жаңыртыңыз
pygame.display.update()
```

Бул код мурунку программа сыяктуу эле иштейт. Биз тик бурчтуу объектти түзөбүз. Бул жолу аны спрайтка киргиздик. Аны жебе баскычтарын колдонуп, терезенин айланасында айлантабыз. Эгерде тик бурчтук терезенин кандайдыр бир четтерине же чектерине тийсе, анда төрт бурчтук "дубалдан" бир нече пикселге секирет.

Буга кошумча биз экинчи статикалык тик2 тик бурчтуктугун түздүк. Бул аянтчанын айланасында кыймылдабайт. Ошондой эле, тик1 тик2 менен урунарын текшерүү үчүн кодду өзгөрттүк. Эгер ошондой болсо, анда биз ал "дубалга" урунгандай кылып, тик1’ объектисинин X жана Y координаттарынын маанисин өзгөртөбүз.

Бул коддун мурунку кагылышууну аныктоо мисалынан негизги айырмачылыгы - биздин тик бурчтуу объекти кантип жараткандыгыбызга байланыштуу. Жөн гана .rect колдонуп, биздин тик бурчтукту көчүрүүнүн ордуна бул жолу биз тик бурчтуктун объектилерин кармоо үчүн өзгөрмө түзгүбүз келет.

Pygame.sprite.Sprite() менен кошо орнотулган кээ бир функцияларга колдоно алуу үчүн биз бул тик бурчтуктарды спрайт кылып жасайбыз. Pygame.sprite модулун коштогон көптөгөн орнотулган функциялар бар. Тилекке каршы, биз алардын бардыгын камтыганга бул китепте орун жок. Бирок, биз кагылышууну аныктоого жардам бере турган өтө маанилүү бир нерсеге токтолобуз.

Биздин тик бурчтук объектилерибизди өзгөрмөлөрдө сактоо жана аларды спрайт кылуу менен, биз ошол атрибуттарга түздөн-түз жетип, алардын XY координаттарын өзгөртүү мүмкүнчүлүгүнө ээ болобуз.

Мисалы, программанын кодунан сиз тик1 объектисине окшош нерсени көрө аласыз. rect.x = 100. Бул код негизинен тик1 деп аталган өзгөрмөнү алып, анда сакталган түз объектке кирип, анын x маанисин 100гө өзгөрткүңүз келет деп айтат.

Биздин коддун ушул бөлүгү төмөнкүчө:

```
тик1 = pygame.sprite.Sprite()
тик1.rect = pygame.Rect(300,300,50,50)

тик2 = pygame.sprite.Sprite()
тик2.rect = pygame.Rect(100,100, 100,150)
```

тик1 жана тик2 биздин эки тик бурчтуу объектилерибизди түзүү үчүн колдонулат. Биз pygame.Rectти колдонгондо, "R", аны тик1.rect болгондой колдонгондон айырмаланып, баш тамга менен белгиленээрин унутпаңыз. Аны баш тамга менен белгилебей койсоңуз, ката кетиресиз.

Кийинки өзгөрүү - объектилердин, атап айтканда, тик1 терезенин айланасында кандайча кыймылдай тургандыгын аныктайбыз.

Биздин *rect* объект азыр спрайт болгондуктан, анын параметрлерине, мисалы, анын XY координаттарына башкача мүмкүнчүлүк алышыбыз керек. Эми объект кыймылдашы үчүн биз бул ыкманы колдонобуз:

```
# Эгерде сол жебе баскычы басылса, объектти солго жылдырыңыз
    if аракет.type == pygame.KEYDOWN:
        if аракет.key == pygame.K_LEFT:
            тик1.rect.x = тик1.rect.x – 10
```

.move_ip ыкмасын колдонуунун ордуна эми биз, тескерисинче, ушул саптагыдай эле, тик1.rect.x маанисин жаңы мааниге которуп алганыбызга көңүл буруңуз:

```
            тик1.rect.x = тик1.rect.x – 10
```

бул rect1 учурдагы X координатасын алып, андан 10ду алып салууну айтат.

Биз муну төрт багыттагы жебе үчүн ар бир баскычты басуу окуясы үчүн жасайбыз жана колдонуучу тик бурчтукту кайсы багытта жылдырып жатканына жараша тик1.rect.x жана тик1.rect.y маанисин өзгөртөбүз.

X мааниси сол жана оң кыймылдарды, ал эми Y мааниси өйдө жана ылдый кыймылдарды билдирерин унутпаңыз.

Кийинки кадам - бул тик1 качандыр бир кезде тик2 объектисине тийип-тийбесин билүү үчүн кагылышууну аныктоону колдонуу. Бул жерде биздин тик бурчтук объектилерди спрайт деп аныктоо ыңгайлуу. Спрайт объектинин камтылган функцияларынын бири collide_rect болуп саналат. Ал эки аргументти алат: кагылышууну аныктай турган эки объектин аталышы.

Биз муну if операторунун ичинде колдонуп, объектилердин бири-бирине тийип же бири-бири менен кагылышып калышын текшере алабыз. Эгер ошондой болсо, аны тик2 объектисинен алыстатып, тик1 объектисинин X жана Y координаттарын 400 x 400 деп өзгөртөбүз. Мунун бардыгы ушул жөнөкөй код менен ишке ашат:

```
        if pygame.sprite.collide_rect(тик1, тик2):
            тик1.rect.y = 400
            тик1.rect.x = 400
```

Акырында биздин кодубузга акыркы жолу киргизилген өзгөрүүлөр чек аралардын оюн терезеси менен кагылышуусун жөнгө салды. Бул негизинен мурдагыга окшош, бирок координаттарды же биздин тик бурчтукту өзгөртүү үчүн x жана y маанилерин алмаштыруунун ордуна, тик бурчтуктун XY параметрлерине түз киребиз.

Эгерде четтери менен кагылышуусу байкалса, биз тик бурчтукту бир нече артка жылдырабыз:

Мисалы:

```
if тик1.rect.x > 750:
      тик1.rect.x = 740
      pygame.display.set_caption('Оң кагылышуу')
```

эгер тик1 объектисинин X координаты 750дөн чоң болсо, тик1 X координатасын 740ка артка жылдырып, андан кийин терезенин жазуусун “*Right Collision*” деп өзгөртүңүз.

Программаны текшериңиз, кагылышууну аныктоо эки объектинин төрт тарабында тең иштей тургандыгын текшерип, тик1 объектисин жогорку, төмөн, сол жана оң жактардан тик2 объектисине жылдырууга аракет кылыңыз.

Андан кийин терезенин чек аралары менен кагылышуусун аныктоону текшерип, дагы бир жолу жогору, төмөн, сол жана оң чектерин текшерип чыгыңыз.

Эгер код иштебей жатса, анын дал келгенине ынанып, кайра окуп чыгып, салыштырып көрүңүз.

Жана, адаттагыдай эле, чегинүү жөнүндө унутпаңыз.

Бул эпизоддо

Акыркы эки бөлүмдө сиз укмуштуудай сонун нерселерди жасадыңыз жана ал сизди токтобос баатыр кылып жаратып жаткандай сезилди! Бул акыркы эки бөлүмдө мүмкүн болгон ар бир оюн темасын камтыган жокпуз. Анын бардыгын туура жасаш үчүн бүтүндөй бир китепти (эгер андан көп болбосо) жазуу талап кылынат. Негизги видео оюнун иштеп чыгууга жана андан да маанилүүсү, өзүңүздүн татаал оюндарыңызды түзүү үчүн изилдөө ишиңизди баштоого жетиштүү маалыматты бул бөлүмдө камтыдык.

Сизге үй тапшырма! Алдыга чыгып, кызыктуу оюн, же жок дегенде анын скелетин түзүңүз. Сиз дүйнөгө белгилүү оюн иштеп чыгуучу болуп калганыңызда мен аны Playstation 402 же башка жерде ойноону чыдамсыздык менен күтөм!

13-БӨЛҮМ

КАТАЛАРДЫ ЧАГЫЛДЫРУУ

Каталар үстүндө иштөө

Биз укмуштуу окуяларды башыбыздан өткөрүп, бул китептин соңуна да жакындап келатабыз. Жакында сиз өзүңүздүн коргоочу кийимиңизди кийип, менин жардамымсыз эле жаман адамдар менен күрөшүп (албетте, сиздин кыялыңызда гана!), анан да өзүңүз программа жазып баштайсыз.

Көңүлүңүздү чөгөрбөңүз.

Азырынча биз дагы деле супер баатырбыз! Көптөгөн программалык шылуундарды жеңип, ар кандай тоскоолдуктарды жеңип чыгууну үйрөнгөнүбүз менен, дагы бир талкуулай турган темабыз бар: өзүбүз ийгиликке жетпей калганда кантебиз?

Ийгиликке жетпей калганда дейсизби? Кандай баатыр же программист ийгиликке жетпей калат болду экен?

Бактыга жараша, мындай нерсе бардыгыбызда бар. Тагыраак айтканда, аны биз бардыгыбыз башыбыздан өткөрөбүз.

Эмне үчүн бактыга жараша? Ийгиликсиздикти да бакыт менен салыштырууга болобу деп ойлошуңуз мүмкүн. Ийгиликсиздик түбүндө жакшы деле нерсе эмес. Бирок биздин ишибизде ийгиликсиздик - бул күчтүү болуунун, биздин көндүмдөрүбүздү жана программалоо жөндөмдөрүбүздү өркүндөтүүнүн эң мыкты жолдорунун бири.

Ойлонуп көрсөңүзчү: эмне үчүн оор салмакты көтөргөндө булчуң чыңалат? Булчуңду айрып алганыгыңыздан кийин аны айыктырып дарылоого туура келет. Алыңыз келбеген оордукту көтөрүп, бир маалда сиз көтөрө албай турган чекитке жетесиз. Башкача айтканда, ийгиликсиз болосуз. Мындай мүчүлүштүктөрдү культурист балбан издейт. Анткени булчуңдар иштебей калган учурда алар зыянды калыбына келтирип, максатына жетүү үчүн ого бетер күчөй башташарын билишет.

Сиздин программалоо чеберчилигиңиз дагы ушуга окшош. Менин оюмча, кодду чындыгында эле түшүнүүнүн бирден-бир жолу - аны бузуп, кандай ката кетиргендигиңизди билүү. Ар бир адам тилди биле алат. Бирок программалык тилди түшүнүү көп жылдар бою сизди коштогон ийгиликсиздиктен улам калыптанып жетилет.

Эскертүү: Ушул эле логиканын сиздин алгебра тесттериңизге тиешеси жок... Бул предметтен деги жаңыла көрбөңүз.

Азырынча кандайдыр бир көйгөйгө туш болгондо, жөн гана .py файлды саптан-сапка окуп чыгып, эмнеси туура эмес болгонун билүүгө аракет кылдык. Чынын айтсам, эч кандай мүчүлүштүктөр болгон жок. Менин мисалдарымды кандай жазылган болсо, ошол бойдон киргиздиңиз.

Бирок, бир-эки сөздү туура эмес жазып, программаңыздын иштебей калышына себеп болдуңуз. Андан кийин эмне кылдыңыз? Эгер ушул убакка чейин жеткен болсоңуз, сиз кодду издеп, көйгөйдүн пайда болуш себебин, туура эмес жазылган сөздү же чегинүүнү издедиңиз. Аны таап, андан кийин аны оңдодуңуз.

Эгер ушундай болгон болсо, куттуктайм. Эң жөнөкөй тил менен айтканда, расмий түрдө программаны сынап, мүчүлүштүктөрдү debugs аркылуу оңдоп алдыңыз.

Мүчүлүштүктөрдү оңдоо - debugging - бул биз үчүн жаңы сөз. Бул биздин коддон каталарды аныктап, алып салуу дегенди билдирет.

Келгиле, дагы бир жолу ошол аныктаманы карап көрөлү. Ал каталарды аныктап, аларды жок кылуу дегенди билдирет.

Мүчүлүштүктөрдү оңдоо процессинин эң чоң бөлүгү - мен үчүн катаны табуу. Ал жерден биз катаны түшүнүшүбүз керек. Ошондо гана катаны пайда кылган көйгөйдү оңдойбуз же алып салабыз.

Чакан программалар үчүн маселени издеп, саптан-сапка өтсөк болот. Чоңураак программалар үчүн мүчүлүштүктөрдү оңдоочу -debugger деп аталган программаны колдонууну каалайбыз.

Каталарды табуу

Python, адатта, бизге бир нерсени бузуп алгандыгыбызды жакшы айтат. Бул бөлүмдө, келгиле, Oops.py аттуу жаңы файл түзөлү.

Бул файл каталарга толуп, программаларыңызды иштетип жатканда IDLE сизди катуу урушат деп күтүңүз. Биз өзүбүздүн кодду киргизгенибизде андагы каталар сизге ачык көрүнүшү мүмкүн же байкалбай калышы мүмкүн. Кандай болгон күндө да, көрсөтмөлөрдү аткарыңыз жана эгер сиз ката кетирген болсоңуз, аны көрмөксөн болуңуз. Ал программаларды оңдоону жакшыраак түшүнүүгө жардам берет.

Oops.py файлына төмөнкү кодду киргизиңиз:

```
print салам
```

Зээндүү программисттер, сиздер көйгөйдү көрүшүңүздөр мүмкүн. Эгер андай болсо, рахмат! Бирок көрмөксөн болуңуз. Программаны иштетип, эмне болуп жатканын көрүңүз. Бүттүбү? Сиз ката жөнүндө кабар алдыңызбы? Чын эле жасадыңызбы! Ал төмөнкүдөй окулушу керек:

```
_____________________________________________________________________________
Missing parentheses in call to 'print'. Did you mean print(салам)?
_____________________________________________________________________________
```

Бул калкыма билдирүү катары көрүнүшү керек эле. Көрүнүп тургандай, IDLE абдан акылдуу. Ал сиздин кодуңузда көйгөй болуп жаткандыгын эле байкабай, аны оңдоо боюнча сунуш берди.

Бул жерде белгилей кетүүчү бир нече нерсе бар. Биринчиден, IDLE бул жерде сунуш киргизгени менен, чындыгында, туура эмес. Басып чыгарууну каалаган текстти тырмакчага киргизбегендиктен, IDLE биз, чындыгында, өзгөрмөнү басып чыгарууга аракет кылып жатабыз деп болжолдойт. Ошондой эле бизге *салам* деген өзгөрмөнү кантип басып чыгарууну көрсөтөт.

Бул учурда бул биз каалаган нерсе болгон жок. Бирок Python катаны көрсөтүп, мүмкүн болгон чечимди сунуштаганы жакшы.

Чындыгында, биз *"салам"* деген сөздү басып чыгарууну көздөгөнбүз, бул, албетте, print(“салам”) деп басып чыгарууну терүү сыяктуу эле. Бирок, биз программанын иштебей калышын каалайбыз.

Демек, бул биринчи катадан сабак. Кээде IDLE бизге калкыма билдирүүсүндө ката жөнүндө кабар берет жана ката кайда болгонун жана мүмкүн болгон чечимди сунуштайт. Бул туура эмес болушу мүмкүн.

Эми файлдагы кодду дал ушуга ылайыкташтырып өзгөртөбүз:

```
print("салам)
```

Дагы, сиз буга чейин көйгөйдү байкагандырсыз - биз print() функциясын тырмакча менен жабууну унутуп койгонбуз. Бирок эмне болуп жаткандыгын көрүү үчүн программаны иштетип көрүңүз.

Бул жерде дагы бир жолу калкыма терезе пайда болот. Бул жолу биз башка типтеги каталарды - саптын акыры катасын - an end-of-line (EOL) алабыз. Ал мындай деп айтышы керек:

```
_____________________________________________________________________________
    EOL while scanning string literal.
_____________________________________________________________________________
```

Бул жерде Python, негизинен, биз сапты жапкан жокпуз деп айтып жатат. Биз жаппагандыгыбызды билебиз. Эгерде биз калкып чыкма терезедеги "ОК" баскычын баса турган болсок, анда IDLE бизди файлга кайтарып, саптын калган бөлүгүн кызыл түс менен белгилеши керек. Анткени ал бул жерде ката кетти деп ойлогонун билдирип атат.

Маселе экинчи тырмакчаны унутуп калгандыгыбызда. Биз муну оңдойлу деп жатабыз. Бирок бул жолу анын ордуна экинчи кашааны таштап, эмне болорун карап көрөлү. Кодуңузду төмөнкүгө дал келтирүү үчүн оңдоп, андан кийин дагы бир жолу иштетиңиз:

```
print("салам "
```

Бул жолу биз дагы бир ката чыкканын көрөбүз. Бул файлдын аягындагы (EOF) ката: анализ жүргүзүүдө күтүлбөгөн EOF.

Бул жолу "ОК" баскычын басканда Python биздин коддун астындагы сызыкты баса белгилейт. Эмне үчүн?

Биздин кодду иштетип жатканда Python коддун сабынын аягына чейин издеп жаткан. Ал жабык кашаа болушу керек эле. Тескерисинче, ал жакындан тапкан жок, андыктан кошумча код издеп кийинки сапка өттү. Ал жерден жабык кашаа табылбай калганда же ал бизди алдап жатат деп чечтиби? Же ката кайдан кеткенин көрсөтүү үчүн сапты бөлүп көрсөттүбү?

Бул абдан маанилүү. Себеби биз, адатта, Python ката кайда экенин көрсөтүп турат деп ойлойбуз. Чындыгында, Python кайда ката кетиргенин көрсөтөт (бул бир сөз болсо дагы).

Аны трамплиндин четинен чуркагандай элестетиңиз. Батуттан биринчи кадам - сиз ката кетирген жериңиз. Бирок сиз, балким, курсагыңыз менен кулап түшүп, жалпы класс алдында шымыңызды жоготуп алган ыңгайсыз абалда калмайынча аны түшүнө албай жатасыз.

Python программалоо тилинде да ушундай эле жол.

Кийинки ката жөнүндө.

Эми кодду өзгөртөбүз, ал төмөнкүгө дал келиши керек:

```
prant("салам")
```

Файлды сактап, аны иштетиңиз. Бул жолу калкыма терезелер жок. Биз жакшы иш кылдык окшойбуз!

Ооба, анча эмес.

Бул жолу Python Shell иштөөгө аракет кылганда биз төмөнкүдөй жыйынтыкка ээ болдук:

```
_____________________________________________________________________
Traceback (most recent call last):
     File "C:/Users/ ... /Python/Python36-32/Oops.py",
     line 1, in <module>
          prant("салам")
NameError: name 'prant' is not defined
_____________________________________________________________________
```

Катанын бул түрү NameError деген ат менен белгилүү. Келгиле, ушул чыгарылыштын ар бир бөлүгүн кылдаттык менен карап чыгалы.

Биринчи сапта:

```
Traceback (most recent call last):
```

Бул Python кодуңуздагы каталарды издеп жаткандыгын айтып, акыркы чалуу биринчи пайда болгонун билдирди. Бизде бир гана ката бар. Бирок буга кабатыр болбоңуз.

Андан кийин Python айрым маанилүү маалыматтарды айтып берет. Ал файлдын жайгашкан жерин (сиздики меники менен айырмаланат) жана ката кайсы сапта деп ойлойт. Бул учурда ал 1-сапты же кодуңуздун биринчи сабын билдирет.

Андан кийин ал бизге белгилүү бир ката кетирген кодду көрсөтөт: prant("салам"). Акыр-аягы, ал ката түрүндөгү билдирүү менен аяктайт - NameError - жана толук маалымат берет: "prant" аталышы аныкталбайт.

Биз ушундай билдирүүнү көргөндө Python билдирет:

```
prant()
```

жана аны функциялар камтылган тизмесинен таба алган жок. Себеп? Себеби ал жок. Биз print() деп туура эмес жазганбыз жана аны биз билгенибиз менен, Python муну билүү мүмкүнчүлүгүнө ээ эмес.

Python prant() деп аталган орнотулган функцияны таппагандыктан, биз өзүбүз түзгөн функцияны prant() деп атоого аракет кылып жатабыз деп болжолдойт. Биз мындай аталыш менен функция жасабагандыктан, Python “биз туура эмес ат тердик же ушул аталыштын функциясын аныктай албай койдук” деп жыйынтык чыгарат.

Бул биз үчүн өтө жөнөкөй ката. Издөө же издөө, анан оңдоо. Биз бар болгону 1-сапты карап, бузулган кодду таап, prant("салам") кодун print("салам") деп өзгөртөбүз. Андан кийин эгер биз аны сактап иштетсек, баары туура болот.

Албетте, биз азырынча минтип убара болуубуздун кажети жок. Анткени биз дагы деле болсо кодубузда атайылап ката кетирип жатабыз.

Келгиле, алдыга жылардан мурун дагы бир ката кетирип көрөлү. Кодуңузду дагы бир жолу өзгөртүп, төмөнкүлөргө дал келтириңиз:

```
a = 1
    whilst a < 4:
        print(a)
        a = a + 1
```

Бул код "a" өзгөрмөсүнө 1 маани бериш керек. Андан кийин биз 'a' мааниси 4кө жетпеген учурда кайталоого аракет кылган while циклин түздүк.

Бул циклдин ар бир кайталанышы үчүн "a" мааниси басылып чыгарылат, ошондой эле "a" маанисине 1 кошуп отурабыз. Теория боюнча, бул коддун чыгышы:

```
1
2
3
```

Бирок, биз кодубузда дагы бир ката кетирдик. Бул жолу аны байкай аласызбы? Эгер сиз кодду иштетсеңиз, анда калкыма терезе түрүндөгү кеңеш пайда болот:

```
invalid syntax.
```

Бул эмнени билдирет? Бул биздин кодду туура эмес жазгандыгыбызды билдирет. Дагы бир жолу, биздин ката кайда экендигин көрсөткөн кызыл түскө бөлөнөбүз.

Көйгөй? while ордуна *whilst* деп жазып койдук.

Келгиле, орфографияны оңдойлу, бирок анын ордуна дагы бир ката кетирели. Кодду дал келгендей кылып өзгөртүңүз:

```
a = 1
while a < 4
    print(a)
    a = a + 1
```

Бул жолу катаны байкай аласызбы? Файлды сактап, кодду иштетебиз. Кайра, while жазуусун оңдосок да, синтаксистик ката кетет. Бардык башка сөздөр да туура жазылган, анда эмне себеп болду?

Синтаксистик каталар орфографиялык каталарды гана эмес, кеңири тараган каталарды дагы камтыйт. Бул учурда while операторунун аягында кош чекитти кошууну унутуп калдык. Сапта:

```
while a < 4:
```

Эгерде сиз кош чекитти аягына кошуп, файлды сактасаңыз, анда ал туура иштеши керек.

Каталардын түрлөрү

Чындыгында, Python тилинде каталардын үч гана негизги түрү бар. Алар: синтаксистик каталар, логикалык каталар жана өзгөчө учурлар. Бул каталардын ар бир түрүн жана аларды кантип иштетүү керектигин ушул бөлүмдө камтыйбыз. Ошентип, супер баатырдын көз айнегин тагынып, ааламдагы көйгөйлөрдү чечүүгө даярданалы!

Макул, алгач биздин коддогу көйгөйлөрдөн баштасак кандай дейсиз.

Синтаксистик каталар

Мурунку каталарга сереп салууда синтаксистик каталарды бир аз талкууладык. Кайталап өтөлү, Python коддун сабын түшүнө албай же окуй албай калганда, синтаксистик ката кеткен болот.

Синтаксистик каталар, адатта, туура эмес жазуудан улам келип чыгат. Балким, сиз функцияны туура эмес жазгансызбы же билдирүүнүн аягында кош чекит кошууну унуткандырсыз. Аларды грамматикалык же орфографиялык каталар деп эсептеңиз.

Сиз алган бардык каталардын ичинен синтаксистик каталар көп кездешиши мүмкүн. Мунун жакшы да, жаман да жагы бар. Жакшы жагы - көбүнчө сиздин каталарыңыз программалоо логикасына эмес, орфографиялык же пунктуациялык көйгөйгө байланыштуу болот. Жаман жагы - аларды көзөмөлдөө жагымсыз болушу мүмкүн. Айрыкча, күнү-түнү код жазуудан көзүңүз тунарып калгандан кийин жагымсыз болот.

Синтаксистик каталардын басымдуу бөлүгү фаталдуу жана кодуңуздун аткарылбай калышына алып келет. Бул дагы бир жолу жашыруун пайда. Кодуңуздун таптакыр иштебей калганы кыжырды келтириши мүмкүн. Ал ошондой эле жашыруун маселеси бар программалык камсыздоону жөнөтпөөнү камсыз кылат.

Эгер синтаксистик ката кетсе, анда IDLE баракчасында кызыл сызык кайсы жерде чагылдырылгандыгын белгилеңиз же ката пайда болгон саптын номерин белгилеп, орфография, чегинүү, чекит, тырмакча жана кашаа, же башка аргументтерден ар кандай көйгөйлөрдү издеңиз.

Логикалык каталар

Бардык каталардын ичинен эң көйгөйлүүсү - бул коркунучтуу логикалык каталар. Аты айтып тургандай, булар программалоо логикасында кемчиликтер болгондо пайда болот. Бул каталардын түрлөрү сиздин программанын күлкүлүү кыймыл-аракетине же толук жарылып кетишине алып келиши мүмкүн.

Логикалык каталар - ушунчалык кыжырданткан нерсенин бир бөлүгү, алар дайыма эле ачык ката кетирбегени менен айырмаланат. Кээде Python катаны байкабай калса, сиз да аны байкабай калышыңыз мүмкүн. Ошондуктан, кодуңузду тез-тез текшерип, мүмкүн болушунча документтештирип турганыңыз өтө маанилүү.

Ушул типтеги каталарды табуунун бир нече жолу бар. Бул тууралуу кийинчерээк талкуулайбыз. Логикалык каталар менен күрөшүүнүн эң мыкты жолу - бул биринчи кезекте алардын алдын алуу. Биз программаларыбызды алдын ала пландаштырып, аларды тез-тез текшерип турабыз. Блок-схемаларды колдонуу кодуңуздун ар бир бөлүмү кандайча аткарылышы керектигин түшүнүүгө жардам берет жана логикалык каталардан алыс болуунун пайдалуу куралы болуп саналат.

Албетте, баары бир логикалык каталар болот. Бул программисттин ишинин бир бөлүгү гана.

Мында логикалык программанын бир мисалы берилген. Бул программанын сиз ойлогондой болбогон натыйжаны эмне үчүн кайтарып берерин билип алыңыз. Бул программанын максаты эки сандын орточо арифметикалык маанисин табуу:

```
a = 10
b = 5
average = a + b / 2
print(average)
```

Эгерде сиз математика жаатында мыкты болбосоңуз, кабатыр болбоңуз. Бул программаны жазганда 10 жана 5тин арифметикалык орточо мааниси 7,5 болот деп күтөбүз. Бирок, биз бул программаны иштетип жатканда, мындай натыйжага ээ болобуз:

```
12.5
```

Бул, албетте, туура эмес. Эмне үчүн мындай болду? Келгиле, эсептөөнү туура жасагандыгыбызды текшерүү үчүн математиканы текшерели.

Эгер биз бул теңдемени кагазга жазып жаткан болсок, анда a + b / 2 - дал өзүбүз көргөндөй кылып жазмакпыз. a + b 15ке барабар, экиге бөлүнгөндө 7.5 туурабы?

Эгерде сиз математика операторлору жана сандар боюнча талкуубузду эстесеңиз, математика программалоодо ар дайым эле калем менен кагазды колдонгондой иштей бербейт. Программалоодо артыкчылык ирети бар. Башкача айтканда, Python бир теңдемени карап, кийинки бөлүккө өтүүдөн мурун кайсы бөлүктү биринчи чечиши керек экендигин аныктайт.

Эгерде бул бөлүк түшүнүксүз болсо, мен операторлор жана номерлер бөлүмүнө кайтып, дагы бир жолу карап чыгууну сунуштайм. Муну түшүнгөдө кайтып келиңиз.

Python теңдемени биз каалаган тартипте аткарышы үчүн биз кашааны () колдонуп, кезектүүлүктүн тартибин орнотушубуз керек. Бул кодду катасы жок жазуунун чыныгы жолу:

```
a = 10
b = 5
average = (a + b) / 2
print(average)
```

Бул жолу сиз кодду иштеткен болсоңуз, төмөнкүнү аласыз:

7.5

Математикага жакын эмес адамдар же уйкусу канбаган адамдар, сиздер, бул коддун жакшы иштебегенин таптакыр эле бакабайсыңар. Python эч кандай эскертүү же ката жөнүндө билдирүү жөнөткөн эмес. Андыктан, натыйжаларды текшерип, алардын туура экенине ынанып, эки жолу текшерип көрбөсөңүз, көйгөй бар экендигин билүү үчүн бизде мындан башка эч кандай жол жок.

Эми элестетип көрсөңүз, бул банктык тиркеменин бир бөлүгү болгон болсо жана бул жөнөкөй логикалык ката бүтүндөй тутумду кандайча бузуп коёрун көрө алдыңыз. Ошондой эле, көптөгөн адамдарды аябай капалантты!

Өзгөчө кырдаалдар

Өзгөчө кырдаалдар - бул каталардын өзгөчө түрү. Камтылган өзгөчө кырдаалдардын бир нече түрлөрү бар. Бирок алардын саны көп болгондуктан бул бөлүмдө бардыгын камтый албайбыз. Анын ордуна, Python.org дарегинде жайгашкан өзгөчө кырдаалдарды камтыган https://docs.python.org/3/library/exceptions.html#bltin-exceptions баракчасына кирип, ар кандай түрлөрүн көрө аласыз. Мен болсо эмне үчүн ал сиз ойлогондон дагы пайдалуу болушу мүмкүн экендигин түшүндүрүп берем.

Азырынча, Python кодуңузду түшүнүп, бирок анын негизинде иш-аракет жасай албаганда, өзгөчө учурлар болорун билип алыңыз. Мисалы, сиз вебсайттан айрым маалыматтарды көчүрүп алуу үчүн интернетке туташууга аракет кылып жаткандырсыз, бирок туташа албай жатасыз. Же сизде көрсөтүлгөн даректе жок болгон API колдонууга аракет кылган скрипт бардыр.

Өзгөчө кырдаалдар синтаксистик каталардан бир нече жол менен айырмаланат. Алардын бири - алардын дайыма эле ката кетирбеши. Мунун да жакшы, жаман да жагы бар. Жакшы жагы - көйгөйлөрүңүз кээде өзгөчө кырдаалдарда дагы иштеши мүмкүн. Жаман жагы - сиздин программалар дагы өзгөчө кырдаалдарда иштей берет!

Биз кодубуздун каталар менен иштешин каалабайбыз!

Өзгөчө кырдаалдардын эң сонун жери - эгер сиз жакшы жагын карагыңыз келсе, биз өзгөчө кырдаалды жөндөө деп аталган нерсени жасай алабыз. Өзгөчө кырдаалды жөндөө дегенибиз биз ката кетиши мүмкүн деп күтүп, андан кийин аларды чечүү же жөндөө жолун скрипт кылабыз.

Бизде колдонуучудан төрт орундуу пин-код сураган программа бар дейли. Биз маани сан экенине ынангыбыз келет. Өзгөрмөбүздү бүтүн санга ылайыкташтырдык. Баштапкы коддон баштайлы:

```
пин = int(input("ПИН номериңизди териңиз: "))
print("Сиз терген ПИН номер: ", пин)
```

Эгер сиз бул кодду файлга киргизип, иштетип көрсөңүз, ал сизден пин-код киргизүүнү суранат. Улантып, байкап көрүңүз. Кааласаңыз, Oops.py файлыңызга кошсоңуз болот. Алгач, төрт орундуу санды киргизип, Enter баскычын басыңыз. Программа төмөнкүдөй жооп кайтарат:

```
_____________________________________________________________________

Сиз терген ПИН номер: 1234

_____________________________________________________________________
```

1234түн ордуна кайсы номерди киргизсеңиз, ошол болот.

Эми программаны дагы бир жолу иштетип көрүңүз, бирок бул жолу сураганда "abcd" деп жазып, Enter баскычын басыңыз.

Бул убактылуу чечим сизге төмөнкү натыйжаны берет:

```
_____________________________________________________________________
Traceback (most recent call last):
     File       "C:/Users/James/AppData/Local/Programs/Python/Python36-
32/Oops.py",
     line 17, in <module>
          пин = int(input("ПИН номериңизди териңиз: "))
ValueError: invalid literal for int() with base 10: 'abcd'
```

Бул жерде ValueError түрүндөгү өзгөчө кырдаалды көрөбүз. Себеби Python пинден табат деп күткөн маанинин түрү бүтүн сан болчу. Ордуна сиз сап киргиздиңиз.

Python бул сыяктуу катаны кетирбөөнү камсыз кылуунун бир жолу жана биздин программанын туура эмес иштешине жол бербөө үчүн катаны алдын ала чечүү керек дегенди билдирет. Кимдир бирөө биздин өзгөрмөбүзгө туура эмес маалымат түрүн киргизиши мүмкүн экендигин билгендиктен, ката пайда болгондо аны кармоо жана аны жөндөө үчүн кодду түзө алабыз.

Коддун башка версиясын алмаштырып, ушул кодду колдонуп көрүңүз, андан кийин файлды сактап, иштетип көрүңүз:

```
# ValueError менен иштөөнүн өзгөчө мисалы
try:
     пин = int(input("ПИН номериңизди териңиз: "))
     print("Сиз терген ПИН номер: ", пин)
except ValueError:
     print("Сиз сандык маани гана киргизишиңиз керек.")
```

Python программалоо тилинде бул try жана except блок деп аталат. Анын конкреттүү максаты - өзгөчө кырдаалды кармоо жана аны жөндөө. Блоктун ичиндеги код этияттулук менен ишгелип чыгышын талап кылат. Python ката кетсе, аны

жөндөөнү көздөп жатканыңызды түшүнөт. Эгерде ката бар болсо (сиз көрсөткөн типте), анда ал сиздин except операторуңузду жаратат.

Бул программаны иштетип, коддун кандайча иштей тургандыгын билүү үчүн кайра суралганда 'abcd' деп киргизиңиз. Сиз төмөнкүдөй жооп алышыңыз керек:

```
ПИН номериңизди киргизиңиз: abcd
```

Сиз сандык маани гана киргизишиңиз керек.

Python except операторуна келери менен, ал сиздин көрсөтмөлөрүңүздү аткарып, андан кийин программадан чыгат. Чындыгында, бул кодду өзгөчө учурларда кайрадан иштей тургандай кылып циклге камтыйбыз. Мисалы, жөнөкөй while циклин колдонсоңуз болот, мисалы:

```
# ValueError менен иштөөнүн өзгөчө мисалы:
кайталоо = 1
while кайталоо > 0:
      try:
            пин = int(input("ПИН номериңизди териңиз: "))
            print("Сиз терген ПИН номер: ", пин)
            кайталоо = 0
      except ValueError:
            print("Сиз сандык маани гана киргизишиңиз керек.")
            кайталоо = 1
```

Try except else блогу

Ошондой эле, Try Except Else блогун түзсөңүз болот. Эгер эч кандай өзгөчө кырдаалдар болбосо, анда код башка нускамаларды аткарат деп ойлойбуз. Мисалы:

```
# ValueError менен иштөөнүн өзгөчө мисалы
# Try Except Else блогун колдонуу
# while циклинде камтылган
кайталоо = 1
while кайталоо > 0:
      try:
         пин = int(input("ПИН номериңизди териңиз: "))
      except ValueError:
            print("Сиз сандык маани гана киргизишиңиз керек.")
            кайталоо = 1
```

```
    else:
        print("Сиз терген ПИН номер: ", пин)
        кайталоо = 0
```

Бул программанын мурунку версиясына окшош натыйжа берет жана окшош иштейт. Айырма? Бул бир аз тазараак жана окула турган нерсе. Негизинен, ал мындай деп окулат:

Бул кодду колдонуп көрүңүз. Эгер ал иштебесе:

Эгер өзгөчө кырдаал келип чыкса, айрым коддорду аткарыңыз.

Башка учурларда, өзгөчө учурлар жок болсо, ушул кодду иштетип коюңуз.

Finally билдирүүсү

Блогубузга дагы бир нерсе кошсок болот. Ал - ***finally (аягы).*** *Finally* эч нерсеге карабастан, алтургай, ката кеткенине карабастан, кандайдыр бир кодду иштетүү керек болгодо колдонулат.

```
# ValueError менен иштөөнүн өзгөчө мисалы
# Try Except Else блогун колдонуу
try:
    пин = int(input("ПИН номериңизди териңиз: "))
except ValueError:
    print("Сиз сандык маани гана киргизишиңиз керек.")
else:
    print("Сиз терген ПИН номер: ", пин)
finally:
    print("Биз дагы эле бүтө элекпизби?")
```

Бул кодду жакшыраак изилдөө үчүн while циклин жана кайталоого байланыштуу кайталанма өзгөрмө кодун алып салдык. Негизинен, бул коддо төмөнкүлөр айтылат:

Бүтүн сан маанисиндеги пин номерин сураңыз.

Эгерде пин номери бүтүн сан болбосо,

Except операторун иштетиңиз.

Антпесе пин номеринин маанисин басып чыгарыңыз.

Андан сырткары, кандай гана болбосун,

Акыркы пунктун аракетке келтирүү.

Эгер сиз 'abcd' деп киргизип, өзгөчө кырдаалды жаратсаңыз, анда бул коддун натыйжасы:

```
Сиз терген ПИН номер: abcd
Сиз сандык маани гана киргизишиңиз керек
Биз дагы эле бүтө элекпизби?
```

Эгер сиз дагы бир жолу иштетип, "1234" кодун пин катары киргизсеңиз, төмөнкүлөргө алып келиши мүмкүн:

```
ПИН номериңизди териңиз: 1234
Сиз терген ПИН номер: 1234
Биз дагы эле бүтө элекпизби?
```

Кандай болбосун, белгилей кетчү нерсе - биздин finally максатка ылайык келтирилген. Бул - өзгөчө кырдаалда программаңызды улантуунун эң сонун жолу.

Ыңгайлаштырылган өзгөчө учурларды түзүү

Камтылган өзгөчөлүктөрдүн аныкталган тизмесинен өзгөчө учурларды жөнгө салуудан сырткары, биз дагы колдонуучунун өзгөчө шарттарын түзө алабыз. Бул үчүн биз *raise* блогун колдонобуз.

```
баатыр = "Жөргөмүштөн корккон адам"
коркок = "Жөргөмүштөр"
if коркок == "Жөргөмүштөр":
    raise Exception("Ооба, рахмат ... менин атым баарын айтып жатат ...
    жаман адам жөргөмүштөргө тең келбеши керек!")
```

Бул жерде биз эки өзгөрмө түзүүнү баштайбыз. Бири биздин супер баатырдын аты - *баатыр*, экинчиси - *коркок* - терс каармандын түрү.

Андан кийин терс каармандын мааниси 'Жөргөмүштөр' маанисине барабар экендигин текшерген if билдирүүсүн жүргүзөбүз (кантсек да анын аты Жөргөмүштөн Корккон Адам да!), шумпай, чынында эле, биз өзгөчө кырдаал түзүү үчүн raise блогун колдонобуз:

Кодду иштеткенде, мен мындай ката алдым:

```
Traceback (most recent call last):
     File "C:/Users/James/AppData/Local/Programs/Python/Python36-
32/Oops.py",
     line 33, in <module>
          raise Exception("Ооба, рахмат ... менин атым баарын айтып
     жатат ... жаман адам жөргөмүштөргө тең келбеши керек!")
     Exception: Ооба, рахмат ... менин атым баарын айтып жатат ...
жаман адам жөргөмүштөргө тең келбеши керек!
```

Эскертүү: Бул мисалда сиз саптын номерине маани бербей койсоңуз болот, анткени менин файлымда башка код бар. Ал ката сизде башкача пайда болот.

Бул жерде биз кээ бир текстти басып чыгарып, биздин өзгөчө катабыз көрсөтүлүп жатканын көрөбүз:

```
     Exception: Ой, рахмат ... менин атым баарын айтып жатат ... жаман
адам жөргөмүштөргө тең келбеши керек!
```

Бул мисалда мен күлкүлүү болууга аракеттендим, андыктан өзгөчө нерсени тамашага салдым. Чындыгында, сиз өзүңүздүн өзгөчө кырдаалдарыңызды түзүп жатканда, төмөнкүдөй сапаттарда дагы бир нерсени айтышыңыз керек:

```
     Өзгөчө кырдаал: коркок өзгөрмөсү жөргөмүштөр деген мүмкүн болбогон
маани камтыйт.
```

Ошентип, кимдир бирөө туура эмес маани киргизсе, өзгөчө кырдаалды карап чыгып, көйгөйдүн эмне экендигин изилдебей туруп, дароо билебиз.

Ыңгайлаштырылган өзгөчө кырдаалдын дагы бир түрү - бул AssertionError өзгөчө кырдаалы. Бул өзгөчө кырдаал, берилген шарттын аткарылгандыгын же канааттандырылгандыгын ырастаган программаны иштетет. Андай болсо, программа иштей берсе болот. Болбосо, AssertionError өзгөчө кырдаалы алып ташталат.

Коддун бул кыска үзүндүсүн карап көрөлү:

```
assert  1 + 1 == 2, "Бир кошуу бир 2ге барабар!
assert  2 + 2 == 5, "2 + 2 бешке барабар эмес! Error in line 2!!"
```

Бул жерде бизде эки ырастоочу билдирүү бар. Эгерде биз бул программаны иштетсек, анда программанын 1-сабында эч нерсе болбойт. Себеби 1 + 1 туюнтмасы, чындыгында, 2ге барабар. Демек, тастыктоо шартын текшерүү - True.

Коддун экинчи сабы аткарууга аракет кылганда, assert сыноо шарты False (2 + 2 бешке барабар эмес) экендигин далилдейт. Ошондуктан AssertionError иштетилип, натыйжада, мындай жыйынтык чыгат:

```
Traceback (most recent call last):
  File "C:/Users/James/AppData/Local/Programs/Python/Python36-32/Oops.py",
  line 2, in <module>
       assert 2 + 2 == 5, "2 + 2 does not equal five! Error in line 2!!"
AssertionError: 2 + 2 does not equal five! Error in line 2!!
```

Ыңгайлуу болуш үчүн, мен биздин кодубуздагы ката ырастоонун жыйынтыгында жазылган сапты, ошондой эле AssertionError себебин коштум.

Logging

Биздин коддогу мүчүлүштүктөрдү табуу үчүн дагы бир курал бар. Ал, айрыкча, узун программалар үчүн колдонулат. Мунун бир нече жолу бар. Бирок оңою - logging модулун импорттоо.

Программисттердин коддогу каталарды азайтуу үчүн колдонгон ыкмаларынын бири болгон print() бардыгы талаптагыдай иштеп жаткандыгын текшерет. Мисалы, менин статистикамдын тобу бар деп коёлу. Биздин Super Hero Generator 3000 тиркемесиндегидей ал да кокустан түзүлгөн.

Мен жөн гана менин кодум туура иштеп жатат деп ишенип, статистика туш келди туура түзүлүп жатат деп ойлойм. Бирок бул эң акылдуулукка жатпашы мүмкүн. Баарын туура коддогонума ынануу үчүн мен бул ысымдардын кокустан пайда болушун жана андан кийин убактылуу статистиканын натыйжаларын чыгаруу үчүн кандайдыр бир код киргизишин каалашым мүмкүн. Кокустан санды түзүүнүн талаптагыдай иштеп жаткандыгына көзүм жеткенден кийин мен бардык print() саптарын өчүрүп, өз кодумду уланта алам.

Мисалы, мен бул кодду мындай деп баштасам болот:

```
import random
зээндүүлүк = 0
тапкычтык = 0
чыдамдуулук = 0
акылмандык = 0
күч = 0
дисциплина = 0
эптүүлүк = 0
ылдамдык = 0
```

```
тапкычтык = random.randint(1,20)
зээндүүлүк = random.randint(1,20)
чыдамдуулук = random.randint(1,20)
акылмандык = random.randint(1,20)
дисциплина = random.randint(1,20)
эптүүлүк = random.randint(1,20)
ылдамдык = random.randint(1,20)
```

Андан кийин бардык маанилерди туш келди текшерилиши керектигин түшүнүп, артка кайтып, print() функцияларын кошуу үчүн кодумду түзөтүп алсам болот:

```
import random
зээндүүлүк = 0
тапкычтык = 0
чыдамдуулук = 0
акылмандык = 0
күч = 0
дисциплина = 0
эптүүлүк = 0
ылдамдык = 0
зээндүүлүк = random.randint(1,20)
print("Зээндүүлүк: ", зээндүүлүк)
тапкычтык = random.randint(1,20)
print("Тапкычтык: ", тапкычтык)
чыдамдуулук = random.randint(1,20)
print("Чыдамдуулук: ", чыдамдуулук)
акылмандык = random.randint(1,20)
print("Акылмандык: ", акылмандык)
дисциплина = random.randint(1,20)
print("Дисциплина: ", дисциплина)
эптүүлүк = random.randint(1,20)
print("Эптүүлүк: ", эптүүлүк)
ылдамдык = random.randint(1,20)
print("Ылдамдык: ", ылдамдык)
```

Андан кийин программаны бир жолу иштетип, менин өзгөрмөлөрүмдө маанилердин сакталып калгандыгын көрүп, мындай натыйжага жетсем болот:

```
Зээндүүлүк:  10
Тапкычтык:  12
Чыдамдуулук:  20
Акылмандык:  7
Дисциплина:  5
Эптүүлүк:  16
Ылдамдык:  19
```

Андан кийин маанилердин кошулгандыгын текшерип бүткөндөн кийин программа иштелип чыккан сайын маанилердин туш келди чыгып жаткандыгына ынануу үчүн дагы бир текшерүүдөн өткөрмөкмүн. Тест эң эле жөнөкөй: эгер экинчи жолу маанилер башкача болсо, анда бардыгы иштейт. Жыйынтыктары кандай болот?

```
Зээндүүлүк:  18
Тапкычтык:  14
Чыдамдуулук:  1
Акылмандык:  5
Дисциплина:  8
Эптүүлүк:  1
Ылдамдык:  4
```

Эки сыноонун тең ар бир өзгөрмөсү үчүн маани ар башка болгондуктан, мен кокустукту туура колдоном деп ойлойм. Мындан ары print() функцияларымдын кереги жок. Мен аларды комментарийлеп же толугу менен алып салсам болот.

Бул жөнөкөй коддук бөлүк болгондуктан, мен кодумдун окулушун жеңилдетүү үчүн print() функциясын алып салам.

Файлымды бир топ print() функциялары менен жасалган башаламандыктын ордуна мен журналды колдонуп, файлды көзөмөлдөп, натыйжаларын өзүнчө тексттик файлга жазып алсам болот.

Мындай текшерүү жургүзүүнүн дагы бир артыкчылыгы – биз кодубузда болуп аткан окуялар менен каталарды каттап турабыз. анткени кийинчерээк ката чыгып калса же кайра карап чыгуу керек болуп калса бизге оңой болот.

Эскерте кетүүчү нерсе, logging эскертүүлөрдү жана каталарды көзөмөлдөө гана эмес, ошондой эле окуяларды байкоо үчүн да пайдалуу.

Чындыгында, logging модулу “маани деңгээли” рейтингдерине ээ. Сиз аны колдоно аласыз. Алар:

Критикалык: Программа олуттуу көйгөйлөргө алып келиши же таптакыр иштебей калышы мүмкүн болгон олуттуу каталар үчүн колдонулат.

Ката: олуттуу, олуттуу эмес көйгөйлөр үчүн.

Эскертүү: күтүлбөгөн нерсе болгондо же болушу мүмкүн болгондо колдонулат.

Маалымат: Сиздин кодуңуз иштелип чыккандыгын тастыктоо иретинде колдонулат. Биздин print() операторубузду колдонууга окшош

Мүчүлүштүктөрдү оңдоо: Мүчүлүштүктөрдү аныктоодо жана мүчүлүштүктөрдү оңдоо процессинде пайдалуу маалыматтарды берүүдө пайдалуу.

Чындыгында, *logging* модулун колдонуу бул китепке кирбейт. Анын колдонулушун туура түшүндүрүп берүү үчүн бир бүтүндөй бөлүмдү үйрөнүү талап кылынат. Ал эми жаңы үйрөнүүчүлөргө *logging* үйрөнүүнү сунуш кылганым менен бул биздин окуу программабызга туура келбейт.

Айткандай эле, бир аз убакыт бөлүп, расмий документтештирүү жана *logging* модулун окуп чыгыңыз. Ошондой эле, Интернеттеги жана башка, өркүндөтүлгөн китептердеги айрым окуу куралдарын карап, татаал программаларды түзүп жатканда, *logging* менен баштаңыз.

Кадам таштаңыз, убактысы келгенде тереңирээк сүңгүйсүз.

Мүчүлүштүктөрдү оңдоо: Debugging

Кодуңуздагы каталарды оңдоо, кодуңузду текшерүү жана өзгөчө кырдаал менен иштөөнү кантип жасоо керектиги жөнүндө көп ирет айттык. Ошондой эле, каттоо, каталарды жана окуяларды байкоо үчүн logging модулун пайдаланууну негизги түшүнүгү жөнүндө сүйлөштүк.

Коддоо көйгөйлөрүн чечүү үчүн супер баатыр жеңибизди дагы бир түрүнүп - *мүчүлүштүктөрдү оңдоочу- debugger* деп аталган куралды колдонобуз. Python үчүн көптөгөн оңдоочулар бар. Алардын ар биринин өзүнүн күчтүү жана алсыз жактары бар. Айрымдары Python программалоо тилинин белгилүү бир тармактарын камтыйт. Адистешүүнү тандайт, калгандарынын башка мүчүлүштүктөрдү оңдоочу программаларга окшош функциялары бар.

Чындыгында, Python тилинин pdb деп аталган өзүнүн мүчүлүштүктөрдү оңдоо куралы бар.

Техникалык жактан pdb - бул импорттоого жана колдонууга боло турган модуль. Модуль программаларыңызга кирип, алардын сапаттуу иштешин текшерип, саптан сапка текшерүүгө мүмкүндүк берет.

Биздин статистикалык кокустук маанилердин туура берилгендигин текшерүү үчүн print() операторлорун колдонгон мурунку мисалыбыз эсиңиздеби? Pdb мүчүлүштүктөрдү оңдогуч модулун колдонуп, ушул бардык print() операторлорун жазбай эле бирдей натыйжага жетише аласыз.

Python мүчүлүштүктөрүн оңдоочу pdb модулу жөнүндө Python документтештирүү веб-сайтынан биле аласыз. Жөн гана көрүп жаткан документтин версиясы компьютериңизге орноткон Python версиясына дал келиши керек. Python 3.6 шилтемеси, мисалы:

https://docs.python.org/3.6/library/pdb.html

Ошондой эле, ал Python 3.7, анда pdb модулун бир аз жаңырткан breakpoint() командасынын формасында болот:

https://docs.python.org/3.7/library/pdb.html

logging сыяктуу эле, мүчүлүштүктөрдү таап, мүчүлүштүктөрдү оңдоону изилдеп, азыртан баштап анын негиздерин үйрөнүшүңүз керек. Андыктан татаал программаларды түзүп жатып, кайсынысын тандасаңыз да, өзүңүзгө ыңгайлуу болсун. Азырынча pdb модулуна көңүлүңүздү бурмакмын.

Ката менен иштөө боюнча акыркы кеңеш

Эгерде мен буга чейин мындай деп айтпаган болсом, анда сиздин кодуңуздагы каталарды таап, аларды чечүү үчүн акыркы бир кеңешти айткым келди. Сиз комментарийлерди колдонуңуз.

Бирок бул эмнени билдирет болду экен?

Концепция жетиштүү деңгээлде жөнөкөй: эгерде коддун бир бөлүгү сизге көйгөй жаратып жатат деп шек санасаңыз, анда коддун сабын көрүнбөй тургандай кылып (#) комментарийди колдонуп, кодуңузду иштетип, көйгөйдүн бар же жогун билиңиз. Эгер ал көйгөйүңүздү таппаса, эгер көйгөй чечилбесе, коддун кийинки бөлүмүнө өтүңүз.

if блоктор сыяктуу татаал конструкциялар үчүн көп бөлүмдүү комментарийлерди (""") колдонуп, бөлүмдү толугу менен комментарийлеп бериңиз. Мисалы:

```
"""
IF
код
код
код
"""
```

үч тырмакчанын (""") ортосунда коддун комментарийлери.

Бул - бардык деңгээлдеги программистер колдонгон жалпы практика. Текшерип же оңдогондон кийин кодуңузга комментарий калтырууну унутпаңыз!

Бул эпизоддо

Биздин ушул жерге жеткенибизге ишенесизби? Экөөбүздүн супер баатырдын укмуштуу окуясын чогуу аяктаганга бир гана бөлүм калды!

Супер! Мыкты! Укмуштуудай таасирлүү! Таң калардык! Аябагандай таасирдүү!

Бум! Бам! Иихоо! Пампарапам!

Бул бөлүмдө каталар жөнүндө сөз болгон. Аларды табуу, оңдоо, каттоого алуу жана мүчүлүштүктөрдү оңдогонду түшүндүрдүк. Эс алуу учурунда айрым негизги учурларын карап чыксаңыз болот.

Ушундан кийин гана акыркы бөлүмгө аттанабыз!

- Python программалоо тилинде каталардын үч түрү бар. Алар: синтаксистик каталар, логикалык каталар жана өзгөчө учурлар.
- Синтаксистик каталар орфографиялык же грамматикалык катага окшош. Алар, адатта, фаталдуу болуп, сиздин кодуңуз аткарылбай калат.
- Логикалык каталар сиздин программалоо логикаңызда кемчилик болгондо пайда болот. Алар ар дайым байкаларлык ката кетиришпейт. Бул көп учурда программалардын таң калыштуу иштешине жана бузулушуна алып келет.
- Өзгөчө кырдаалдар ар дайым ката кетире бербейт. Көбүнчө программалар өзгөчө учурду карабастан деле иштеши мүмкүн.
- Орнотулган өзгөчө учурлардын көптөгөн түрлөрү бар. Анын ичинде ValueError жана NameError.
- Камтылган өзгөчө учурлардан тышкары, raise жана assert ыкмаларын колдонуп, өзүбүздүн да өзгөчө учурларды жасай алабыз.
- Try-except-else-finally блоктору сизге каталарыңыз кандайча иштелип жаткандыгын көзөмөлдөп, критерийлерге же каталардын түрлөрүнө туш болгондо эмне болорун айтууга мүмкүндүк берет.
- Өзгөчө кырдаалды жөндөө - бул иштөө процесси же аны менен мамиле кылуу.
- Logging кодуңуздагы каталарды, эскертүүлөрдү, мүчүлүштүктөрдү оңдоо маалыматтарын жана окуяларды көзөмөлдөөгө мүмкүндүк берет. Бул журнал файлдарын келечектеги маалымат үчүн өзүнчө файлга сактасаңыз болот.
- Python программалоо тилинде мүчүлүштүктөрдү оңдогонго жардам берген же мүчүлүштүктөрдү таап, оңдогон көптөгөн куралдар бар.

- Python программалоо тилинин орнотулган мүчүлүштүктөрүн оңдоочу модулу - pdb.

- Сиздин программаларыңызда ката кетириши мүмкүн (же мүмкүн эмес) болгон коддордун блокторун комментарийлөө үчүн бир саптуу комментарий жана көп саптуу комментарийди колдонсоңуз болот. Андан кийин комментарий берилген бөлүктөрдүн туура же туура эместигин билүү үчүн кодуңузду текшерип көрсөңүз болот.

14- БӨЛҮМ

PYTHON КАРЬЕРАСЫ

Азаматсыз, жаш баатыр! Бул сапарыбыз узак болду. Биз Джек Хаммер жана карасанатай Алгебро сыяктуу кара ниет көптөгөн душмандарды жеңдик. Биз бул мистикалык томду окуп чыгып, мурда белгисиз болгон өз күчүбүздү бийиктикке чыгарууга мүмкүндүк берген түшүнүккө жана дааныштандыкка ээ болдук. Бул жерде Эверест чокусу сыяктуу чокулардын түрлөрү жөнүндө сөз болуп жатат.

Жок дегенде, сиз, оюн аянтчасындагы бийик слайддын үстү жагына чыктыңыз деп коёлу.

Эмнеси болсо да, биз бул укмуштуу окуяны алгач чогуу баштаганда сиз катмарлашкан кир жабышкан шымыңыз жана бырышкан чапан менен дос болуп жүргөнсүз. Сиздин маскаңыз жаркырак болсо да, жүзүңүздү араң жаап турчу.

Бирок, эми сизге көз чаптырып көрсөк! Укмуштуудай күчкө толгон толук кандуу баатырга айландыңыз. Эми сиз өзүңүздүн программаларыңызды түзө аласыз. Видео оюндарды жаза аласыз. Компьютерлерди бузуп-оңдоп, математиканын мыкты жетишкендиктерин жасай аласыз. Статистиканы кокустан түзө аласыз жана башка көптөгөн нерселерди жасай аласыз.

Сиз жаш баатырдан суперге чейин жеттиңиз. Катардагы окуучудан мыкты, жогорку деңгээлдеги окуучу болдуңуз.

Бирок, негизинен, сиз окурмандан программистке айландыңыз. Досум, бул китептин максаты ушул болчу.

Ошондой эле, сиз ушул чоң туңгуюктун жанында турганыңызда супер баатырдын жакшынакай кийимдери денеңизге бап келип, жаңы күч-кубатыңыз менен билимиңизди иш жүзүндө колдонуп жатканыңызда эч качан эс албаңыз. Дүйнө - дайыма өзгөрүлүп турган пейзаж. Ошондой эле, технология да барган сайын кулачын жайып баратат. Python болсо аягы көрүнбөгөн ар дайым өнүгүүдөгү жырткычка окшош.

Ушул себептен мурунку алган билимиңизди экинчи тилге айланганга чейин колдоно беришиңиз керек. Сиз код менен түш көрүүңүз керек! Андан дагы улантып, көбүрөөк билип, кыялданыңыз!

Программалоодо сиз таба турган жана өсө турган дагы көп нерселер бар. Бул китеп айсбергдин чокусу эле. Сизди китептен үйрөнө албаган практикалык, чыныгы тажрыйба күтүп турат. Программалоо тилинин жаңыртылган версиялары сизди күтөт.

Ошондой эле, башка тилдер да сизди күтүп жатышы мүмкүн.

Мен сизди билимиңизди кеңейтип, эч качан токтоп калбоого чакырам. Башка тилдерди караңыз. Python менен абдан окшош жана үйрөнүүңүзгө оңой болгон Perl программасын үйрөнүп көрүңүз. Айрыкча, веб колдонмолорун программалоого өтүүнү кааласаңыз, Ruby on Rails жана PHP - да келечектеги эң сонун тилдер,

C жана C ++ бир аз татаалыраак. Бирок сиз жөн гана негиздерин үйрөнүп жаткан болсоңуз да, үйрөнүү үчүн көп күч-аракет жумшашыңыз керек. Сиз иштеп жаткан учурда HTML, JavaScript жана JSON - резюмеңизге жана чеберчилик топтомуңузга кошууга боло турган ыңгайлуу куралдар.

Резюме жөнүндө айта турган болсок, бул акыркы бөлүм бир чыныгы максатты көздөйт. Анын максаты - сизди программалоонун чыныгы дүйнөсүнө даярдап берүү. Сиз 13 же 14 жашта болосузбу, эртеби-кечпи мансапка байланыштуу кандай багытты көздөй турганыңызды чечишиңиз керек болот. Сизге азыр кандай варианттар бар экендигин билүү маанилүү. Бул сизди келечектеги окуу жолуна багыт берет.

Мисалы, эгер сиз программалоо оюндарын улантууну чечсеңиз, анда Pygame программасын үйрөнүү жана Scratch менен тажрыйба жүргүзүү сөзсүз жардам берет. C, JAVA жана C ++сыяктуу тилдерди кошуу өзгөчө талап болуп саналат.

Бул бөлүмдө биз чоңойгондо эмне кылууну каалаганыңыз жөнүндө ой жүгүртүүгө жардам берүү үчүн учурдагы жана келечектеги бардык кесиптик жолдорду карап чыгабыз. Ошондой эле, биз буга чейин чоңойгон жана ошол эсептерди төлөп башташы керек болгон адамдар үчүн маектешүүнүн мүнөздүү суроолорун камтыйбыз!

Бул бөлүмдө биз чоңойгондо эмне кылышыбыз керек экенин ойлонуп, баштоого жардам берүүчү учурдагы жана келечектеги кесиптердин бардык варианттарын карап чыгабыз. Ошондой эле, маектешүүнүн жалпы суроолорун карап чыгабыз.

Ошондой эле, биз мыкты код жазууну улантып, жумушка орношкондон кийин аларды сактап калуу үчүн программалоонун мыкты тажрыйбалары жөнүндө билимибизди жаңыртабыз. Коддоо принциптеринин маанилүүлүгүн жеткиликтүү айта албайм. Дүйнөнүн келечеги ушундан көз каранды!

Келечек жөнүндө айтсам (ооба, мен Segue падышасымын!), биз Python программалоо тилинин келечегин тил катары карайбыз. Виртуалдык чындык - кеңейтилген чындык, жасалма интеллект жана башка бир катар терминдерди айтканда жана укканда ал кулакка укмуштуудай угулат.

Акырында, биз бөлүмдү Python терминдеринин баракчасы менен жыйынтыктап, Python жана жалпы программалоого байланыштуу адамдардын эң көп берилген суроолоруна жооп беребиз.

Бул узак жол болду. Эми мындан ары муну улантууга эч кандай себеп жок. Супер баатыр өтүктөрүңүздү кийип, алардын боосун бекем байлаңыз! Бул сапарга чекит коюуга убакыт келди.

Супербаатыр стили!

Python менен иштөө

Сиз бул китепти колуңузга алганыңызда карьера жөнүндө ойлошуңуз же ойлобошуңуз да мүмкүн. Бул - кадимкидей эле көрүнүш. Көптөгөн адамдар "чоңойгондо" эмне иш кылууну ойлонушпайт. Бул маселени чоңойгонго чейин калтырып коюшат!

Ким болгуңуз келерин билүүнү кааласаңыз дагы каалабасаңыз да же кайсы кесипти аркалайын десеңиз да, бир нерсе белгилүү. Сиз ал нерсенин эмне экенине көңүл бурасыз. Буга сиздин ушул китепке инвестиция кылганыңыз, эң негизгиси, өзүңүзгө инвестиция кылганыңыз күбө болуп турат. Сиз бул баракчаларды окуп, кодду сынап көрүүгө убакыт бөлөсүз. Буга чейин муну сиздин курактагы көптөгөн адамдар жасаган эмес. Сиздин жолуңуз болду!

Кийинки кадам - алган билимиңиз менен эмне кылууну кааларыңызды билүү. Сиз үйрөнгүңүз келген тармакка же башка тилдерге жана көндүмдөргө карабастан, Python иштеп чыгуучусу катары уланта берүүнү кааласаңыз керек.

Карьераңызда жасаган айрым иш-аракеттериңиз сиз тандаган факторлордон башка факторлорго байланыштуу болот. Жолуккан адамдар, жашаган жерлериңиз жана тапкан жумуштарыңыз ар дайым жолдун кайсы жакка алып барышына себепчи болот. Сиз видео оюнунун программисти болом деп ойлоно башташыңыз мүмкүн. Тажрыйба топтоо жана ошол жолду тандап алуу үчүн оюн сыноочу катары видео оюнун иштеп чыгуучу компанияда иштейм деп тургандырсыз. Ким билет? Балким, бул жашоонун укмуштуу окуясынын бир бөлүгү чыгар!

Бул белгилүү бир максатка умтулууга, алтургай, ага бекем турууга болбойт дегенди билдирбейт. Сиздин пландарыңыз канчалык жакшы болбосун, кээде өзүңүз болжолдогондон башкача жерде болуп каларыңызды билиңиз. Бул, албетте, жакшы.

Кандай болгон күндө да кайда барууну кааларыңыз тууралуу түшүнүк болсо жакшы болот. Ушуну эске алып, жарым – жартылай супер баатыр же толук кандуу программист болгондо карьера тандоонун айрым жолдорун карап көрөлү.

Python үчүн карьера жолдору

Бул бөлүмдө келтирилген мансапка жетүү жолдору кандайдыр бир тартипте жайгаштырылган эмес. Бири-биринен жакшыраак эмес. Бирок айрым жерлерде эмгек акынын деңгээли башкаларга караганда жогору экендигин байкасаңыз болот. Биз буга көңүл бурбайбыз. Сиз жасаган ишиңизди сүйөрүңүзгө бекем ишенем. Эгер ошондой болсо, анда ийгиликке жетесиз.

Бул тизме сиздин каалоолоруңуздун бардыгын камтыбайт. Сиз тандай турган бир нече эле кесип бар. Бирок алар - учурда эң кеңири тараган кесиптер.

Бета тестер

Бета тестерлер - программалык камсыздоону иштеп чыгуучулар дүйнөсүнүн көзгө көрүнбөгөн каармандары. Алар программаларды жана программалык камсыздоону текшерип, техникалык жактан дагы, колдонуучунун тажрыйбасы жагынан да эмненин иштей тургандыгын жана эмненин иштебей тургандыгын аныкташат. Айрым учурларда сизден программанын белгилүү бир өзгөчөлүгүн же аспектсин атайын текшерип чыгууну талап кылышы мүмкүн. Башкаларында сиз баарын текшерип чыгышыңыз мүмкүн.

Бул кесип үчүн программалоо боюнча билимдин болгону маанилүү. Бирок ал эң маанилүүсү эмес. Мен программалоо тили жагынан реалдуу тажрыйбам жок эле көптөгөн программалардын бета версияларын сынап көрдүм. Мен түшүнүктөр жана нерселердин кандайча иштээрин түшүндүм. Бул максатта ал мага жакшы кызмат кылды.

Албетте, эгер сиз тилди билсеңиз жана коддогу маселелерди так аныктай алсаңыз, анда ишиңиз оңолуп, жумуш табуу оңоюраак болот.

Мүмкүн, сиздин буга чейин бета тестирлөөдөн өткөн программаңыз бар. Бирок аны толугу менен ишке ашыра элексиз. Эгер сиз видео оюндарды жакшы көрсөңүз же мобилдик оюндардын күйөрманы болсоңуз, анда көпчүлүк учурда бета нускасын жалпы элге жарыялаганга чейин байкап көрөсүз. Бул таптакыр акы төлөнүүчү жумушка окшош болбосо дагы, акысыз программалык камсыздоо же жабдыктар сыяктуу артыкчылыктар болушу мүмкүн.

Коддогу катаны оңдоочу

Бул бета тестирлөөчүдөй сезилиши мүмкүн. Бирок көпчүлүк учурда бир аз татаалыраак келет. Сиздин миссияңыз – туура эмес кодду таап, аны кантип оңдоо керектигин айтып берүү же тапшырмага жараша оңдоо.

Эгер сиз программанын туура эместигинин сырын чечүүгө бир нече саат сарптаганды жактырган адам болсоңуз же бир нерсени бөлүп-бөлүп алууну кааласаңыз, бул сиз үчүн жакшы вариант болушу мүмкүн. Кесип жолуңуз кайда алып барбасын, жок дегенде, бул - жакшы чеберчилик.

Башка адамдардын кодун жана көбүнчө *бир нече* адамдын кодун көрө турганыңызды унутпаңыз. Бул адамдар документтештирүүнү жана стандарттык эрежелерди сактоону жакшы билишет деп үмүттөнөм. Бирок сиз эмнеге туш болоруңузду билбейсиз.

Бирок, бул оюнуңуздун үстүндө жүрүүнүн мыкты жолу жана программалардагы мүчүлүштүктөрдү табууну үйрөнүүдө, программалык камсыздоону иштеп чыксаңыз же өзүңүздүн тиркемелериңизди түзсөңүз пайдалуу болот.

Маалыматтарды иштеп чыгуу жана талдоо боюнча адиси

Эгерде сиз статистика, сандар жана изилдөөлөрдү жакшы билсеңиз, анда маалымат таануу жаатына кирүүнү ойлонушуңуз мүмкүн. Python маалымат илиминин дүйнөсү эбегейсиз зор, статистика жана машинаны үйрөнүү техникасынын айкалышынан турат.

Математикалык жана маалыматтарды визуалдаштыруу инструменттеринин (мисалы, matplotlib жана NumPy) чоң китепканасынын жардамы менен Python программисттери маалымат илиминин карьерасы боюнча алдыңкы орундарды ээлешет. Бул иш чөйрөсүндө сиз тармактардын жана колдонмолордун кеңири чөйрөсү үчүн маалымат топтомдорун уюштурууга, чечмелөөгө жана көрсөтүүгө жардам берүү үчүн графиктерди жана башка куралды колдоносуз.

Сиз иштеп чыккан алгоритмдер жана маалыматтарды чечмелөө уюмга же бизнеске чечимдерди кабыл алууга жардам берет. Бул карьера үчүн сизге аналитикалык мээ, жакшы математикалык көндүмдөр жана, албетте, бир аз программалоо боюнча ноу-хау керек болот. Бирок, чындыгында, ал маалымат эмнени билдирерин билип алууну жакшы көргөндөр үчүн пайдалуу багыт болорун убада кылат!

Программа иштеп чыгуучу / программалык камсыздоо инженери

Программалык камсыздоону иштеп чыгуучуга келгенде, көптөгөн мүмкүнчүлүктөр бар. Бул нерсенин жалпы схемасында кайда болоруңуз жөнүндө ойлонгондо биринчи кезекте сиз ойлогон роль ушул болсо керек.

Программалык камсыздоону иштеп чыгуучулар көптөгөн программалык камсыздоолорду, анын ичинде өндүрүмдүүлүк тиркемелерин (мисалы, Microsoft Office), музыка түзүү программаларын жана сиз ойлогон нерселердин бардыгын түзүшөт. Компьютериңиздеги тиркемелерге көз чаптырып көрсөңүз, ал спектрдин канчалык деңгээлде кең экендигин жакшы түшүнүп каласыз.

Эгер сиз программалык камсыздоону иштеп чыгуучу же программалык камсыздоо инженери болууну чечсеңиз, анда Python жана башка тилдер жөнүндө мүмкүн болушунча көбүрөөк билүү керек экенин унутпаңыз. Башка тилдерди жана алкактарды билүү эч качан зыян келтирбейт жана бонус катары эсептелет. Python тилин билгенден кийин экинчи же үчүнчү программалоо тилин үйрөнүү бир топ жеңилдейт. Анткени логика жана структуралардын көпчүлүгү тилдерде бирдей. Кеп негизинен жаңы синтаксисти жана программалоонун стилдерин үйрөнүүдө гана болот (мисалы, бардык тилде чегинүү колдонулбайт).

Видео оюн программисти

Бул техникалык жактан программалык камсыздоону иштеп чыгуучу менен бирдей жолдо болсо да, мен аны атайын белгилеп кетүү керек деп ойлодум. Видео

оюндун күйөрманы катары, бул - баарынан мурда мени программалоого аралаштырган нерсе. Эгер мен аны карьеранын өзүнчө варианты деп эсептебесем, анда мен үчүн кечиримсиз болмок.

Акыркы он жылдын ичинде видео оюндун өнүгүшү, чындыгында, гүлдөп-өстү. Мен колледжде окуп жүргөндө көбүнчө адистештирилген курстарды сунушташкан. Оюндарды өнүктүрүү боюнча курстарды, бир аз азыраак деңгээлде сунуш кылган бир нече гана колледждер бар болчу. Чындыгында, менин колледжим мындай курстун бирөөсүн гана сунуш кылчу жана ал жылына бир жолу гана болчу!

Албетте, ал кезде биз кодубузду ири таштарга чегип, мурдубузга сөөктөрдү тагып жүрчүүбүз, бирок ошентсе да...

Эгер сиз ири консолдор үчүн иштеп чыгууну кааласаңыз, анда Python жана Pygame оюндарынан көбүрөөк билишиңиз керек болот. Чындыгында, Python сизге талап кылынган логиканын бир бөлүгүн түшүнүүгө жардам берет, бирок сиз негизинен C ++ тилине кирип, C жана JAVA менен иштөөгө аттанышыңыз керек.

Эгерде сиз консолу жок оюн же PC оюну боюнча тандап алсаңыз, сизде дагы көп мүмкүнчүлүктөр болушу мүмкүн, бирок чындыгында C ++ ушул китепти жазуу учурунда колдонулуп жатат.

Мобилдик колдонмо

Python мобилдик колдонмону иштеп чыгуу үчүн сиз ойлой турган биринчи тил болбосо да, бул тилди колдонмолорду түзүү жана / же мобилдик колдонмолорду иштеп чыгууга ылайыкташтырылган башка тилдер менен байланыштыруу үчүн колдоно аласыз.

Мобилдик колдонмолор сиз телефонуңузда же планшетиңизде колдонгон бардык колдонмолордон турат. Бул оюндар, мессенджердик тиркемелер, жаңылыктарды окуучу тиркемелер, банктык программалар, жадакалса, веб-сайттардын мобилдик версиялары болушу мүмкүн. Тизме улана берет.

Эгер сиз ушул маанилүү жана чоң рынокту тандасаңыз, анда мобилдик өнүгүү үчүн чыныгы кубаттуу тилдерди үйрөнсөңүз болот: C # же Objective-C, C ++, JAVA, Swift, жада калса, HTML5. Жөнөкөйлүк үчүн HTML5 менен баштасаңыз болот. Анткени тизмедеги башка тилдерге караганда аны үйрөнүү жеңилирээк болушу мүмкүн. Веб иштеп чыгуу үчүн HTML5 да колдонсоңуз болот. Себеби, мобилдик колдонмону иштеп чыгуу сиз үчүн эмес деп тапсаңыз, анда ал сиздин арсеналыңызда кала турган абдан ыңгайлуу курал болот.

Албетте, C ++, C жана JAVA сиз үчүн дагы башка эшиктерди ачат, бирок аларды үйрөнүү бир аз татаалдаштырылган. Ошондуктан бардыгы сиздин убакытыңызга жана муктаждыктарыңызга байланыштуу.

Кандай болбосун, көп колдонулбаса дагы, Python программалоо тилин мобилдик колдонмо үчүн колдоно берсеңиз болот.

Веб иштеп чыгуу жана веб тиркемелер

Эгерде сиз вебке негизделген тиркемелерди түзгүңүз келсе, анда Python бул иште сөзсүз жардам берет. Чындыгында, Python тилинин күчтүү пункттарынын бири - Django жана Flask сыяктуу күчтүү желе алкактары.

Бул алкактар колдонмолоруңуздун сөөктөрүн тез жайгаштыруу үчүн бир катар пландардын же скелеттердин ролун аткарат. Ошону менен сиз көптөгөн орнотуу жана коддоо убактысын үнөмдөйсүз. Негизинен, алар веб-тиркемелерден сиз таба турган негиздерди түзүшөт, андыктан сизге дөңгөлөктү кайрадан ойлоп табуунун кажети жок.

Python программалоо тилин жана желе алкактарын HTML5 жана JavaScript менен айкалыштырасыз. Интернет дүйнөсүндө сизге ишенүүгө туура келет. Мисалы, Google, YouTube жана Yahoo платформалары Python тилине таянышат. Эгер бул сизге Python канчалык деңгээлде жакшы экендигин айтпаса, анда эмне дээримди билбейм!

Системаны башкаруу

Системдик администраторлору кызыктуу. Алар ар кандай уюмдун өтө зарыл бөлүгү болушат. Жана Python, сиз болжолдогондой эле, системдик администраторлоруна жумушун бүтүрүүгө жардам берүү жагынан өзгөчө мыкты.

Системдик администраторлор компьютердик системаларды башкарууга, операциялык системаларды башкарууга жана тармактык тапшырмалар менен иштөөгө жардам берген шаймандарды жана утилиталарды түзүүдө Python колдонушат.

Ошондой эле, өз серверлериңизди жана кардарларыңызды, билдирүү системаларын жана башкаларды түзүүгө мүмкүнчүлүк берет. Python –- системдик администраторлордун эң жакын досу.

Эмнегедир мышыктар дагы алардын достору...

Изилдөө, окутуу жана башкалар

Маалымат илимпоздору бөлүмүндө талкуулангандай, Python - мыкты изилдөө куралы. Комплекстүү теңдемелерди жана маалыматтар топтомун иштетүү үчүн китепканалар жана куралдар ушунчалык көп болгондуктан, мекемелердин бул тилге ушунчалык таянганы таң калыштуу эмес.

Андан сырткары, мектепте же университетте Python сабак берүү - бул ар дайым жан багуунун жана келечектеги муундарга билимди өткөрүп берүүнүн мыкты жолу. Аны үйрөнүү жана кошумча продукт катары үйрөтүү оңой болгондуктан, информатика сабагынын талаптарына ылайыкташтырылган биринчи тепкич болуп саналат.

Аны жетиштүү деңгээлде үйрөтсөңүз, балким, бир күнү бул жөнүндө өзүңүздүн китебиңизди жазсаңыз болот. Ким билет?

Кана аны чийип салыңыз. Китепти жазууну мага калтырыңыз. Менин бага турган иттерим бар!

Жумушка кирүүдө Python боюнча суралган жалпы суроолор

Бул китепти окуп жаткан айрымдарыңызга маектешүү учурунда кандай суроолорду сурашат деп тынчсызданууунун кажети жок. Башкалар үчүн бул эртедир-кечтир өтө олуттуу көйгөйгө айланат. Кандай болсо да, сиз баштоого даярсызбы же мындай нерсени ойлоно элексизби? Биз убакыт бөлүп, ушул бөлүмдө Python маегинин жалпы суроолорунун тизмесин карап чыгабыз.

Бул китепте ушул темалардын көпчүлүгү чагылдырылса да, бир топ темалары камтылбай калган. Эсиңизде болсун, бул - Python программалоону баштоо үчүн эмне керектигин үйрөтүү үчүн башталгыч китеп. Бул сизди толук куралдантып, жумушчу күчүнө жөнөтүү эмес.

Эгерде кандайдыр бир мааниси жок терминдер же идеялар болсо, биз аларды Google, башка китептерден издеп, мүмкүн болушунча көбүрөөк билүүгө чакырабыз. Бул суроолор жана жооптор сизге алдап жумушка кирүү үчүн гана бул жерде керек эмес. Анткени сиз жаттап алууну мыкты билесиз. Чындыгында, булар - жалпы суроолор, түшүнүктөр программалоонун маанилүү принциптери болуп саналат.

Ошондуктан, бул суроолордун жообун билүү, андан ары окуу жана практика аркылуу түшүнүү сиз даяр болгондо гана жумушка орношууга жардам бербестен, ошол жумушта калууга, алтургай, гүлдөп-өсүшүңүзгө жардам берет!

Python программалоо тилинин айрым негизги өзгөчөлүктөрүн айтып бере аласызбы?

Бул - жөнөкөй алаксытуучу суроо. Python канчалык деңгээлде билериңизди, тилге канчалык кызыкдар экениңизди жана анын жалпы өзгөчөлүктөрүн канчалык деңгээлде билериңизди байкап көрүшөт. Алардын айрымдарын көрсөтө алышыңыз мүмкүн, бирок эң кеңири колдонулгандары төмөнкүлөр:

- Python - бул ар тараптуу багыттарда колдонууга жөндөмдүү, көп максаттуу тил. Анын ичинде маалымат таануу, этикалык хакерлик, системдик администрация, веб иштеп чыгуу, мобилдик тиркемелерди иштеп чыгуу, видео оюндарды программалоо, илимий моделдөө жана башкалар.
- Python - жогорку деңгээлде окула турган, үйрөнүүгө жеңил, бирок күчтүү тил. Бул - динамикалык түрдө объектке багытталган тил (бул сиз жарыялаган өзгөрмө маалыматтардын түрлөрүн аныктоонун кажети жок дегенди билдирет. Python көпчүлүк учурда сиз каалаган маалыматтардын түрүн аныктай алат).

Кортеж менен тизменин айырмасы эмнеде?

Бул суроону мурунку бөлүмдө талкуулаганбыз жана анын жообу төмөндөгүдөй: Кортеждер өзгөрүлгүс, башкача айтканда алардын маанилерин өзгөртүү мүмкүн эмес. Ошол эле учурда, тизмелер өзгөрүлмө, демек, алардын маанилерин өзгөртө аласыз. Дагы бир айырмачылыгы, кортеждерге тегерек кашаа () талап кылынат. Ал эми тизмелерде төрт бурчтуу кашаа колдонулат []. Бирок тизмелер техникалык жактан кортеждерден жайыраак .

Мурас деген эмне?

Мурас түшүнүгүн биз объектилер жана класстар жөнүндө сөз кылган бөлүмдө караганбыз. Эсиңиздерде болсо, класстар ата-эне менен баланын мамилесине окшоп, иерархиялык түзүлүштү карманат. Бизде ата-эне болгондо - же суперкласс - ошол ата-эненин бала классы ата-эненин касиеттерин жана методдорун тандап алат.

Эске салсак, класстар жана объектилер - объектиге багытталган программалоонун (ООР) негизги өзгөчөлүгү. Алардын бардыгы кодду кайра колдонууга байланыштуу болот. Бала классы бир ата-эне классынан же бир нече ата-эне классынан мураска алат. Бул чоң ийкемдүүлүктү жана коддун жогорку натыйжалуулугун камсыз кылат.

Python тилинде кокустук маанилерин кантип жаратасыз?

Бул китепте биз көп колдонгон маанилүү модул random болгон. Бул биздин каармандардын статусун биздин супербаатыр жаратуучу программабызда тандоо / рандомизациялоо үчүн өтө маанилүү болгон. Ошондой эле алардын каармандарынын аттарын жана алардын жөндөмдөрүн тандоо үчүн колдонулган.

Аны колдонуу үчүн, алгач аны импорттошубуз керек:

```
import random
```

Андан кийин биздин коддо колдонулат. Мисалы, биз мындай деп жаза алабыз:

```
import random
a = random.randint(1, 10)
print(a)
```

Бул a өзгөрмөсүндө 1ден 10го чейинки туш келди маанини сактап, андан кийин басып чыгарат.

Python программалоо тилинде кантип тизме, кортеж жана сөздүк түзсө болот?

Бул жөнөкөй суроо сыяктуу сезилиши мүмкүн. Бирок эгерде аны ошол жерден жасашы керек болсо, анда ал программисттин башын айландырышы мүмкүн.

Ошондуктан ар бирин жасап, аларды качан колдонууну билип алсаңыз, бул сиз үчүн экинчи мүнөзгө айланат.

Жооптору:

```
менинТизмем = ['Жеймс', 'Ход-Дог', 'Жөргөмүш', 'Мистер Кунг Фуд']
менинКортежим = (' Жеймс ', 'Ход-Дог', 'Жөргөмүш', 'Мистер Кунг Фуд')
менинСөздүгүм = {'Жазуучу' : 'Жеймс Пейн', 'Студент' : 'Сиздин атыңыз'}
```

Локалдык өзгөрмө менен глобалдык өзгөрмөнүн айырмасы эмнеде?

Локалдык өзгөрмөлөр функциянын чегинде колдонулат. Башкача айтканда, функциянын ичинде өзгөрмө түзөбүз. Эгерде өзгөрмө функциянын сыртында аныкталса, анда ал глобалдык деп эсептелет.

Python сунуш кылган маалыматтардын түрлөрү кайсылар?

Python программалоо тилинде маалыматтардын төмөндөгүдөй жалпы беш түрү бар: сандар, саптар, тизмелер, кортеждер жана сөздүктөр.

Графикалык интерфейс (GUI) деген эмне? GUI иштеп чыгуу үчүн кайсы Python китепканасы мыкты?

Бул эки бөлүктөн турган суроонун жөнөкөй жообу бар. GUI колдонуучунун графикалык интерфейсин билдирет жана баскычтар, этикеткалар, текст кутучалары, белгилөө кутучалары, радио баскычтар жана башка ушул сыяктуу нерселерди программаңызга киргизүүгө мүмкүнчүлүк берет.

GUI иштеп чыгуу үчүн Python тилинин демейки китепканасы Tkinter деп аталат.

Python программалоо тилинде файлды кантип ачасыз?

Бул китепте биз козгогон дагы бир тема бар. Эсиңизде болсо, биз файлды ачуу үчүн open() функциясын колдонобуз. Алгач, файлдын аталышын жана жайгашкан жерин, андан кийин файлды кайсы режимде ачарыбызды көрсөтөбүз. Мисалы:

```
файлым = open("test.py', 'w')
```

Негизги тутумда жайгашкан test.py файлын ачат.

Модулдун функцияларын кантип санап берет элеңиз?

Жалпы маектин дагы бир суроосу бар. Ушул модулдагы функциялардын тизмесин кандайча көрө аласыз. Бул үчүн dir() ыкмасын колдонобуз:

```
import random
print dir(random)
```

help() колдонуу дагы модулдун ичиндеги документтерди көрүү үчүн пайдалуу.

Python тууралуу башка суроолор

Жумуш маегинде Pythonго байланыштуу кандай суроолор берилерин так билбейбиз. Андыктан маектешүүдөн мурун аларды жакшылап окуп чыгыңыз. Бул жерде камтылган бир топ суроолор бар. Бирок сизден дагы башка көп нерселерди сурашы мүмкүн.

Андан тышкары, сизден кодго байланыштуу айрым суроолорго жооп беришиңизди же белгилүү бир функцияны көз ачып жумганча аткарылышы үчүн код жазууну сурашы мүмкүн. Негиздерин жазууга даярданыңыз жана кеңири жайылган, орнотулган модулдарды жана функцияларды билип алыңыз.

Жумуш маегине даярдануунун эң сонун жолу - бул компанияны жана ошол жерде жасай турган программалоонун түрлөрүн изилдөө. Мисалы, эгер компания веб-тиркемелерди иштеп чыкса, анда сизден веб-фреймерлер тууралуу сурай тургандыгын алдын ала билесиз.

Акырында, Python же программалоого тиешеси жок суроолорго жооп берүүгө ар дайым даяр болуңуз. Бул суроолор дагы берилет. Карьералык максаттар, мурунку тажрыйба, жеке мүнөздөгү суроолор жана жалпы мүнөз, мамилелер бардыгы маектешүү учурунда каралат. Андыктан маектешүүнүн негизги даярдыгын көз жаздымда калтырбаңыз.

Кулагыңыздын артын дагы тазалаңыз... Келечектеги кожоюнуңуз кулагыңызды текшерип көрүшү мүмкүн!

Программалоонун мыкты тажрыйбалары

Программалоонун көпчүлүгү жеке тандоо болсо да, жумушка орношкондо ар дайым сиз сактоого тийиш болгон стандарттар бар. Жакшы жана туура документтештирүүнүн маанилүүлүгүн талкууладык. Бул бөлүм эң мыкты программалоонун тажрыйбалары жөнүндө болмокчу.

Бул бөлүмдөгү кеңештер сизди мыкты программист кылууга, эффективдүүлүгүңүздү жогорулатууга, жалпы тузактардан алыс болууга жана коддоодо каталарыңызды азайтууга жардам берет. Бул толук тизме эмес, бирок сизди супер баатырдай программалоо жолуна салышы керек!

Стиль боюнча көрсөтмөлөрдү аткарыңыз

Чексиз акылмандык менен Python ойлоп табуучусу өзү айткан стилдер боюнча көрсөтмө түзгөн. Python сыяктуу эле, PEP же Python программалоо тилин өркүндөтүү сунуштары деп аталган бул стилдик колдонмо - ар кандай темалардагы

сунуштардын тизмеси. Мында мурунку (алынып салынган) модулдардан баштап, стилдик көрсөтмөлөргө жана тил эволюциясы боюнча колдонмолорго чейин камтылган.

Бир катар түзмө-түз PEP мисалдары бар. Мисалы, стиль боюнча көрсөтмө PEP 8 болуп саналат жана ал алгач биздин улуу жол башчыбыз Гидо Ван Россум, Барри Варшава жана Ник Коглан тарабынан 2001-жылы түзүлгөн.

Анда кодуңузду кантип жайгаштыруу керек, өтмөктөрдү же бошлуктарды колдонуу же колдонбоо, кодуңуздун максималдуу узундугу, сап тырмакчалары менен иштөө жана башка ушул сыяктуу көптөгөн маселелер камтылган.

Жумушка орношуп жаткан учурларда сизден PEP менен, айрыкча, чегинүү жана ат коюунун шарттарын камтыган бөлүмдөр менен таанышасыз деп күтүшөт. Бул сиздин кодуңузду карап чыгып, иштешкен кесиптештериңизге гана жардам болбостон, PEP стили боюнча колдонмо сизге кодду жакшыраак түзүүгө гана эмес, аны натыйжалуу жана катасыз түзүүгө жардам берерин айгинелейт.

PEP 8 Python.org веб-сайтында бар: www.python.org/dev/peps/pep-0008/.

Бардык тизмесин төмөнкү шилтемеден таба аласыз:

www.python.org/dev/peps/.

Мисал катары, PEP 8 ат коюуга байланыштуу мындай дейт: Класстар: ысымдагы биринчи жана экинчи сөзгө (жана андан кийинки сөздөргө) баш тамгаларды колдонуңуз. Мисалы: *VillainType же МутантКласс.*

Өзгөрмөлөр, функциялар, методдор, модулдар жана пакеттер: кичинекей тамга менен жазылган сөздөрдү астынкы сызыктар менен ажыратыңыз. Мисалы: *менин_атым же my_villain_name.*

Эгер ал бузулган болсо, азыр оңдоңуз (кийинчирээк эмес)

Көбүнчө, биз программаны ийгиликтүү илгерилетип жатканда, алдыга умтулууну каалайбыз. Бул, айрыкча, офис чөйрөсүндө, убакыттын тардыгын сезип, коддун бир бөлүгүн бүтүрүп, кийинки бөлүгүнө өтүү үчүн сизге убакыт жетпей баштаганда болот.

Бирок, бул чоң көйгөй жаратышы мүмкүн. Кодубуздагы майда мүчүлүштүктөргө көңүл бурбай, аларды кийинчерээк оңдоп алсак болот деп ойлоп, азгырыктарга кабылышыбыз мүмкүн. Бирок чындыгында мындай ой жүгүртүү жардамчы эмес, тескерисинче, тузак болорун эсиңизден чыгарбаңыз.

Бул жерде же ал жерде ката болушу мүмкүн. Бирок, карлуу тоодогу кар көчкү сыяктуу, алар тез эле үйүлүп, жолундагы бардык нерсени жок кыла башташат. Ката көп учурда башка каталарга алып келип, домино эффектисин жаратат. Эгер коддун бир бөлүгү туура иштебесе же ката кетсе, анда ал башка бөлүмдөрүндө таң калыштуу жыйынтыктарды жаратышы мүмкүн. Андан да жаманы, жабыр тарткан бөлүмдөр эскертүүлөрдү же каталарды пайда кылбашы мүмкүн, бул акыр-аягында дагы көптөгөн көйгөйлөргө алып келет.

Бул жердеги сабак жөнөкөй: кодуңузду тез-тез текшерип туруңуз. Эгер сиз мүчүлүштүктү тапсаңыз, аны токтоосуз оңдоңуз жана ал көйгөй чечилмейинче алдыга жылбаңыз. Мага кийинчерээк рахмат айтасыз, мага ишениңиз!

Документтештирүү – баарынан маанилүү

Китепте биз ушул нерсеге бир нече жолу токтолгонбуз, бирок бул жерде дагы бир жолу кайталай кетүү керек. Ар дайым кодуңузду документтештирип алыңыз. Так документация - бул ийгиликтүү программанын ачкычы. Ал баштапкы версиясына жана кийинки версиясына дагы тиешелүү.

Белгилүү болгондой, Python программалары миңдеген саптар кодунан турушу мүмкүн. Атүгүл, миллиондогон саптардан да турушу мүмкүн. Башка бирөөнүн каттарын же электрондук почталарын окудуңуз беле? Адамдар сиз менен бир тилде сүйлөшсө да, ар дайым түшүнүктүү боло бербейт. Python айырмаланбайт. Ар бир код (же баары бир) салттуу ат коюу эрежелерин жана код структурасын карманууга аракет кылып, чындыгында, көптөгөн программистер өз алдынча үйрөнгөндүр.

Ошондой эле, биз убакыттын өтүшү менен жалкоо жана өзүбүзгө өтө эле ишенип калабыз. Биздин кодду караган адам биздин эмне кылганыбызды түшүнөт деп ойлойбуз. Андан да жаманы, биз бир нече жыл мурун жасаган нерсебизди эстеп калабыз деп ойлойбуз.

Кодуңузду документтештирүү бир аз көбүрөөк убакытты талап кылса, келечекте ал толгон токой убакытты үнөмдөйт. Анткени ал каталарды издегенге кетирген убактыңызды жана код каталарыңызды кыскартат. Ошондой эле, сиз коддун бөлүктөрүн тез эле кайра колдоно аласыз. Менин китебимде (акыры бул менин китебим экени чын да) документтештирүү эң маанилүү деп саналган, сиз колдоно турган мыкты тажрыйба.

Биз "документтештирүү" дегенде # комментарийди же """көп саптуу комментарийди гана эмес, докстринди туура колдонууну да камтыйбыз.

Кесипкөй программист болуп калыптанганыңызда сизде дагы колдоно турган шаймандар болот. Бирок азырынча негиздеринен баштаңыз жана ар бир жазган кодуңузду документтештирип көрүңүз.

Код кампаларын жана таңгактарды колдонуңуз

Python колдонуунун эң чоң сатылуучу пункттарынын бири - бул Python иштеп чыгуучулар жамааты тарабынан түзүлгөн жана сыналган Python пакеттеринин чоң китепканасына кирүү мүмкүнчүлүгү.

Бул код жана функция бөлүктөрү сиз каталарды жана өз долбоорлоруңузда иштеп жатканда кайгы-капаны азайтууга жардам берет, убакытты үнөмдөйт. Айткандай эле, дөңгөлөктү эмне үчүн кайрадан ойлоп табыш керек?

Пайдалануу үчүн топтомдорду https://pypi.org дарегинде жайгашкан Python Package Index (PyPi) кампасынан таба аласыз. Учурда 300000 колдонуучу катышкан 155000 долбоор бар.

Эгер сиз издеп жаткан нерсеңизге ишенбесеңиз же идея издесеңиз, анда долбоорлорду издеп, карап чыксаңыз болот же популярдуу долбоорлордун тизмесин көрө аласыз.

Программаларыңызга жардам берүү үчүн топтомдорду издөөдөн тышкары, PyPi веб-сайтында башкаларга тестирлөө жана колдонуу үчүн өз пакеттериңизди таңгактоону жана хостингди жүргүзүүнү үйрөнсөңүз болот.

Мен сизди ушул сайтка тез-тез кирип, Python жамаатынын башка мүчөлөрү эмне кылып жаткандыгын карап чыгууга чакырам.

Ушул китептен бир нече таңгакты орнотуу үчүн pip кандайча колдонулары эсиңизде болсо керек. Бул - пакеттер алынган репозиторий.

Тез-тез текшерип туруңуз

Кайра кайталоо керек болгондуктан, бир же эки абзацты алганга туура келип калат. Кодуңузду сынап көрүңүз. Аны тез-тез текшерип туруңуз.

Качан жаңы өзгөрүүлөрдү жасасаңыз же башка бөлүм кошсоңуз, мурунку кодду сынап көрүңүз. Бул if блогу же кичинекей цикл сыяктуу жөнөкөй нерсе болсо дагы текшериңиз. Эгер сизде чечим кабыл алууга таянган коддун бир бөлүгү же шарттуу билдирүү болсо, ар бир мүмкүн болгон жоопту текшерип көрүңүз.

Мисалы, эгерде сизде if блогу болсо, анда "Эгер ооба болсо, анда X, эгерде жок болсо, анда Y, болбосо z" деп айтылган болсо, анда бул шарттардын ар бирине жооп бергениңизди текшерип көрүңүз. Тесттерде этият болуңуз, жогоруда айтылгандай, эскертүүлөргө же каталарга туш болсоңуз, оңдоп алыңыз.

Ошентип, аны оңдогондон кийин, дагы бир жолу сынап көрүңүз.

Чегинүү же бошток

Бул бизди стилдик колдонмолор жана PEP көрсөтмөлөрү жөнүндө маегибизге кайтып алып келет. Код жазганда ар дайым чегинүү бошкугун же табулатураны колдоно турганыңызды тандап алыңыз.

Андан кийин ушул чечимди бекем карманыңыз.

Аларды колдонуу боюнча талаш-тартыштар бар. Мен аларды өз көзүм менен көрдүм. Күндүн аягында мен Python колдонуучуларынын жарымына: “сиз вариантыңызды ырааттуу колдонсоңуз, бул эч кандай мааниге ээ эмес” деп тамашалаштым.

Жеке тандоодон жана PEP көрсөтмөлөрүнөн сырткары, сиз иштеген кайсы гана уюм болбосун өзүнүн стилдик эрежелерине ээ болорун жана бул нерселер

баарынан - сиздин жеке каалоолоруңуздан жана башкалардан - жогору турарын унутпаңыз.

Бирок, дагы бир жолу, коддоо учурунда ар дайым бирдей аралык / табулатура шарттарын колдонуңуз.

Класстар жакшы, бирок бардыгы бирдей болушу шарт эмес

Python программалоо тилинде кандайдыр бир структураны же объектини программалаган сайын ал жасап жатканыңызга ылайыктуубу же жокпу деп ойлонуп көрүңүз. Мисалы, класстар кайрадан колдонууга сонун, бирок функциялар дагы эле ошондой. Модулдар менен дагы ушундай эле болот.

Кантсе да, сиздин эң башкы милдетиңиз - нерселерди мүмкүн болушунча жөнөкөй кылуу. Эгер сиз ушуну жасасаңыз, анда бул китепте биз көп айткан максаттарга жете аласыз. Алар төмөнкулөр:

Көп жолу колдонула турган код;

Каталарды азайтуу;

Натыйжалуу код.

Жөнөкөйлүктүн дагы бир артыкчылыгы - бул бардык нерсени окууга ыңгайлуу кылат. Ошондуктан мунун маанилүүлүгүн айтпай кетүүгө болбойт. Бир нерсени окуу канчалык оңой болсо, сизге жана сиздин кесиптештериңизге маселелерди көзөмөлдөө же коддун бөлүктөрүн кошуу ошончолук оңой болот. Бул - өтө көп класстарды жана модулдарды колдонуунун терс жактарынын бири. Алар көп жагынан жакшы болгону менен, Python кодунун окулушун бузушат.

Аларды кандай гана болбосун колдонуңуз. Жөн гана алардын зарыл экендигине жана сиз жетишүүгө аракет кылып жаткан нерсени жасоого эң оңой жол экенине ынаныңыз.

Python программалоо тилинин келечеги

Python планетада эң көп колдонулган программалоо тили экени талашсыз. Көптөн бери болуп келген бул тенденция басаңдаган жок. Тилди үйрөнүү оңой, күчтүү жана ийкемдүү болгондуктан, жакынкы келечекте анын популярдуулугун жоготуп алуу мүмкүнчүлүгү өтө төмөн.

Python менен тез арада өнүгүшү күтүлүп жаткан бир нече тармак бар. Бул жарым-жартылай ушул айдыңдардын же тармактардын популярдуулугун жогорулатуу менен байланыштуу. Башкаларды Python тилинин жалпы аренада мыкты болгондугу өзүнө тартып турат.

Мунун бир мисалы - маалыматтарды анализдөө, изилдөө жана илимий программалоо үчүн тиркемелер. Бул аренада кыймылдаткыч күчкө ээ болгон тил, маалымат илиминин куралы катары колдонуу жагынан өсөт.

Python программалоо тилинин өнүгүшүнө өбөлгө түзгөн дагы бир жагдай - Python 2нин негизинде тиркемелерди куруп жаткан көптөгөн корпорациялар бар. Python 3түн туруктуулугуна байланыштуу бул компаниялар өз коддорун жаңыртып, Python 3кө өткөрө башташкан. Бул коддун таптакыр жаңы топтомуна өтүүгө караганда, негизинен, жеңилирээк процесс.

Албетте, Python кол тийгис эмес. Python сөзсүз түрдө өсүшү керек болгон тармактар бар. Алардын бири - мобилдик тиркеме. Бирок, бул тармактан алыс болуунун ордуна, мобилдик тиркемелерди иштеп чыгууда жана ушул тармакты чечүүгө жардам берүүчү шаймандар жөнүндө сөз болгондо Python жамааты көтөрүлүп, чаңда калтырбайт деп күтсөк болот.

Программалоону үйрөнүүнүн кайсы баскычында экениңизге жана технологияга болгон көз карашыңызга жараша - ушул тапта эшигиңизди каккылап жаткан жасалма интеллект (AI), виртуалдык чындык (VR), кеңейтилген чындык (AR) сыяктуу жогорку технологиялык багыттар, өсүп келе жаткан IoT (Интернет нерселери) тармагы бар. Акылдуу үйлөр жана туташкан шаймандар тездик менен өнүгүп келе жаткан базар бар. Ошондуктан Python бул аралашманын бир бөлүгү болот деп ишенсеңиз болот.

Кантсе да, Python үйрөнүүнүн жана колдонуунун жеңилдиги аны кеңири колдонула турган тилге айландырды. Эмне үчүн? Мунун жообу жөнөкөй: эгер сизде бизнес болсо, анда Python программасын тез үйрөнө турган адамды жалдай аласыз. Анын ийкемдүүлүгү, Python коомчулугу тарабынан иштелип чыккан топтомдору жана ушул китепте аябай талкууланган бардык башка сонун нерселер менен айкалыштырыңыз. Менин акчам Python тилин программисттер үчүн кыймылдаткыч кылып сактаганга жумшалып жатат.

Сиздики да болушу керек.

Python терминдери

Python программалоо тилин үйрөтүүдө талкууланган бир топ терминдер бар. Бул китеп канчалык ар тараптуу болсо да, алардын бардыгын камтыбайт. Ушул китептеги маалыматтарды жалпылоо жана бир нече жаңы терминдерди үйрөтүү үчүн ушул бөлүмдү программист катары иштеп атканыңызда кезигиши мүмкүн болгон Python терминдеринин айрымдарын аныктоого жардам беребиз.

***Аргумент**: Функцияга берилген маани. Ошондой эле параметр катары белгилүү.*

***Маани берүү (assign)**: өзгөрмө, тизме, сөздүк, кортеж же башка объектке маани берүү.*

***Маалыматтын логикалык түрү (boolean)**: True (Чын) же False (Жалган) жыйынтыгына барабар маани.*

Класс: Сиз классты бир объекттин планы деп эсептесеңиз болот. Суперкласстар же негизги класстары жана субкласстары бар. Субкласс негизги классынын белгилерин, методдорун жана атрибуттарын мураска алат. Ушул долбоорлорду колдонуп, бир же бир нече класстын негизинде тез арада объекттерди түзсө болот.

Комментарий: Комментарий документтин же коддун бир бөлүгү же үзүндүсү эмне үчүн колдонулгандыгын түшүндүрүп берүү үчүн колдонулат. Сиз #, андан кийин боштук мейкиндигин колдонуп, комментарий жазсаңыз болот. Андан кийин бир саптуу комментарий үчүн текст жазсаңыз болот, мисалы:

 # Бул комментарий.

Python # белгисин көргөндө боштуктун артынан ошол сызыктагы нерселердин бардыгына көңүл бурбай, өзүңүзгө же башка программисттерге жазууларды калтырууга мүмкүнчүлүк берет. Эгер сизге көбүрөөк орун керек болсо, анда кийинки комментарийлердин ар бири үчүн # колдонууну уланта берсеңиз болот же көп саптуу комментарий үчүн ' ' ' же " " " белгилерин колдонсоңуз болот. Бул жерде көп саптуу комментарийдин мисалы келтирилген:

```
' ' '
Бул комментарий.
Бул дагы комментарий.
Бул жерде ...
' ' '
```

Шарттуу оператор: Белгилүү бир шарттын аткарылышына же аткарылбастыгына байланыштуу аткарыла турган же аткарылбай турган билдирүү.

def: def функцияны аныктоо же түзүү үчүн колдонулат. Көбүрөөк билүү үчүн термин функциясын караңыз.

Сөздүк: Сөздүк - бул бир же бир нече ачкыч мааниси жуптарынан турган маалыматтардын түрү. Бул учурда ар бир ачкыч мааниге туура келет. Сөздүктүн негизги бөлүгү өзгөрүлбөйт, башкача айтканда аны өзгөртүүгө болбойт. Сөздүктөрдөгү маанилер кандай гана болбосун - саны, сабы жана башка нерсеси болушу мүмкүн. Аларды өзгөртүүгө болот. Сөздүккө төмөнкүдөй аныктама беребиз:

```
example_dict{'Аты' : 'Мурат', 'Жашы': '23'}
```

Бул сөздүккө эки элементти бөлүп берет. Биринчи ачкыч - маани жубунун аты: Мурат. Бул жерде Аты - ачкыч, Мурат болсо мааниси. Экинчи ачкычы - жаш, мааниси - 23.

docstring: Документтик сап - Python программасына, модулуна же функцияларына киргизилген документтердин бир бөлүгү.

калкыма чекит: ондук сан, мисалы, 2.5 же 02.19.

Функция: Функция - бул сиз программаңыздан чакыра турган код. Адатта, биз бир нече жолу колдоно турган коддун үзүндүлөрүн сактоо үчүн функцияларды колдонобуз. Функцияны мындайча аныктайбыз:

```
def функциянын_аталышы(parameters):
```

```
    # Сиздин кодуңуз ушул жерге жазылат. Мисалы:
    print("Карачы, мен функциямын!")
```

Функцияны чакыруу үчүн биз мындай теребиз:

```
функциянын_аталышы ()
```

Өзгөрүлбөс: Эгерде кандайдыр бир нерсе өзгөрүлбөс болсо, анда анын маанисин өзгөртө албайбыз дегенди билдирет.

Импорт (import): Китепкананы программаңызга жүктөө.

бүтүн сан(integer): бүтүндөй сан, мисалы 1, 400, 20,000, ал тургай - 50,000.

итерация: циклдин дагы бир аталышы.

len: len() функциясы объекттин узундугун, мисалы, өзгөрмө, тизмедеги нерсени ж.б. эсептөө үчүн колдонулат. Эгер тизмеде колдонулса, ал нерселердин же элементтердин санын эсептейт. Эгер сапта колдонулса, анда ал саптагы белгилерди санап берет. Бул жерде бир нече мисал келтирилген:

```
a = "Бул менин өзгөрмөм"
тизме = [1,2,3,4,5]
len(a)
len(тизме)
```

Жыйынтыгы:

```
18
5
```

Эскертүү! Len() боштуктарды дагы эсептейт .

Тизме: Тизмелер - бул каалаган типтеги маанилердин иреттелген тобун сактай турган Python маалыматтарынын түрү. Кортеждерден айырмаланып, тизмелер өзгөрүлмө. Демек, алардын мааниси өзгөрүшү мүмкүн.

Тизме түзүү үчүн төмөнкүнү терсеңиз болот:

```
менин_тизмем = [0,1,2,3,4,5]
менин_кийинки_тизмем = ['Жеймс', 'Супер Жигит', 'Акылдуу бала']
```

Цикл: Бир цикл критерийлердин жыйындысына жараша берилген коддун бөлүгүн кайра аткаруу же кайталоо үчүн колдонулат. for цикли бар, аны канча жолу айтсаңыз да кайталанат. Ал эми шарт True болсо, кайталана турган while цикли бар.

for циклинин мисалы:

```
for i in range(0, 5):
       print(i + 1)
```

Натыйжада:

```
1
2
3
4
5
```

While циклинин мисалы:

```
while a == 4:
        print("a 4кө барабар!")
```

метод: объектке таандык функция.

өзгөрүлүүчү: маанисин өзгөртүүгө болот.

объект: класстын планын колдонуп түзгөн нерсе.

параметр: аргументтин башка аталышы (айрымдар мындай салыштыруу менен макул болбошу мүмкүн).

print(): print() функциясы колдонуучунун экранына бир нерсени көрсөтүүгө же чыгарууга мүмкүнчүлүк берет:

```
print("Саламатсыңбы Дүйнө!")
```

натыйжасы:

```
Саламатсыңбы Дүйнө!
```

сап: Ар кандай тамгалардан, сандардан, боштуктардан же атайын белгилерден турган маалыматтардын түрү.

синтаксистик ката: Эгер сиз туура эмес текст киргизип, коддун бир бөлүгүн туура эмес жазганыңызда беттешкен ката.

көзөмөлдөө: ката кетирген функциялардын ырааттуу тизмеси.

кортеж: кортеж - бул каалаган типтеги маанилердин иреттелген жыйнактарын сактаган маалыматтардын түрү. Тизмелерден айырмаланып, алар өзгөрүлбөйт жана маанилерин өзгөртүү мүмкүн эмес.

Сиз төмөнкүдөй кодду колдонуу менен кортеж түзө аласыз:

```
менин_кортежим = ('Математика', 'Шаар', 'Тентек окуучу')
менин_кийинки_кортежим = ('0', '1', '2', '3', '4')
```

өзгөрмө: өзгөрмө - бул бир гана маанини сактаган маалыматтардын түрү. Бул маани сан, символ, сүйлөм жана башкалар болушу мүмкүн. Аларда тизмелер, сөздүктөр, алтургай, функциялар камтылышы мүмкүн.

Өзгөрмө түзүү үчүн дайындоо операторун колдоносуз:

```
a = 12
b = "Педагогика илимдеринин кандидаты"
```

АЛФАВИТТИК КӨРСӨТКҮЧ

S

T, U, V

W, X, Y, Z

А, Б, В, Г, Д

Ж, З, И

К

Л, М, Н

О, Ө

П, Р, С

Т, У

Ф, Ч, Ш, Ы

Өспүрүмдөр үчүн Python

Супер Баатырдай программалоону үйрөнүңүз!

Жеймс Р. Пейн

(Кыргыз тилинде)

Которгон *Шурубү Каримова*

Техникалык редактор *Муратбек Касымалиев*

Басманын редактору *Миргүл Низаева*