КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКА ДЛЯ СТУДЕНТОВ ПЕДАГОГИЧЕСКОГО НАПРАВЛЕНИЯ НА ДВУЯЗЫЧНОЙ (ТАТАРСКО-РУССКОЙ) ОСНОВЕ

учебное пособие

ИКЕТЕЛЛЕ (ТАТАРЧА-РУСЧА) НИГЕЗДӘ ПЕДАГОГИКА ЮНӘЛЕШЕ БУЕНЧА УКУЧЫ СТУДЕНТЛАР ӨЧЕН ИНФОРМАТИКА

уку ярдәмлеге

КАЗАНЬ

2024

УДК

ББК

Авторы-составители:

А.В. Данилов, Т.Р. Фазлиахметов

Компьютерная вёрстка: Т.Р. Фазлиахметов

Рецензенты:

Нуриев Н.К. — профессор, доктор педагогических наук, рочетный работник высшего профессионального образования РФ, профессор кафедры информатики и прикладной математики ФГБОУ ВО «Казанский национальный исследовательский технологический университет

Төзүче-авторлар:

А.В. Данилов, Т.Р. Фазлиахметов

Компьютерда текстларны һәм битләрне төзү: Т.Р. Фазлиахметов

Информатика для студентов педагогического направления на двуязычной (татарско-русской) основе: учебное пособие. — Казань, Казанский федеральный университет, 2024 – 169 с.

ISBN

УДК

ББК

Кереш сүз

Бүгенге көндә мәгарифтә цифрлы технологияләр куллану глобаль тенденциягә әйләнде. Белем бирүнең сыйфатын арттыру һәм мәгълүмат белән эшләү күнекмәләрен үстерү өчен, заманча компьютер технологияләрен нәтиҗәле куллану зарур.

Әлеге ярдәмлек заманча фәнни алымнарны һәм практик күнекмәләрне үз эченә ала. Информатика нигезләрен аңлау гына түгел, ә аларны педагогик эшчәнлектә ничек кулланырга икәнлеген күрсәтү дә безнең төп максатларның берсе булып тора. Шулай ук уку ярдәмлеге, милли-мәдәни контекстны исәпкә алып, белем алу процессын яңа дәрәҗәгә күтәрүгә юнәлтелгән.

Ярдәмлек берничә бүлектән тора, һәм һәр бүлек студентларның практик белемнәрен һәм аналитик күнекмәләрен үстерүгә юнәлтелгән. Беренче бүлектә информатика фәненең тарихы, аның иң күренекле шәхесләре һәм аларның дөньякүләм технологияләр үсешенә керткән өлеше тасвирлана. Алдагы бүлекләрдә исә Python һәм JavaScript программалаштыру телләре өйрәнелә. Python теленең универсальлеге, мәгълүматларны эшкәртү һәм анализлау өчен кулайлыгы күрсәтелә. JavaScriptның веб-технологияләр үсешендәге әһәмияте һәм аны укыту процессларында куллану алымнары аңлатыла.

Педагогик максатларда кулланырга мөмкин булган компьютер технологияләре, аларны өйрәнү өчен киңәшләр һәм күнегүләр белән тәэмин ителгән бүлекләр дә ярдәмлекнең әһәмиятле өлеше булып тора.

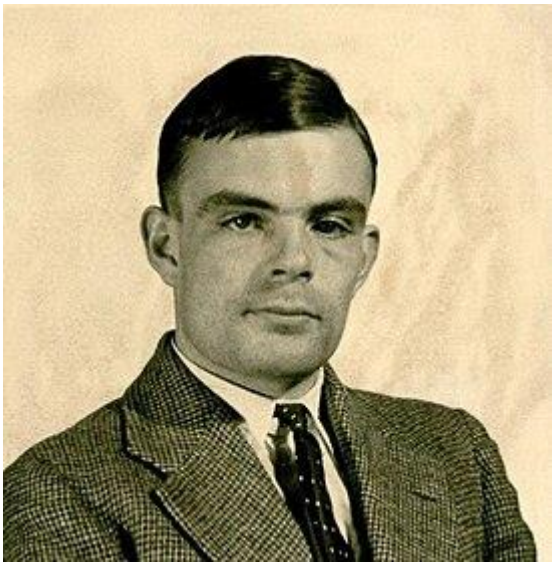
Әлеге уку ярдәмлеге, информатика фәне нигезләрен тирәнтен аңларга ярдәм итеп, милли үзенчәлекләрне саклаган мохиттә заманчалык һәм профессиональлекне берләштерә.

Авторлар бу ярдәмлекне төзергә булышкан Казан федераль унивеситетының билингваль һәм цифрлы белем алу кафедрасы укытучыларына, шулай уң анда укучы 10.2-424 группа студентларына рәхмәт белдерә.

Авторлар

Беренче бүлек.
Информатика тарихы һәм иң мөһим шәхесләр

1.1 Алан Тьюринг: информатика фәне һәм заманча дөньяның нигез ташы



Алан Тьюринг

Балачак һәм белем алуы

Алан Мэтисон Тьюринг 1912 елның 23 июнендә Лондонда урта катлам гаиләсендә туа. Аның әти-әнисе – Джулиус һәм Сара Тьюринглар – Һиндстанда Британия колониаль администрациясендә эшләгәннәр, әмма Аланны Бөекбританиядә тәрбияләү өчен кире Англиягә кайтканнар.

Алан кечкенәдән үк фәнгә зур кызыксыну күрсәткән. Ул үзлегеннән укып, фәнни эксперименталар ясаган, әмма башлангыч мәктәптә аңа авырлыклар да килгән: укытучылар аны артык оригиналь һәм игътибарсыз дип санаганнар. Шулай да, фәнгә тартылуы аның интеллектуаль үсешенә этәргеч биргән. Тьюринг Шерборн мәктәбендә укыганда, классик белем бирү системасына каршы торып, математик исәпләү һәм физика белән мавыккан.

Кембридж университетының Кингс Колледжында укыганда, ул үзен математиклыкта искиткеч сәләтле итеп күрсәтте. 1935 елда, 22 яшендә, ул үзенең беренче зур казанышын – үзара бәйле булган математика теоремаларын исбатлады һәм университет стипендиясе алды. Аның кызыксыну өлкәләре математик исәпләүләр, квант механикасы һәм физика белән чикләнмәде; ул, шулай ук, математик логикада да зур ачышлар ясарга әзер иде.

Тьюринг машинасы һәм исәпләүләр теориясе

1936 елда Тьюринг "Исәпләү машиналары һәм акыл" ("On Computable Numbers, with an Application to the Entscheidungsproblem") хезмәтен бастырды. Бу эшендә ул хәзерге заман компьютерлары өчен теоретик нигез булган **Тьюринг машинасы** концепциясен тәкъдим итте. Тьюринг машинасы алгоритмнарны исәпләүнең универсаль моделе буларак карала. Ул исбатлады: теләсә нинди исәпләү, аны башкаручы җайланма ничек кенә төзелгән булмасын, Тьюринг машинасы ярдәмендә симуляцияләнә ала.

Бу ачыш исәпләүләр теориясенә генә түгел, шулай ук ясалма интеллект һәм компьютер программалаштыру өлкәсенә дә зур йогынты ясады. Тьюрингның әлеге моделе хәзерге заман компьютерлары эшчәнлеген аңлау өчен нигез булып тора.

Икенче бөтендөнья сугышы һәм Энигманы вату

Икенче бөтендөнья сугышы вакытында Тьюринг Блетчли-Паркта, Британиянең шифрлау үзәгендә, эшләде. Ул немецларның шифрлау машиналарын, аеруча **Энигма**ны, ватуда белән шөгыльләнде. Тьюринг һәм аның хезмәттәшләре криптографик алгоритмнарны җиңү өчен махсус машиналар булдырдылар. Аларның иң мөһим эшләнмәләренең берсе **Бомбе** дип аталган җайланма булды, ул Энигма шифрларын ачу өчен кулланылды.

Тьюрингның криптография өлкәсендәге эшчәнлеге союздашларның сугышта җиңүенә зур өлеш кертте. Тарихчылар фикеренчә, Энигманы ватканга күрә, сугыш ике елга кыскарды, һәм миллионлаган кешенең гомере саклап калынды.

Сугыштан соңгы эшчәнлек

Сугыш тәмамланганнан соң, Тьюринг Манчестер университетында эшләде һәм беренче электрон санлы компьютерларны булдыруга зур өлеш кертә. Ул программалаштыруның теоретик һәм практик нигезләрен булдыру белән шөгыльләнде. Шул ук вакытта ул ясалма интеллект концепцияләрен үстерде.

1950 елда Тьюринг үзенең мәшһүр мәкаләсен бастырды: "Машина уйлый аламы?" ("Computing Machinery and Intelligence"). Бу хезмәтендә ул хәзер **Тьюринг тесты** дип аталган концепцияне тәкъдим итте. Бу тест ясалма интеллектның "акыллы" дип саналырлык дәрәҗәсен билгеләү өчен кулланыла. Тестның асылы шунда: әгәр дә компьютер кешене аралашу процессында үзенең машина икәненә ышандыра алмаса, ул акыллы дип санала.

Хөкем һәм фаҗигале язмыш

1952 елда Тьюринг шәхси тормышы өчен җәберләүләргә дучар булды. Аңа мәҗбүри химик кастрация узарга тәкъдим иттеләр, һәм бу аның психик һәм физик хәленә зур зыян китерде.

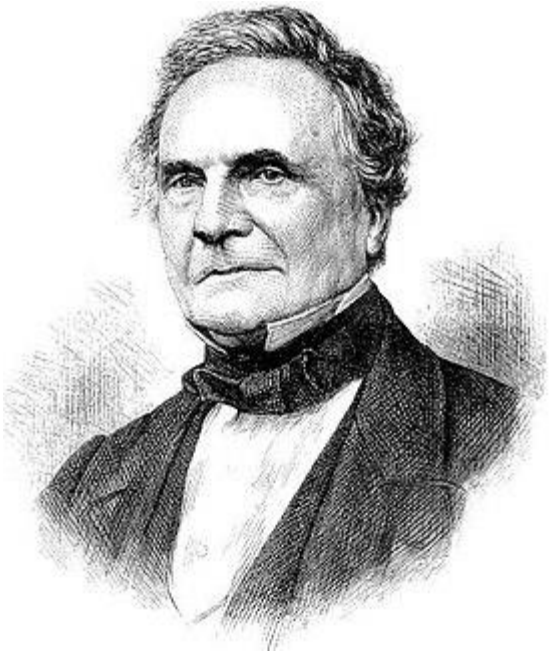
1954 елның 7 июнендә Алан Тьюринг вафат булды.

Мирас

Алан Тьюрингның хезмәтләре заманча цифрлы технологияләр үсешенә искиткеч йогынты ясады. Аның эшләре компьютерлар, алгоритмнар һәм ясалма интеллект үсешенең нигезен тәшкил итә.

Бөекбритания аны 2013 елда рәсми рәвештә реабилитацияләде, һәм 2019 елда аның портреты £50 банкнотада пәйда булды. Бүген Тьюрингның исеме компьютер фәненең нигез ташы буларак телгә алына, һәм аның хезмәтләре киләчәк буыннар өчен илһам чыганагы булып тора.

1.2. Чарльз Беббидж: заманча компьютерлар концепциясен нигезләүче



Чарльз Беббидж

Балачак һәм яшьлек еллары

Чарльз Беббидж 1791 елның 26 декабрендә Лондонда туган. Ул бай сәүдәгәр гаиләсендә тәрбияләнгән, һәм аның ата-анасы яшь Чарльзның белем алуына зур игътибар биргән. Балачактан ук ул математиканы һәм механиканы үз иткән. Математика өлкәсендәге аеруча сәләте аны Кембридж университетына алып килде, анда ул үзен дөньяга танылган галимнәр сафына кертте.

Кембриджда ул математик анализ белән шөгыльләнгән һәм күп кенә классик эшләнмәләргә тәнкыйть белән караган. Шул чорда ул фәндә яңа алымнар һәм концепцияләр кертүгә омтыла башлый.

Ижади эшчәнлеге

Беббиджның иң танылган хезмәтләренең берсе — “Аналитик машина” проекты. Ул беренче универсаль программалаштырыла торган механик компьютерның концепциясен тәкъдим итте. Аналитик машина башкарырга тиешле төп эшләр түбәндәге принципларга нигезләнде:

- Мәгълүматлар саклау: Ул механизмнарны хәтерләү һәм мәгълүматларны саклау өчен кулланырга теләде.
- Программа буенча эш: Машина математик исәпләүләрне башкару өчен карталар (перфокарталар) ярдәмендә программалаштырыла ала.
- Башкару җайланмасы: Машина алгоритмнарны үтәү һәм нәтиҗәләр чыгару өчен махсус берәмлекләрдән торырга тиеш.

Бу концепция хәзерге заман компьютерларының нигезен тәшкил итә.

“Дифференциаль машина”

Аналитик машинадан тыш, Беббидж Дифференциаль машина концепциясен дә тәкъдим итте. Бу машина математик таблицаларны автоматик рәвештә исәпләү өчен эшләнелгән. Ул җәяүле исәпләүләр белән бәйле хаталарны киметү максаты белән төзелгән.

Беббидж берничә прототип ясаган, ләкин заманының технологияләре аның идеяләрен тулысынча тормышка ашыру өчен җитәрлек булмаган. Шулай да, аның эшләнмәләре информатика тарихында төп казанышлар булып тора.

Ада Лавлейс белән хезмәттәшлек

Беббидж белән берлектә эшләгән һәм аның эшләрен аңлаган кешеләрнең берсе — Ада Лавлейс. Ул Аналитик машинаның программалаштыру мөмкинлекләрен аңлап, беренче программаларны булдырды. Лавлейс Беббидж идеяләренең практик әһәмиятен киңәйтеп, аларны теоретик нигез белән ныгытуга зур өлеш керткән.

Гыйльми мирас

Чарльз Беббиджның идеяләре үз заманында тиешле дәрәҗәдә бәяләнмәгән. Ул бик күп авырлыклар белән очрашты, шул исәптән финанс кыенлыклары һәм техник чикләүләр. Шулай да, аның эшләре математик исәпләүләр һәм мәгълүмат эшкәртү өлкәсендә киләчәк өчен юл күрсәтте.

Аның концепцияләре хәзерге заман инженерлары һәм фәнни җәмгыять тарафыннан югары бәяләнә. XX гасырда аның эшләре яңадан өйрәнелде, һәм Чарльз Беббидж “компьютерларның атасы” дип танылды.

Шәхси тормыш һәм соңгы еллары

Беббидж гомере буе үз идеяләрен тормышка ашырырга тырышкан. Ул озак еллар Лондонда яшәде, җәмгыять тормышында актив катнашты һәм инновацияләр тәкъдим итүдән туктамады. Аның шәхси тормышы исә күпмедер дәрәҗәдә фаҗигале: берничә баласын һәм тормыш иптәшен югалту аны тирән хәсрәткә салган.

1871 елның 18 октябрендә, 79 яшендә, Беббидж дөньядан китә. Аның эшләнмәләре, гәрчә ул үзе исән чагында тулысынча танылмаса да, бүгенге заман өчен бәяләп бетергесез мирас булып калды.

Мирас

Чарльз Беббиджның хезмәтләре компьютерлар һәм мәгълүмат эшкәртү системалары үсешендә мөһим роль уйнады. Аның идеяләре хәзерге заман инженерларына һәм галимнәренә илһам бирә.

Лондонның фәнни музейларында аның оригиналь машиналары һәм проектлары бүген дә күрсәтелә. Чарльз Беббидж — информатика һәм санлы технологияләр тарихында онытылмас фигура.

1.4 Ада Лавлейс: беренче программист һәм компьютер фәненең пионеры

Ада Лавлейс

Балачак һәм яшьлек

Августа Ада Кинг, графиня Лавлейс, 1815 елның 10 декабрендә Лондонда туган. Ул инглиз шагыйре Лорд Байрон һәм аның хатыны Аннабелла Милбэнкның кызы була. Байрон гаиләсе бәхетсез никах кичергән, һәм Ада яшь чакта ук әтисеннән аерыла. Әнисе, кызына атасының "хисләргә бирелүчәнлеге" күчмәсен өчен, аны фән, бигрәк тә математика һәм логика буенча тирән белем алырга мәҗбүр иткән.

Ада яшьтән үк гаять акыллы һәм үзенчәлекле шәхес булган. Ул математика өлкәсендә зур сәләт күрсәткән һәм күренекле математиклар белән аралашкан, шул исәптән танылган галим Мэри Сомервиль белән.

Беббидж белән хезмәттәшлек

Ада Лавлейсның тормышындагы иң мөһим вакыйга – аның Чарльз Беббидж белән танышуы. Беббиджның Аналитик машина концепциясе белән кызыксынып, ул аның эшләренә зур кызыксыну күрсәтә башлый. Лавлейс Беббиджның идеяләрен тирәнтен өйрәнеп, бу проектны гамәлгә ашырыр өчен үзенчәлекле фикерләр тәкъдим итә.

1843 елда ул итальян инженеры Луиджи Менабреаның Аналитик машина турында язган мәкаләсен инглизчәгә тәрҗемә итүне үз өстенә ала. Тәрҗемәгә ул үзенең киң аңлатмаларын һәм өстәмәләрен кертә, алар, чынлыкта, компьютер программалаштыруның беренче үрнәге булып тора. Бу аңлатмаларда ул, беренче тапкыр, компьютерның универсаль мәгълүмат эшкәртү җайланмасы була алачагын ассызыклый.

Беренче программа

Лавлейсның иң зур казанышы – Аналитик машина өчен беренче программаны булдыру. Ул машинада Бернулли саннарын исәпләү өчен алгоритм тәкъдим итә. Әлеге алгоритм хәзерге заман программалаштыруның нигезен тәшкил итә дип санала.

Лавлейс шулай ук компьютерларның математик исәпләүләрдән тыш, бүтән төрле мәгълүмат эшкәртү өчен кулланыла алуын да алдан күргән. Ул машиналарның музыка, графика һәм текст эшкәртү өчен дә мөмкинлекләре булачагын фаразлаган. Бу фикерләр замандашлары өчен революцион булып кабул ителгән.

Гыйльми карашлары

Ада Лавлейс "уйлап табылган машина"ның потенциалын тирән аңлаган һәм компьютерларның киләчәктәге үсеше турында искиткеч алдан күрү сәләтен күрсәткән. Ул аны бары тик исәпләү җайланмасы гына түгел, ә программалаштырыла торган универсаль механизм буларак кабул иткән.

Аның фәлсәфи карашлары, бигрәк тә "Аналитик машинаның мөмкинлекләре" хезмәтендә, фәнни һәм практик мәгънәдә информатика үсешенә юл күрсәткән. Ул мәгълүмат эшкәртүне механик төгәллек белән башкару концепциясен яклаган.

Шәхси тормыш

Ада Лавлейс, гомере буе искиткеч интеллектуаль потенциалга ия булса да, тормышта зур кыенлыклар кичергән. Ул 20 яшендә бай кеше булган Уильям Кингка кияүгә чыккан һәм өч бала тапкан. Әмма аның шәхси тормышы да, сәламәтлеге дә күп кенә авырлыклар белән тулы булган.

Лавлейс гомеренең ахыргы елларында рак белән авырган һәм 1852 елда, нибары 36 яшендә, вафат булган.

Мирас

Ада Лавлейсны хәзерге заман информатикасы "беренче программист" дип таный. Аның эшләре информатика һәм компьютер фәненең башлангыч нигезен салды. Ул беренче булып компьютерларның универсаль булу мөмкинлеген аңлаган һәм программа концепциясен тормышка ашырган.

1979 елда АКШ Оборона министрлыгы "Ada" исемле программалаштыру телен аның хөрмәтенә атады. Ул бүгенге көндә дә күпләр өчен илһам чыганагы булып тора.

Ада Лавлейс – үз заманының кыю фикер иясе, һәм аның фәнни карашлары заманча компьютер фәне үсешенә фундаменталь нигез салды.

1.5. Джон фон Нейман: компьютер архитектурасын нигезләүче һәм фәнни революционер

Джон фон Нейман

Балачак һәм яшьлек еллары

Джон фон Нейман (венгр телендә Янош Нейман) 1903 елның 28 декабрендә Венгриянең Будапешт шәһәрендә бай гаиләдә туа. Ул балачактан ук гаять сәләтле булып, үзен феноменаль хәтер һәм математик фикерләү белән күрсәтә. Фон Нейман 6 яшендә латинча һәм грекча укый белгән, ә 8 яшендә дифференциаль һәм интеграль исәпләүләрне өйрәнгән.

Ул Будапештгы иң яхшы мәктәпләрдә укый һәм 1921 елда Берлин университетына укырга керә. Соңрак, Цюрихта химия өлкәсендә докторантурага керсә дә, аның төп кызыксынуы һәрвакыт математика булып кала.

Математика һәм теоретик физика

1920-нче елларда фон Нейман математиканың төрле өлкәләрендә, шул исәптән функциональ анализда һәм квант механикасында зур уңышларга ирешә. Ул оператор теориясе буенча төп хезмәтләр авторы һәм квант механикасына Гильберт киңлекләре төшенчәсен кертүче булып санала.

1930 елда ул Принстон университетына эшкә күчә һәм тиздән Принстонның күренекле Фәннәр Институтында профессор була. Шул чорда аның эшчәнлеге күп тармаклы булып формалаша: ул математик логика, статистика, гидродинамика һәм уен теориясе белән шөгыльләнә.

Уен теориясе

1944 елда фон Нейман Оскар Моргенштерн белән берлектә "Уеннар теориясе һәм икътисади тәртип" хезмәтен бастыра. Бу хезмәттә ул уен теориясенең нигезләрен сала, ул бүгенге көндә икътисад, политология, биология һәм башка фән өлкәләрендә кулланыла.

Уен теориясендә фон Нейман төп төшенчәләрнең берсе – "Нейман эквилибриумы" (баланс хәлләре) – концепциясен тәкъдим итте. Бу теория катлаулы системалардагы стратегик үзара бәйләнешләрне аңлау өчен кулланыла.

Компьютер архитектурасы һәм исәпләүләр

Икенче бөтендөнья сугышы чорында һәм аннан соң фон Нейман компьютерлар архитектурасы өлкәсендә зур өлеш кертә. Ул "Фон Нейман архитектурасы" дип аталган концепцияне тәкъдим итә. Аның төп принциплары:

- Программа һәм мәгълүматлар бер үк хәтердә саклана.
- Компьютер эшчәнлеге этапларга бүленә: мәгълүматларны уку, эшкәртү һәм яздыру.
- Мәгълүматларны процессор белән эшкәртү берәмлегендә башкару.

Бу архитектура хәзерге заман компьютерларының нигезен тәшкил итә һәм әле дә киң кулланыла.

Хәрби эшчәнлек

Сугыш вакытында фон Нейман АКШның Лос-Аламос лабораториясендә атом-төш коралы үсешенә бәйле эшләрдә катнаша. Аның математик исәпләү һәм модельләштерү буенча күнекмәләре Манхэттен проекты өчен мөһим була. Ул шулай ук гидродинамика һәм шартлаткычлар теориясе өлкәсендә зур тәҗрибәгә ия.

Соңгы еллары һәм мирасы

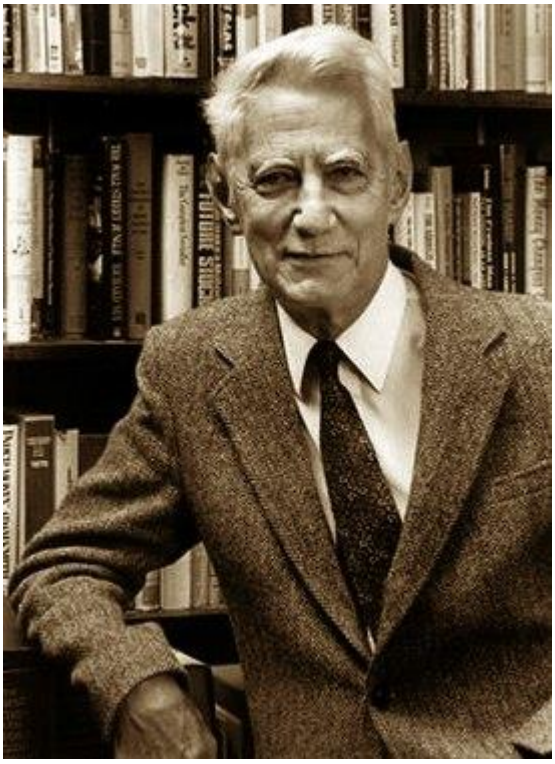
Фон Нейман, кыска гына гомер кичереп, 1957 елда 53 яшендә яман шештән вафат була. Шулай да, аның эшчәнлеге компьютер фәне, математика, физика һәм икътисад үсешенә йогынты ясау буенча тиңсез булып кала.

Аның хезмәтләре нәтиҗәсендә компьютерларның программалаштырыла торган структуралары барлыкка килде, алар фән һәм технология өлкәсендә үзгәрешләр китерде. Фон Нейманны хәзер заманча цифрлы дөньяның нигез ташын салучыларның берсе дип атыйлар.

Йомгак

Джон фон Нейман – күпкырлы шәхес. Аның интеллект һәм күзаллау киңлеге фәндә яңа юнәлешләр ачуга этәргеч булды. Ул заманча математик исәпләүләр һәм компьютерлар өчен фундаменталь нигезләр булдырды.

1.6. Клод Шеннон: мәгълүмат теориясенең атасы



Клод Шеннон

Балачак һәм яшьлек еллары

Клод Элвуд Шеннон 1916 елның 30 апрелендә АКШның Мичиган штатындагы Петоски шәһәрендә туа. Аның әтисе – судья, ә әнисе мәктәптә укытучы була. Шеннон яшьтән үк инженериягә һәм механикага зур кызыксыну күрсәтә. Ул үзенчәлекле уйлап табулар ясый, мәсәлән, чыбыклар ярдәмендә идарә ителә торган механик җайланмалар һәм радиоалгычлар төзи.

Шеннонның интеллект һәм инженерия өлкәсендәге талантлары аны Мичиган университетына алып килә, анда ул электротехника һәм математика буенча белем ала. Аның гыйльми эшчәнлеге инде шул чорда үзен күрсәтә: ул математик теоретикларның нигезләре белән кызыксына һәм инженерия мәсьәләләренә математик караш кертү юлларын эзли.

“Мәгълүмат теориясе” тууы

1930-нчы елларда Шеннон Массачусетс технология институтында (MIT) эшли башлый. Шул чорда ул логик алгебраны (булев алгебрасын) электр

схемаларын проектлау белән бәйләргә мөмкинлеген аңлый. 1938 елда Шеннон электр схемаларының математик логикага нигезләнүен аңлаткан мәкаләсен бастыра. Бу хезмәт “санлы схемалар” һәм санлы компьютерлар өчен фундаменталь нигез булып тора.

1948 елда Шеннон үзенең иң билгеле хезмәте – "Мәгълүмат теориясенең математик нигезләре" мәкаләсен бастыра. Бу хезмәтендә ул мәгълүматның математик концепциясен формалаштыра һәм аны үлчәү өчен төп төшенчә – бит (binary digit) – тәкъдим итә. Мәгълүмат теориясенең төп принциплары:

- Мәгълүматны тапшыру каналлары аша тапшыру өчен компрессия һәм кодлау алымнары.
- Мәгълүмат тапшыру системаларында хаталарны төзәтү механизмнары.
- Мәгълүматның “энтропия” төшенчәсе, ул мәгълүматның үз эчтәлеген үлчәү өчен кулланыла.

Клод Шеннон һәм цифрлы элемтә революциясе

Шенноның эшләре элемтә һәм санлы технологияләр үсешендә революция ясады. Ул сигнал тапшыруның эффективлыгын һәм төгәллеген күтәрү юлларын аңлатты. Бу эшләр санлы элемтә системалары – интернет, санлы телевизор, мобиль телефоннар һәм Wi-Fi технологияләре өчен нигез булып тора.

Шеннон шулай ук санлы кодлау системалары өчен төп концепцияләрне тәкъдим итте. Аның кодлау теоремалары санлы элемтә системаларында тапшыру тизлеге һәм сыйфат арасындагы бәйләнешне аңлата.

Шәхси тормыш һәм кызыксынулары

Шеннон фәннән тыш, төрле кызыклы шөгыльләре белән дә танылган. Ул механик җайланмалар ясау һәм шахмат уйнау белән мавыккан. Аның махсус эшләнмәләреннән берсе – шахмат уйнарга сәләтле робот, ул заманында зур кызыксыну уяткан. Шеннон шулай ук җәяүләп йөрүче бер көпчәкле велосипед (уницикл) йөртергә өйрәнгән.

Соңгы еллары һәм мирасы

Шенноның фәнни карьерасы дөньякүләм танылуга ирешә. Ул Массачусетс технология институтында һәм Белл лабораторияләрендә эшли. Аның хезмәтләре фән һәм инженерия өлкәсендә күп санлы премияләр белән бүләкләнгән, шул исәптән IEEE медале һәм Националь фән премиясе.

Клод Шеннон 2001 елның 24 февралендә вафат була. Ләкин аның эшчәнлеге мәңгелек әһәмияткә ия. Ул дөньяны цифрлы технологияләр революциясенә алып килгән һәм бүгенге заман элемтә системалары өчен нигез салган.

Йомгак

Клод Шеннон мәгълүмат теориясен формалаштырып, элемтә һәм санлы технологияләр үсешенә юл ачкан. Аның концепцияләре бүген дә актуаль һәм бөтен дөньяда кулланыла.

1.7. Грейс Хоппер: программалаштыру пионеры һәм “COBOL анасы”

Грейс Хоппер

Балачак һәм яшьлек

Грейс Мюррей Хоппер 1906 елның 9 декабрендә Нью-Йорк шәһәрендә туа. Ул яшьтән үк үзен белемгә омтылучы һәм аналитик фикерләүче итеп күрсәтә. Грейс физика, математика һәм инженериягә зур кызыксыну белән карый. Аның сәләтләре аеруча математика өлкәсендә ачыла, бу аны белем алуын дәвам итәргә этәрә.

Грейс математика буенча бакалавр дәрәҗәсен 1928 елда Вассар көллиятендә ала, аннары Йель университетында магистр һәм доктор дәрәҗәләренә ирешә. Ул Америкада доктор дәрәҗәсенә ирешкән аз санлы хатын-кыз галимнәрнең берсе була.

Хәрби хезмәт һәм фәнни эшчәнлек

Икенче бөтендөнья сугышы башлангач, Грейс АКШ Хәрби-диңгез флотының резерв көчләренә кушыла. Сугыш вакытында ул Гарвард университетында Mark I дип аталган беренче зур электрон компьютерда эшли башлый. Хоппер

әлеге машинаның программаларын булдыру һәм отладка үткәрү белән шөгыльләнә.

Mark I белән эшләү дәверендә ул программалаштыруның төп алымнарын булдыручы пионерларның берсе була. Бу тәжрибә аңа компьютерларны эш белән тәэмин итү системаларын нәтиҗәлерәк итү юлларын эзләргә этәрә.

Программалаштыруны революцияләү

Грейс Хопперның иң зур казанышларының берсе – компилятор булдыру. Ул программалаштыру телләрен кеше өчен аңлаешлы формага китереп, машинаның "аңлый" ала торган кодына әйләндерүче системаны эшли. Бу 1952 елда эшләнгән беренче компилятор компьютерлар белән эшләүне җиңеләйтте һәм программалаштыруда зур сикерешкә китерде.

Хоппер шулай ук COBOL (Common Business-Oriented Language) программалаштыру телен булдыруга зур өлеш керткән. COBOL киңкүләм бизнес һәм финанс операцияләре өчен төп тел булып формалашты. Аның концепциясе – программаларны кеше теленә мөмкин кадәр якынрак язу – ул заман өчен революцион идея иде.

“Баг” төшенчәсе

Грейс Хоппер белән бәйле кызыклы вакыйгаларның берсе – компьютер программасындагы хаталарны "баг" дип атау. 1947 елда ул Mark II машинасындагы бер ватылган бөҗәк аркасында эшләүдә хаталар барлыкка килүен күрсәтә һәм бу вакыйга "debugging" терминын популярлаштыра.

Соңгы еллары һәм мирас

Грейс Хоппер хәрби карьерасында вице-адмирал дәрәҗәсенә кадәр күтәрелә, бу Америка хәрби-диңгез флотында хатын-кыз өчен бик сирәк күренеш иде. Ул 1986 елда отставкага чыкканчы да, компьютер фәне һәм программалаштыруны үстерүгә зур өлеш кертә.

Хопперга күпсанлы бүләкләр һәм фәнни дәрәҗәләр бирелде, шул исәптән АКШ Президентының Азатлык медале (Presidential Medal of Freedom). Аның хөрмәтенә USS Hopper дип аталган хәрби-диңгез корабы һәм "Грейс Хоппер" фәнни конференциясе исемләнгән.

1992 елның 1 гыйнварында, 85 яшендә, Грейс Хоппер дөньядан китә. Ләкин аның фәнни мирасы һәм компьютер технологияләре үсешенә йогынтысы мәңгелек.

Йомгак

Грейс Хоппер – программалаштыруны гади һәм кулланучыларга уңайлы итү өчен нигез салган күренекле шәхес. Аның эшчәнлеге компьютер дөньясының үсешендә фундаменталь роль уйнады, һәм ул бүген дә компьютер фәнендә тарихи фигура булып санала.

1.8. Марвин Мински: ясалма интеллект фәненең пионеры

Марвин Мински

Балачак һәм яшьлек еллары

Марвин Мински 1927 елның 9 августында Нью-Йорк шәһәрендә туа. Аның әтисе күз табибы, ә әнисе еврей мәдәнияте һәм сәнгате белән кызыксынган шәхес була. Мински балачактан ук фән һәм технологиягә зур кызыксыну күрсәтә. Ул механик җайланмалар һәм роботлар ясый башлый, бу киләчәктәге эшчәнлеге өчен нигез сала.

Мински мәгариф алуда зур уңышка ирешә: ул Гарвард университетында белем ала, анда физика, математика һәм нейропсихология белән кызыксына. Соңрак Принстон университетында докторантурада укуын дәвам итә.

Ясалма интеллект һәм MIT лабораториясе

1956 елда Мински тарихка кереп калган Дартмут конференциясендә катнаша, анда “Ясалма интеллект” төшенчәсе рәсми рәвештә фәнни термин буларак тәкъдим ителә. Конференциядән соң ул ясалма интеллект фәне үсешенә җитди өлеш кертә башлый.

1959 елда Мински MIT университетында Ясалма интеллект лабораториясен нигезли. Бу лаборатория ясалма интеллект өлкәсендәге төп үзәкләрнең берсенә әйләнә һәм инновацион тикшеренүләр үткәрү урыны булып тора.

Гыйльми хезмәтләр һәм ачышлар

Марвин Мински ясалма интеллектның фундаменталь концепцияләрен үстерү белән шөгыльләнә. Аның төп казанышлары арасында:

- Нейрон челтәрләр: Мински нейрон челтәрләрнең исәпләү мөмкинлекләрен өйрәнгән һәм аларның теоретик нигезләрен үстергән. Ул аларның чикләрен дә билгеләгән, бу соңрак нейрон челтәрләр үсешенә йогынты ясаган.
- Рамка теориясе: Мински кеше акылының эшчәнлеген модельләштерү өчен “рамка” төшенчәсен тәкъдим итә. Бу теориягә күрә, кеше акылы мәгълүматны структуралы элементларга бүлеп эшкәртә. Әлеге теория ясалма интеллектка кеше фикерләүен имитацияләү мөмкинлеген бирә.
- Символик ИИ: Мински ясалма интеллектны символик структуралар ярдәмендә модельләштерү концепциясен эшләгән, бу системаларга логик фикер йөртү мөмкинлеген бирә.

Китаплары һәм фикерләре

Марвин Мински 1986 елда “Акыл җәмгыятьләре” (“The Society of Mind”) китабын бастыра. Бу китапта ул кеше акылының күпсанлы гади агентларның үзара хезмәттәшлегеннән барлыкка килгәнен фаразлый. Аның фикеренчә, ясалма интеллект та шундый күп агентлы структуралар ярдәмендә төзелергә тиеш.

Аның икенче китабы – “Emotion Machine” (“Хисләр машинасы”), анда Мински акыл һәм эмоцияләр арасындагы бәйләнешне тикшерә.

Робототехника һәм техника үсеше

Мински робототехникада да актив эшли. Ул беренче автоном роботларның берсен – “Арма” (ARM) – эшләгән. Шулай ук ул роботларның әйләнә-тирә мохит белән үзара бәйләнештә эшләү концепцияләрен өйрәнгән.

Соңгы еллары һәм мирас

Марвин Мински MIT университетында эшләвен һәм укытуын гомеренең ахырына кадәр дәвам итә. Ул ясалма интеллект үсешенә гаять зур йогынты ясаган, аның фәнни хезмәтләре бүген дә кулланыла һәм өйрәнелә.

Мински күпсанлы фәнни бүләкләр белән бүләкләнгән, шул исәптән Тьюринг премиясе – компьютер фәненең иң югары бүләге.

Мински 2016 елның 24 гыйнварында вафат була. Аның мирасы ясалма интеллект һәм робототехника өлкәсендәге үсеш өчен нигез булып тора.

Йомгак

Марвин Мински – ясалма интеллектның нигезен салучыларның берсе. Аның фәнни ачышлары һәм концепцияләре ясалма интеллект фәнен формалаштырган һәм бу юнәлештәге киләчәк тикшеренүләргә юл ачкан.

1.9. Дональд Кнут: алгоритмнар теориясе остасы һәм компьютер фәненең формалаштыручысы

Дональд Кнут

Балачак һәм яшьлек еллары

Дональд Эрвин Кнут 1938 елның 10 гаңварында Милуоки шәһәрендә (Висконсин, АКШ) туа. Ул кечкенәдән үк фән һәм музыка белән мавыга. Уку елларында Кнутның математик сәләте аеруча яхшы күренә: ул мәсьәләләрне гадәти юлдан түгел, ә креатив алымнар кулланып чишәргә тырыша.

Кнут Кейс-Вестерн резерв университетында белем ала, анда математика һәм физика буенча бакалавр дәрәҗәсен ала. Ләкин аны компьютерлар дөньясы һәм алгоритмнарның мөмкинлекләре аеруча җәлеп итә. Университетта ул беренче тапкыр программалаштыру белән таныша, һәм бу аның тормыш юлын билгели.

“Алгоритм сәнгате” хезмәте

Дональд Кнут компьютер фәненең нигез ташы булып торган хезмәтләрнең авторы булып санала. Аның иң билгеле эше – "Алгоритм сәнгате" ("The Art of Computer Programming") китаплар сериясе. Бу хезмәтләр компьютер фәне

буенча теоретик һәм практик белемнәрне системалаштыру өчен нигез булып тора.

"Алгоритм сәнгате" түбәндәге принципларга нигезләнә:

- Алгоритмнарның теоретик анализы һәм эффективлыкны бәяләү.
- Массивлар, мәгълүмат структуралары, эзләү һәм сортлау алгоритмнары кебек базис төшенчәләрнең тирән анализы.
- Компьютер фәненең математика белән тыгыз бәйләнеше.

Бу сериянең беренче томы 1968 елда дөнья күрә, һәм ул вакыттан бирле бу хезмәт компьютер фәне студентлары һәм белгечләре өчен классикага әйләнә.

Алгоритмнар теориясендә зур өлеш

Кнут алгоритмнарны оптимальләштерү һәм анализлау өчен төп төшенчәләрне тәкъдим итә. Ул "О-зурлык" (Big-O notation) системасын популярлаштыра, ул алгоритмның эш тизлеген һәм ресурс чыгымнарын бәяләү өчен кулланыла. Бу бүгенге көндә компьютер фәнендә стандартка әйләнгән.

“TEX” һәм типография системасы

1970-нче елларда Кнут компьютер фәне белән шөгыльләнеп кенә калмыйча, математик текстларны форматлау өчен үзенең программасын эшли башлый. Аның нәтиҗәсе – TEX (LaTeX платформасы өчен нигез) системасы.

TEX типография системасы:

- Математик һәм гыйльми текстларны форматлау өчен кулланыла.
- Текстларның төгәллеген һәм югары сыйфатын тәэмин итә.
- Академик басмалар һәм диссертацияләр өчен стандарт инструмент булып санала.

Компьютер фәненә карашы

Кнут компьютер программалаштыруны “сәнгать” дип атый. Ул фикер йөртү һәм чишелеш табу процессын математик төгәллек һәм иҗади караш белән берләштерергә тырыша. Аның эшләре фәнни төгәллекне һәм алгоритмнарның эстетик ягын берләштерү белән аерылып тора.

Бүләкләр һәм танылу

Дональд Кнутка күп санлы бүләкләр бирелгән, шул исәптән компьютер фәне өлкәсендәге иң югары дәрәҗә – Тьюринг премиясе (1974). Аны еш кына "алгоритмнарның Пикассосы" дип атыйлар, чөнки ул алгоритмнарны өйрәнү һәм куллануны сәнгать дәрәҗәсенә күтәргән.

Шәхси тормыш

Кнут музыка белән дә мавыккан һәм органда уйнаган. Аның шәхси тормышы гади һәм тәртипле булып кала, ул күбрәк гыйльми тикшеренүләргә һәм укыту эшчәнлегенә багышланган.

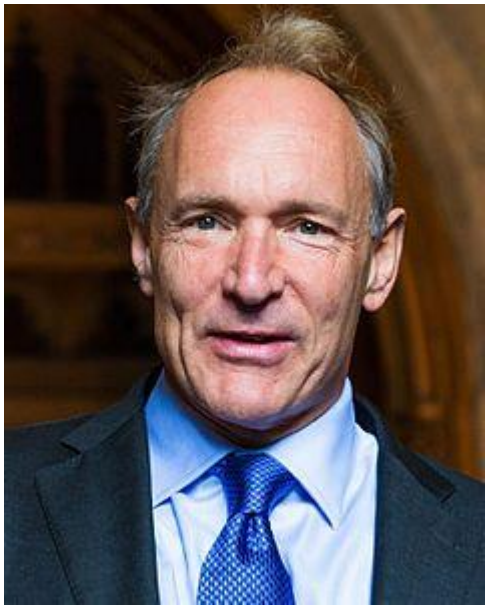
Мирас

Дональд Кнут компьютер фәнен фән буларак формалаштырган һәм аны системалаштырган. Аның "Алгоритм сәнгате" хезмәтләре буыннан-буынга укытыла һәм өйрәнелә. Кнутның алгоритмнарны өйрәнүгә керткән өлеше компьютер фәненең теоретик нигезен булдырган.

Йомгак

Дональд Кнут – компьютер фәнен үзенә күрә фән һәм сәнгать берләшмәсе итеп караган шәхес. Аның эшчәнлеге бу өлкәдәге башка галимнәр өчен маяк булып тора.

1.10. Тим Бернерс-Ли: Бөтендөнья пәрәвезен уйлап табучы



Тим Бернерс-Ли

Балачак һәм яшьлек еллары

Тимоти Джон Бернерс-Ли 1955 елның 8 июнендә Англиянең Лондон шәһәрендә туа. Аның әти-әнисе, Конвей Бернерс-Ли һәм Мэри Ли Вудс, математиклар һәм беренче коммерцияле компьютерларның берсе – Ferranti Mark 1 – эшләүчеләр арасында була. Тим яшьтән үк компьютерлар һәм технологияләр белән кызыксына. Ул Квинз Колледжда (Оксфорд университеты) физика укый һәм 1976 елда бакалавр дәрәҗәсен ала.

Оксфордта укыганда Тим беренче компьютеры өчен үзенчәлекле җайланмалар ясый. Аның инженер сәләте киләчәк эшчәнлегенең нигезен формалаштыра.

CERN һәм Бөтендөнья пәрәвезе идеясе

1980 елда Бернерс-Ли Европа атом тикшеренүләре үзәге (CERN) өчен эшли башлый. Ул монда беренче тапкыр гипертекст технологиясе белән кызыксына. Бу технология мәгълүматларны текстлы сылтамалар ярдәмендә бәйләргә мөмкинлек бирә. Ул CERN фәнни хезмәткәрләре арасында мәгълүмат алмашуны җиңеләйтү өчен гипертекстка нигезләнгән эчке система булдыра.

1989 елда ул бөтен дөньяны үзгәрткән идея белән чыга: мәгълүмат алмашу өчен глобаль гипертекст системасы булдыру. Ул әлеге системаны "World Wide Web" дип атый. Бу идея компьютерлар арасында мәгълүматны җиңеллек белән табу һәм тапшыру мөмкинлеге тудыра.

Бөтендөнья пәрәвезенең тормышка ашырылуы

1990 елда Тим Бернерс-Ли беренче веб-браузерны һәм серверны эшли. Бу проект нигезендә ул өч төп технология тәкъдим итә:

- HTML (HyperText Markup Language): веб-битләрне ясау өчен төп тел.
- HTTP (HyperText Transfer Protocol): мәгълүматны веб-сервердан кулланучыга тапшыру өчен протокол.
- URL (Uniform Resource Locator): интернеттагы ресурсларга сылтама бирү өчен уникаль адрес системасы.

Бу өч технология интернет үсешенең нигезен тәшкил итә. 1991 елда беренче веб-бит интернет челтәренә урнаштырыла, һәм Бөтендөнья пәрәвезе җәмәгатьчелеккә тәкъдим ителә.

WWW үсеше һәм Tim Berners-Leeның роле

Бернерс-Ли веб-технологияләрнең ачык һәм ирекле булуын тәэмин итүгә зур игътибар бирә. 1994 елда ул World Wide Web Consortium (W3C) оешмасын нигезли. W3C веб стандартларын үстерү һәм аларны гармонияле итү өчен җаваплы.

Бернерс-Ли шулай ук интернетның этик принципларын яклаучы буларак билгеле. Ул ирекле мәгълүматка керү, шәхси тормышны саклау һәм цензурасыз интернет өчен көрәшә.

Бүләкләр һәм танылу

Тим Бернерс-Ли үзенең хезмәтләре өчен дөнья күләмендә танылу ала. Аңа түбәндәге премияләр һәм титуллар бирелә:

- Тьюринг премиясе (2016): компьютер фәненең иң югары бүләге.
- Бөекбритания рыцарь титулы (2004): Королева Елизавета II тарафыннан бирелә.
- Милли технология медале (2007): АКШ президенты тарафыннан бүләкләнә.

2012 елда Олимпия уеннарының ачылыш тантанасында ул глобаль интернетның символы итеп тәкъдим ителә.

Соңгы еллары һәм мирас

Бүген Тим Бернерс-Ли интернетның ирекле һәм демократик булуы өчен актив эшли. Ул Web Foundation оешмасы аша цифрлы тигезлек һәм интернетка керү мөмкинлеген киңәйтү өчен кампанияләр оештыра.

Йомгак

Тим Бернерс-Ли – заманча дөньяның төп фигураларының берсе. Аның Бөтендөнья пәрәвезен булдыруы кешеләрнең аралашу, эшләү һәм белем алу юлларын үзгәртте. Аның мирасы интернет үсешенең һәм ирекле мәгълүмат алмашуның символы булып тора.

1.11. Линус Торвальдс: Linux операцион системасын булдыручы һәм ачык код хәрәкәте лидеры

Линус Торвальдс

Балачак һәм яшьлек еллары

Линус Бенедикт Торвальдс 1969 елның 28 декабрендә Финляндиянең Хельсинки шәһәрендә туа. Аның әти-әнисе журналистлар булып эшләгән, һәм алар, гадәттә, белемгә һәм ижади фикерләүгә зур әһәмият биргән гаиләдә тәрбияләгәннәр. Линус балачактан ук техник җайланмалар белән кызыксына башлый: ул беренче компьютерын – Commodore VIC-20 – 11 яшендә ала, һәм шул вакыттан бирле программалаштыру белән шөгыльләнә.

Линус Хельсинки университетында компьютер фәне укый. Университет елларында ул операцион системалар архитектурасы белән тирәнтен кызыксына һәм бу өлкәдә мөһим тикшеренүләр башлый.

Linuxның тууы

1991 елда, 21 яшендә, Линус MINIX операцион системасын өйрәнә башлый, ул укыту максатында эшләнгән, ләкин чикләнгән функциональлеккә ия була. Торвальдс MINIXның чикләүләренә канәгать булмыйча, үзенең шәхси операцион системасын язарга карар кыла.

1991 елның 25 августында Линус интернетка мәгълүмат урнаштыра, анда ул яңа операцион системаның башлангыч версиясе турында хәбәр итә һәм аны ирекле куллану өчен тәкъдим итә. Ул аны "Linux" дип атый (Линус исеменә һәм UNIX системасына ишарә ясап).

Linux тиз арада хакерлар, программалаштыручылар һәм компьютер энтузиастлары арасында популярлык казана. Торвальдс үзенең системасын ачык кодлы итәргә карар кыла, бу аңа глобаль хәрәкәтнең өлешенә әйләнергә ярдәм итә.

Ачык код хәрәкәте һәм GPL

Linuxның үсеше GPL (General Public License) лицензиясе кысаларында тормышка ашырыла. Бу лицензия программаларны ирекле таратырга һәм үзгәрешләр кертергә мөмкинлек бирә. Linux дөньяның төрле почмакларыннан меңләгән программистларның бергәләп эшләве нәтиҗәсендә үсә.

Ачык код хәрәкәте концепциясе Торвальдсның төп идеологиясе булып кала. Аның фикеренчә, программаларның ачык булуы инновацияләрне тизләтә һәм кулланучылар өчен яхшырак продуктлар булдыра.

Linuxның глобаль үсеше

Linux хәзерге көндә серверлар, суперкомпьютерлар, мобиль җайланмалар һәм күпсанлы башка системалар өчен төп операцион система булып тора. Мисаллар:

- Android: Android операцион системасы Linux ядросына нигезләнгән, бу аны миллиардлаган җайланмада куллануга китерә.
- Серверлар: Бүгенге көндә интернет серверларының күпчелеге Linux белән идарә ителә.
- Суперкомпьютерлар: Дөньяның иң куәтле суперкомпьютерлары Linux нигезендә эшли.

Шәхси тормыш һәм характер

Линус Торвальдс үзен гади һәм практик фикер йөртүче кеше итеп күрсәтә. Ул беркайчан да байлыкка һәм танылуга омтылмаган, ә программалаштыруны үзенең яраткан шөгыле дип санаган. Шулай ук, ул гадәттән тыш туры сүзле һәм кайвакыт үз фикерләрен кырыс формада белдерүе белән билгеле.

Бүләкләр һәм танылу

Линус Торвальдс күпсанлы бүләкләр белән бүләкләнгән, шул исәптән:

- Милли технология медале (2014): АКШ президенты тарафыннан бирелгән.
- Технология, күңел һәм инновацияләр буенча Миллениум премиясе (2012): Аның глобаль ачык код хәрәкәтенә керткән өлеше өчен.
- Тьюринг премиясе (2018): Компьютер фәнендәге иң югары бүләк, Linux ядросын булдырганы һәм ачык код фәлсәфәсен алга этәргәне өчен бирелгән.

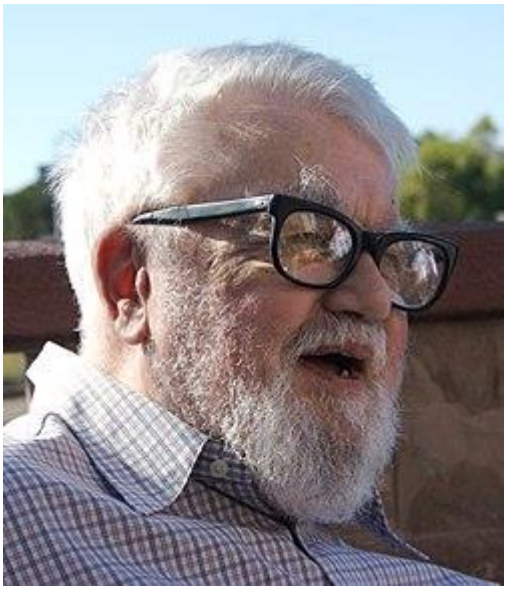
Мирас

Линус Торвальдсның Linux операцион системасы һәм ачык код хәрәкәте дөньяны үзгәртте. Аның эшләре нәтиҗәсендә технологияләр дөньясы тагын да ачык, инклюзив һәм ижади булып формалашты. Бүгенге көндә Linux һәм ачык код хәрәкәте глобаль технологияләр үсешенең төп нигезе булып тора.

Йомгак

Линус Торвальдс заманча технологияләр үсешенә үзенчәлекле өлеш кертте. Аның Linux операцион системасы һәм ачык код идеологиясе дөньяда миллионлаган программист һәм кулланучыга илһам бирде.

1.12. Джон Маккарти: ясалма интеллектның атасы һәм Lisp телен булдыручы



Джон Маккарти

Балачак һәм яшьлек еллары

Джон Маккарти 1927 елның 4 сентябрендә Массачусетс штатындагы Бостон шәһәрендә туа. Аның әти-әнисе, Джон Патрик һәм Ида Глэйзер, Ирландия һәм Литвадан Америкага күченгән эмигрантлар булган. Маккартиның әтисе эшче хәрәкәттә актив катнашкан, бу яшь Джонга социаль фикерләрнең һәм белемнең әһәмиятен аңларга ярдәм итә.

Балачак елларында ук ул үзен гениаль сәләтле бала итеп күрсәтә: мәктәптә аның математик мәсьәләләрне тиз һәм нәтиҗәле чишүе укытучыларны шаккаттыра. Маккарти математиканы һәм исәпләүләрне тирәнтен өйрәнү өчен үзлегеннән укый башлый. Соңрак ул Калифорния Технология Институтына (Caltech) укырга керә һәм 1948 елда математика буенча бакалавр дәрәҗәсен ала.

Ясалма интеллектка нигез салу

Джон Маккартиның төп казанышларының берсе – ясалма интеллект (ИИ) төшенчәсен формалаштыру. 1956 елда ул ясалма интеллект фәнен үстерүгә багышланган беренче конференцияне – Дартмут семинарын – оештыра. Бу семинарда “Artificial Intelligence” термины беренче тапкыр рәсми кулланыла. Маккарти ясалма интеллектны болай билгели:

“Машиналар кешегә хас булган акыл сәләтләрен имитацияли ала торган фән һәм технология.”

Бу билгеләмә ясалма интеллект үсешенең теоретик һәм практик нигезләрен билгели.

Lisp телен булдыру

1958 елда Маккарти программалаштыру телләренең иң әһәмиятле берсен – Lisp – булдыра. Lisp теленең төп үзенчәлекләре:

- Функциональ программалаштыру концепцияләрен куллану.
- Символик исәпләүләр өчен уңайлы булу.
- Рекурсия һәм динамик типлаштыруны куллану мөмкинлеге.

Lisp ясалма интеллект өлкәсендә төп тел булып формалашты һәм әле дә киң кулланыла. Аның төзелеше инновацион һәм ул программалаштыруның күп алымнарына нигез булып тора.

Вакыт бүлешү системалары

Маккарти шулай ук вакыт бүлешү системалары концепциясен эшләүгә зур өлеш керткән. Ул компьютерларны берьюлы берничә кулланучыга хезмәт күрсәтергә өйрәтүче техникаларны үстергән. Бу идея бүгенге көндәге күп кулланучылы операцион системалар һәм серверлар нигезендә ята.

Гыйльми һәм педагогик эшчәнлек

Маккарти Стэнфорд университетында озак еллар эшли. Ул анда ясалма интеллект лабораториясен җитәкли һәм күпсанлы тикшеренүләр үткәрә. Аның студентлары арасында бүгенге көндә ясалма интеллект өлкәсендә билгеле галимнәр күп.

Маккарти ясалма интеллектның мораль һәм этик аспектларын да өйрәнә. Ул роботларның һәм интеллектуаль системаларның кешелеккә ничек хезмәт итәргә тиеш булуын фаразлый.

Бүләкләр һәм танылу

Джон Маккарти ясалма интеллект өлкәсендә төп фигура буларак күпсанлы бүләкләр белән бүләкләнә, шул исәптән:

- Тьюринг премиясе (1971): ясалма интеллект фәненә керткән фундаменталь өлеше өчен.
- Националь фән премиясе (1990): АКШ президенты тарафыннан тапшырылган.
- Киото премиясе (1988): фән һәм технология үсешенә зур өлеше өчен.

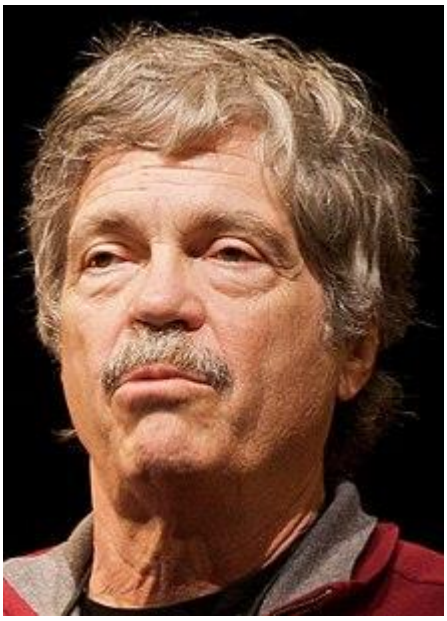
Соңгы еллары һәм мирас

Джон Маккарти 2011 елның 24 октябрендә вафат була. Ләкин аның эшләре ясалма интеллект һәм компьютер фәне үсешендә мәңгелек эз калдырды. Ул ясалма интеллектны фән буларак нигезләгән һәм аны киләчәккә алып бару юнәлешен билгеләгән шәхес.

Йомгак

Джон Маккарти – ясалма интеллектның нигез ташын салучы һәм фәннең чикләрен киңәйтүче күренекле фигура. Аның фәлсәфи карашлары, Lisp телен булдыруы һәм ИИ өлкәсенә керткән өлеше күп буын галимнәр өчен илһам чыганагы булып тора.

1.13. Алан Кэй: объект-юнәлешле программалаштыру һәм персональ компьютер концепциясенең пионеры



Алан Кэй

Балачак һәм яшьлек еллары

Алан Кэй 1940 елның 17 мае көнне Спрингфилд шәһәрендә (Миссури, АКШ) туа. Ул балачактан ук сәләтле һәм креатив фикер йөртүче буларак билгеле. Кэй яшь чакта музыка белән кызыксына, ләкин техник җайланмалар һәм фәнни ачышлар аны үзенә ныграк тарта.

Калифорния университетында биология укып чыкканнан соң, ул АКШ Хәрби-һава көчләрендә эшли. Шул вакытта ул беренче тапкыр компьютерлар белән таныша һәм бу юнәлештә зур сәләт күрсәтә. Компьютерлар белән кызыксынуы аны компьютер фәненә алып килә.

Объект-юнәлешле программалаштыру һәм Smalltalk

1970-нче елларда Алан Кэй Xerox PARC (Palo Alto Research Center) тикшеренү үзәгендә эшли. Биредә ул программалаштыруда төп революцияләрнең берсен башлап җибәрә – объект-юнәлешле программалаштыру концепциясен формалаштыра. Бу концепциянең асылы шунда: программалар мөстәкыйль объектлардан төзелә, алар үз мәгълүматларын саклый һәм башка объектлар белән үзара бәйләнештә була ала.

Алан Кэй һәм аның командасы Smalltalk программалаштыру телен булдыра. Smalltalk үзенең объект-юнәлешле архитектурасы белән программалаштыруның киләчәк юнәлешен билгели. Ул бүгенге көндә киң кулланыла торган Java, Python һәм C++ кебек телләргә зур йогынты ясый.

Персональ компьютер концепциясе

Кэй персональ компьютер идеясенең башында торган. Ул компьютерлар һәр кешегә караган инструмент булырга тиеш дип исәпли. 1968 елда ул Dynabook концепциясен тәкъдим итә – бу концепция бүгенге планшетлар һәм ноутбукларның прототибы буларак санала. Кэй фаразлавынча, персональ компьютер белем бирү, иҗат һәм коммуникация өчен универсаль инструмент булырга тиеш.

График интерфейс үсешенә йогынты

Xerox PARC үзәгендә Алан Кэй график кулланучылар интерфейсын (GUI) үстерүдә катнаша. Ул тәзергә нигезләнгән интерфейсларны яклый һәм бүгенге көндәге график интерфейслы системалар өчен фундаменталь нигез сала. Бу концепция соңрак Apple һәм Microsoft тарафыннан кулланыла һәм киң тарала.

Белем бирүдә технологияләр куллану

Алан Кэй технологияләрнең белем бирүдәге әһәмиятен таный һәм аларны уку-укыту процессында куллану яклы була. Ул яшь буынны компьютерлар белән таныштыру өчен махсус программа тәэминаты һәм җайланмалар булдыруга үз өлешен кертә.

Бүләкләр һәм танылу

Алан Кэй күпсанлы бүләкләргә ия, шул исәптән:

- Тьюринг премиясе (2003): компьютер фәне өлкәсендәге иң югары бүләк, объект-юнәлешле программалаштыру һәм персональ компьютер концепцияләренә керткән өлеше өчен.
- IEEE Джон фон Нейман медале (2004): информатика өлкәсенә яңалыклар кертү өчен.
- Japan Prize (2018): мәгълүмат технологияләре үсешенә фундаменталь өлеш керткәне өчен.

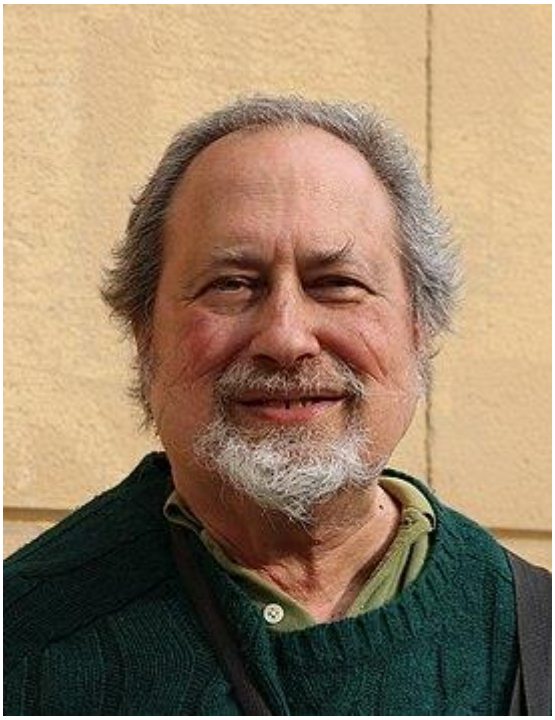
Соңгы еллары һәм мирас

Кэй бүген дә компьютер фәне һәм белем бирү технологияләрен үстерүдә актив эшли. Аның идеяләре һәм хезмәтләре заманча технологияләрнең күпчелек аспектларын формалаштырган.

Йомгак

Алан Кэй – объект-юнәлешле программалаштыру һәм персональ компьютер концепциясен тудыручы төп фигура. Аның Dynabook идеясе, Smalltalk теле һәм график интерфейс үсешенә керткән өлеше заманча цифрлы дөньяның нигезе булып тора. Аның хезмәтләре һәм концепцияләре бүгенге көндә дә актуаль һәм киләчәк өчен илһам чыганагы булып кала.

1.14. Ричард Столлман: ирекле программалар хәрәкәте лидеры һәм GNU проектын нигезләүче



Ричард Столлман

Балачак һәм яшьлек еллары

Ричард Мэтью Столлман 1953 елның 16 марты көнне Нью-Йорк шәһәрендә туа. Балачактан ук ул искиткеч аналитик сәләте һәм белемгә омтылуы белән аерылып тора. Мәктәптә физика һәм математика белән тирәнтен кызыксынган Столлман, үсмер чакта ук программалаштыру белән таныша.

Гарвард университетында белем алганда, ул физика буенча бакалавр дәрәҗәсен ала, шул ук вакытта Массачусетс технология институтының (MIT)

лабораторияләрендә эшли башлый. Биредә ул ирекле фикер йөртүче һәм сәләтле программист булып таныла.

GNU проекты һәм ирекле программалар хәрәкәте

1983 елда Столлман GNU проектын башлап җибәрә. Аның төп максаты – ирекле операцион система булдырып, кулланучыларга программаларны ирекле куллану, өйрәнү, үзгәртү һәм тарату мөмкинлеге бирү. GNU операцион системасы концепциясе Unix операцион системасы нигезендә эшләнгән, ләкин ирекле лицензия шартларына нигезләнгән була.

GNU проекты кысаларында Ричард Столлман күп санлы инструментлар һәм программалар эшли, шул исәптән:

- GNU Compiler Collection (GCC): программалаштыру өчен универсаль компиляторлар җыелмасы.
- GNU Emacs: көчле һәм үзләштереп була торган текст мөхәррире.
- GNU Debugger (GDB): программаларны отладка үткәрү өчен корал.

GNU проектының төп өлеше – ирекле программалар төшенчәсен алга сөрү һәм кулланучыларга үз программаларын куллану һәм үзгәртү хокукларын кайтару.

Ирекле программалар фәлсәфәсе

1985 елда Столлман Free Software Foundation (FSF) оешмасын нигезли. Бу оешма ирекле программаларны пропагандалау һәм үстерү белән шөгыльләнә. Аның төп фәлсәфәсе дүрт ирекkә нигезләнә:

1. Программаны теләсә нинди максатта куллану иреге.
2. Программаны өйрәнү һәм үзгәртү иреге.
3. Программаны башка кешеләргә тарату иреге.
4. Үзгәртелгән версияләрне тарату иреге.

Бу фәлсәфә программаларны коммерция кысаларыннан чыгарып, иҗтимагый ресурс итеп кабул итүгә юнәлтелгән.

GPL лицензиясе

Ричард Столлман 1989 елда General Public License (GPL) лицензиясен эшли. Бу лицензия ирекле программаларны яклау һәм аларны коммерциягә яраклаштырудан саклау өчен махсус төзелгән. GPL бүгенге көндә иң киң таралган лицензияләрнең берсе булып тора һәм ачык кодлы программалар үсешен тәэмин итә.

Ачык код һәм ирекле программалар арасында каршылык

Столлманның фәлсәфәсе ачык код хәрәкәте белән якын булса да, ул бу төшенчәне кулланудан саклана. Аның фикеренчә, ачык кодлы программалар техник өстенлекләр турында күбрәк, ә ирекле программалар фәлсәфи һәм этик нигезләр турында сөйли. Ул ирекле программаларны кулланучыларның хокукларын яклау инструменты дип кабул итә.

Танылу һәм мирас

Ричард Столлманның хезмәтләре дөньякүләм танылу ала. Ул күпсанлы бүләкләр һәм премияләр белән бүләкләнә:

- ACM Grace Murray Hopper Award (1990): Emacs һәм башка инструментлар өчен.
- Электрон чикләр фонды (EFF) пионер премиясе (1998): интернет иреге һәм технологияләрне алга җибәрү өчен.
- Макартур премиясе: фәнни һәм иҗади сәләте өчен.

Соңгы еллары

Бүгенге көндә Ричард Столлман Free Software Foundation эшчәнлегендә актив катнаша. Ул ирекле программаларның җәмгыять өчен әһәмиятен пропагандалый һәм бу юнәлештә эшләүче яшь буын программистлар өчен юл күрсәтә.

Йомгак

Ричард Столлман ирекле программалар хәрәкәтенең символына әйләнде. Аның хезмәтләре һәм фәлсәфәсе программалаштыруның социаль һәм этик аспектларын яңача күзалларга мөмкинлек бирде. Ул программаларның бары тик техник продукт кына түгел, ә җәмгыять өчен мөһим ресурс булуын ассызыклый.

1.15. Винтон Серф һәм Роберт Кан: Интернетның аталары

Винтон Серф (сул якта) һәм Роберт Кан (уң якта)

Балачак һәм яшьлек еллары

Винтон Серф 1943 елның 23 июнендә Калифорния штатының Нью-Хейвен шәһәрендә туа. Яшьтән үк фән һәм технология белән кызыксына, бигрәк тә физика һәм математика өлкәсендә зур сәләтләр күрсәтә. Серф Стэнфорд университетында математика буенча бакалавр дәрәҗәсен ала, ә аннары Калифорния университетында (UCLA) компьютер фәне буенча магистр һәм доктор дәрәҗәләренә ирешә.

Роберт Кан 1938 елның 23 декабрендә Нью-Йорк шәһәрендә туа. Ул электротехника һәм коммуникацияләр белән кызыксына. Кханның инженериягә булган омтылышы аны Принстон университетында электротехника буенча белем алырга этәрә. Соңрак ул ARPA (Advanced Research Projects Agency) агентлыгында эшли башлый, бу аның карьерасында мөһим роль уйный.

TCP/IP протоколлары

1970-нче елларда Серф һәм Кан АКШ хөкүмәтенең ARPANET проектында эшли. ARPANET – интернетның башлангыч версиясе. Бу проектның төп максаты – берничә компьютер челтәрен бергә бәйләү һәм мәгълүматны эффектив һәм ышанычлы тапшыру системасы булдыру.

1974 елда Винтон Серф һәм Роберт Кан TCP/IP (Transmission Control Protocol/Internet Protocol) протоколын тәкъдим итә. Бу технология интернетның эш принциплары өчен нигез булды:

- TCP: Мәгълүматны пакетка бүлү һәм аларны адрес буенча тапшыру.
- IP: Мәгълүматның билгеләнгән адреска килеп җитүен тәэмин итү.

TCP/IP интернетның төп архитектурасы булып формалашты һәм хәзер дә кулланыла.

Интернетның үсеше

Серф һәм Каннның эшчәнлеге интернетны киң тарату өчен шартлар тудырды. 1980-нче елларда TCP/IP ARPANETның төп протоколы буларак кабул ителә, һәм бу барлык компьютер челтәрләрен бердәм глобаль челтәргә берләштерергә мөмкинлек бирә.

Серф һәм Кан интернет үсешенең техник нигезләрен генә түгел, шулай ук халыкара стандартлар системасын да булдыруга ярдәм итте. Алар интернетны коммерция һәм җәмәгатьчелек өчен ачык ресурс итеп таратуга омтылды.

Бүләкләр һәм танылу

Винтон Серф һәм Роберт Кан үзләренең хезмәтләре өчен күпсанлы премияләр һәм танылу алды:

- Тьюринг премиясе (2004): Интернет архитектурасына һәм TCP/IP протоколларына керткән өлеше өчен.
- Президентының Азатлык медале (2005): АКШның иң югары гражданлык бүләге.
- Япония премиясе (1997): Фән һәм технология өлкәсендәге инновацияләр өчен.
- IEEE Alexander Graham Bell медале (1992): телекоммуникацияләргә керткән өлеше өчен.

Соңгы еллары һәм эшчәнлек

Винтон Серф хәзерге вакытта Google компаниясендә “Интернет-елчысы” булып эшли. Ул интернет үсешенә багышланган проектларда катнаша һәм интернетка керү мөмкинлеген киңәйтү белән шөгыльләнә.

Роберт Кан CNRI (Corporation for National Research Initiatives) җитәкчесе буларак интернет технологияләрен алга җибәрүгә юнәлтелгән тикшеренүләр үткәрә.

Мирас

Серф һәм Кан интернетның нигезен формалаштырып, дөньякүләм коммуникацияләр революциясенә юл ачты. Аларның TCP/IP протоколлары интернетны теләсә кайсы кулланучы өчен уңайлы һәм эффектив итүгә мөмкинлек бирде. Бүгенге интернет инфраструктурасы аларның гениаль эшләнмәләренә нигезләнгән.

Йомгак

Винтон Серф һәм Роберт Кан – интернетны тормышыбызның аерылгысыз өлешенә әверелдергән шәхесләр. Аларның хезмәтләре глобаль мәгълүмат челтәрләрен үстерү өчен нигез булып тора һәм киләчәк буын өчен этәргеч булып хезмәт итә.

1.16. Барбара Лисков: объект-юнәлешле программалаштыру һәм мәгълүмат абстракциясе пионеры

Барбара Лисков

Балачак һәм яшьлек еллары

Барбара Лисков 1939 елның 7 ноябрендә Калифорниянең Лос-Анджелес шәһәрендә туа. Аның әти-әнисе белемгә зур әһәмият биргән гаиләдә үскән. Ул балачактан ук математика һәм аналитик фикерләүдә сәләт күрсәтә. Лисков Калифорния университетында (Berkeley) математика буенча бакалавр дәрәҗәсен ала.

Бераз эшләгәннән соң, Лисков Массачусетс технология институтына (MIT) укырга керә. Ул 1968 елда MITта компьютер фәне буенча докторлык

диссертациясен яклый. Барбара Лисков MIT тарихында компьютер фәне буенча доктор дәрәҗәсен алган беренче хатын-кыз булып санала.

Мәгълүмат абстракциясе һәм программалаштыру

Барбара Лисковның төп өлеше – мәгълүмат абстракциясе концепциясен эшләү. Мәгълүмат абстракциясе объект-юнәлешле программалаштыруның һәм модульле системалар эшләүнең нигез ташы булып тора. Бу концепция программистларга катлаулы системаларны гади, идарә ителә торган компонентларга бүлеп эшләргә мөмкинлек бирә.

Лисков үз эшләрендә мәгълүмат абстракциясе ярдәмендә программаларның үзгәрешләргә чыдамлы булуын һәм кабат куллану мөмкинлеген арттыру юлларын тикшерә. Бу идеяләр бүгенге көндә программалаштыруда стандарт булып санала.

Лисков алмашу принцибы (Liskov Substitution Principle)

Лисковның исеме белән бәйле булган иң мөһим концепцияләрнең берсе – Лисков алмашу принцибы (LSP). Бу принцип объект-юнәлешле программалаштыруда мөһим роль уйный һәм түбәндәгечә формулировкалана:

"Әгәр дә S объекты T объектының туындысы булса, T объекты кулланылырга тиеш булган урында S объекты да кулланылырга тиеш."

Бу принцип кодның модульлелеген һәм көйләнүчәнлеген тәэмин итү өчен нигез булып тора. Лисков алмашу принцибы объект-юнәлешле телләрдәге класслар һәм мирас алулар белән эшләгәндә киң кулланыла.

CLU программалаштыру теле

Барбара Лисков CLU программалаштыру телен эшләү командасын җитәкли. CLU – объект-юнәлешле программалаштыру идеяләрен кулланучы беренче телләрнең берсе. Бу телдә мәгълүмат абстракциясе, класслар, югары дәрәҗәле мәгънәле мәгълүмат структуралары кебек төшенчәләр кертелгән. CLU телендәге күпчелек концепцияләр хәзерге заман телләрендә, мәсәлән, Java һәм Python, киң кулланыла.

Таркатылган системалар

Лисков шулай ук таркатылган системаларны тикшерүгә зур өлеш керткән. Ул таркатылган системалардагы ышанычлылык һәм хаталарга чыдамлылык мәсьәләләрен тикшергән. Аның тикшеренүләре заманча таркатылган системаларның тотрыклылыгын һәм эффективлыгын арттыру өчен кулланыла.

Бүләкләр һәм танылу

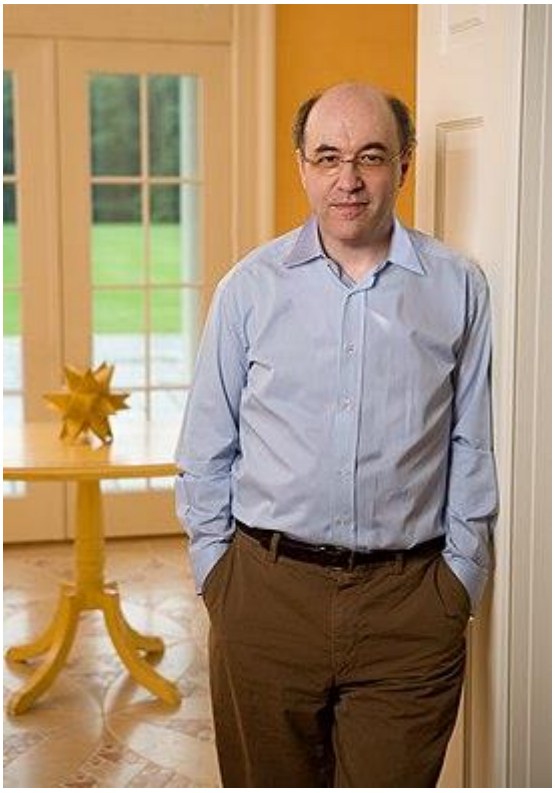
Барбара Лисков хезмәтләре дөньякүләм танылу ала һәм ул күпсанлы бүләкләр белән бүләкләнә:

- Тьюринг премиясе (2008): объект-юнәлешле программалаштыру һәм мәгълүмат абстракциясе өлкәсенә керткән өлеше өчен.
- IEEE Джон фон Нейман медале: компьютер фәне үсешенә керткән казанышлары өчен.
- АКШ Милли Инженерия Академиясенә сайлану: инженерлык өлкәсендәге җитди өлеше өчен.

Йомгак

Барбара Лисков – компьютер фәне үсешенә зур өлеш керткән күренекле шәхес. Аның мәгълүмат абстракциясе, CLU теле, һәм Лисков алмашу принцибы программалаштыру концепцияләренең нигез ташлары булып тора. Аның эшләре заманча программалаштыруның күп аспектларын билгеләгән һәм үстергән.

1.17. Стивен Вольфрам: фәнни исәпләүләр революционеры һәм клетчатлы автоматлар теоретигы



Стивен Вольфрам

Балачак һәм яшьлек еллары

Стивен Вольфрам 1959 елның 29 августында Лондонда туа. Аның әтисе – Ричард Вольфрам – фәлсәфәче, ә әнисе – Сибил Вольфрам – оксфорд профессоры. Балачактан ук Стивен үзен гениаль бала итеп күрсәтә: 15 яшендә инде фәнни мәкаләләр яза башлый. Мәктәптә укыганда ук, ул физика һәм математика белән тирәнтен кызыксына.

Ул Оксфорд университетына укырга керә, ләкин аннан АКШның Калифорния технология институтына (Caltech) күчә. Биредә ул физика өлкәсендә доктор дәрәҗәсен ала. Аның тикшеренүләре физика, математика һәм компьютер фәне өлкәләрендә киң колачлы була.

Клетчатлы автоматлар һәм "Яңа төр фән"

1980-нче елларда Вольфрам фәндә революцион теманы – клетчатлы автоматлар (cellular automata) – өйрәнә башлый. Бу төшенчә табигый системаларны гади математик кагыйдәләр ярдәмендә модельләштерү мөмкинлеген күрсәтә. Вольфрам фикеренчә, гади алгоритмнардан катлаулы системалар тудырырга мөмкин.

2002 елда ул үзенең төп хезмәтен – "A New Kind of Science" – бастыра. Бу китапта ул бөтен фәнне һәм табигать законнарын математик модельләр ярдәмендә аңлатырга мөмкинлеген тәкъдим итә. Ул клетчатлы автоматлар ярдәмендә тормыш, табигать, космос, хәтта кеше акылының да эшчәнлеген аңлату мөмкинлеген күрсәтә.

Mathematica һәм WolframAlpha

Стивен Вольфрам фән һәм инженерия өлкәсендә төп эшләренең берсе итеп Mathematica системасын эшләүне саный. 1988 елда чыгарылган бу программа:

- Символик исәпләүләр өчен көчле инструмент.
- Графиклар һәм модельләр ясау мөмкинлеген бирә.
- Алгоритмнар эшләү һәм сынау өчен универсаль платформа булып тора.

2009 елда ул WolframAlpha дип аталган интеллектуаль эзләү системасын тәкъдим итә. WolframAlpha математик һәм фәнни сорауларга җавап табу өчен кулланыла, мәгълүматларны структуралы һәм аңлаешлы формада бирә.

Башка казанышлары

Вольфрам күптөрле фәнни һәм технологик өлкәләрдә эшләгән. Ул физиканы, биологияне һәм химияне математик модельләштерү юлы белән аңлатуга зур өлеш керткән. Аның эшләре бүгенге цифрлы технологияләр һәм ясалма интеллект өчен нигез булып тора.

Бүләкләр һәм танылу

Стивен Вольфрам хезмәтләре өчен күпсанлы бүләкләр белән бүләкләнгән, шул исәптән:

- Макартур премиясе (1981): яшь галимнәр өчен иң мәртәбәле бүләк.
- ACM сыйфатында танылу: компьютер фәненә керткән өлеше өчен.
- Фән һәм технология өлкәсендә инновацияләр өчен премияләр.

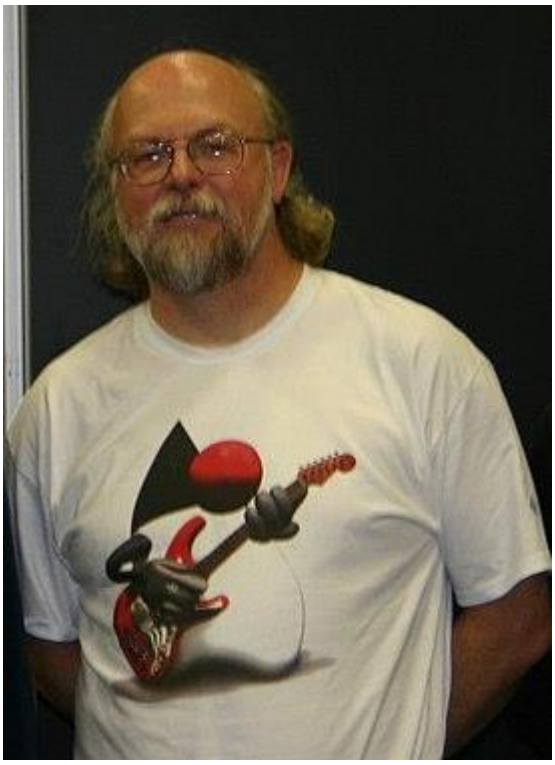
Шәхси тормыш һәм фәлсәфәсе

Вольфрам фәннең популяризациясе белән дә шөгыльләнә. Ул фәнне һәркем өчен аңлаешлы итү кирәклеген пропагандалый. Аның фикеренчә, технология һәм математика кешелекнең төп проблемаларын чишү өчен кулланыла ала.

Йомгак

Стивен Вольфрам – заманча фәннең иң төп фигураларының берсе. Аның Mathematica һәм WolframAlpha системалары фәнни тикшеренүләрне һәм мәгълүмат эшкәртүне гадиләштерде, ә "A New Kind of Science" концепциясе табигатьне аңлауның яңа алымнарын тәкъдим итте. Вольфрамның эшчәнлеге фән, техника һәм мәгариф өлкәләрендә глобаль йогынты ясый.

1.18. Джеймс Гослинг: Java телен булдыручы һәм платформалы программалаштыру революционеры



Джеймс Гослинг

Балачак һәм яшьлек еллары

Джеймс Гослинг 1955 елның 19 маенда Канадада, Калгари шәһәрендә туа. Яшьтән үк ул техника һәм фән белән кызыксына, бигрәк тә электроника һәм программалаштыруга зур сәләт күрсәтә. Мәктәптә укыганда, ул радиотехника белән шөгыльләнә һәм үз компьютерларын ясый.

Гослинг Калгари университетында белем ала һәм компьютер фәне буенча бакалавр дәрәҗәсен ала. Соңрак ул Карнеги-Меллон университетында магистр һәм доктор дәрәҗәләрен үзләштерә, анда ул виртуаль машиналар белән бәйле тикшеренүләр үткәрә. Аның докторлык эшендә күпчелек идеяләр соңрак Java телен булдыруда файдаланыла.

Javaның тууы

1990-нчы елларда Гослинг Sun Microsystems компаниясендә эшли башлый. 1991 елда ул һәм аның командасы “Green Project” дип аталган проект өстендә эшли. Бу проектның максаты – төрле җайланмаларда эшли ала торган универсаль программа платформасы булдыру.

1995 елда Гослинг Java программалаштыру телен тәкъдим итә. Javaның төп үзенчәлекләре:

- Платформага бәйсезлек: Java виртуаль машинасы (JVM) теләсә кайсы платформада эшли ала, бу "бер тапкыр яза һәм һәркайда эшли" концепциясен тормышка ашыра.
- Объект-юнәлешле архитектура: Java катлаулы программалар төзү өчен уңайлы структуралар тәкъдим итә.
- Куркынычсызлык: Java кодны изоляцияләү һәм куркынычсызлыкны тәэмин итү буенча алдынгы ысуллар куллана.

Java тиз арада дөньякүләм популярлык казана һәм хәзерге вакытта серверлар, мобиль җайланмалар, веб-технологияләр, суперкомпьютерлар һәм күп кенә башка өлкәләрдә кулланыла.

Javaның үсеше һәм әһәмияте

Javaның популярлыгы аның универсальлегендә. Ул:

- Android: Android операцион системасы өчен төп программалаштыру теле булып тора.
- Корпоратив системалар: Java платформасы зур масштаблы корпоратив кушымталар өчен стандарт итеп кабул ителгән.
- Веб кушымталар: Java сервер ягыннан эшләү өчен киң кулланыла.

Javaның архитектурасы бүгенге көндәге күпчелек программалаштыру телләренә йогынты ясаган.

Башка казанышлары

Гослинг Javadан кала башка күп санлы проектларда катнашкан, шул исәптән:

- Gosling Emacs: текст редакторы.
- NeWS: график интерфейс өчен системалы платформа.
- Программалар өчен инструментлар һәм башка инновацион системалар.

Бүләкләр һәм танылу

Джеймс Гослинг үзенең эшчәнлеге өчен күпсанлы бүләкләр алган:

- ACM Grace Hopper Award: Javaны булдыру өчен.
- Order of Canada (2013): Канададан чыккан дөньякүләм танылган галим буларак.
- IEEE John von Neumann Medal: Javaның платформалы программалаштыруда революция ясаучы булуын тану өчен.

Соңгы еллары һәм мирас

Гослинг Sun Microsystems компаниясе Oracle тарафыннан сатып алынганнан соң, Google һәм Amazon Web Services кебек оешмаларда эшләвен дәвам итте. Ул һәрвакыт инновацияләргә омтыла, программалаштыруның яңа чикләрен тикшерә һәм Javaның киләчәктәге үсеш юнәлешләрен билгели.

Йомгак

Джеймс Гослинг – Java программалаштыру телен булдыручы, һәм ул платформалы программалаштыруда революция ясады. Аның концепцияләре һәм эшчәнлеге бүгенге цифрлы дөньяның нигезе булып тора. Javaның “бер тапкыр яза һәм һәркайда эшли” принциплары аның инновацияләренең әһәмиятен тулысынча ача.

1.19. Ричард Карп: алгоритмнар теориясе һәм NP-тулы мәсьәләләр өлкәсендәге ачышлары

Ричард Карп

Балачак һәм яшьлек еллары

Ричард Манн Карп 1935 елның 3 январенда Бостон шәһәрендә (Массачусетс, АКШ) туа. Ул математик сәләтле гаиләдә үсә һәм мәктәптә үк үзенең аналитик фикер йөртү сәләтен күрсәтә. Карп Гарвард университетында белем ала, анда ул бакалавр, магистр һәм соңрак доктор дәрәҗәләрен үзләштерә. Белем алу елларында ул математик исәпләүләр һәм оптимизация белән кызыксына башлый.

NP-тулы мәсьәләләр һәм компьютер фәне

Карп 1971 елда үзенең иң мөһим хезмәтләренең берсен бастыра: "NP-тулы мәсьәләләргә кайтарылу". Бу мәкаләдә ул NP-тулы мәсьәләләр төшенчәсен формалаштыра һәм 21 NP-тулы мәсьәлә арасындагы бәйләнешне күрсәтә. Бу ачыш:

- Теоретик информатиканың үсешендә зур роль уйный.
- "P = NP?" мәсьәләсенең мөһимлеген ассызыклый.
- Оптимизация, алгоритмнар һәм исәпләүләрнең чикләрен аңлау өчен нигез булдыра.

Карпның эшләре NP-тулы мәсьәләләрнең күпчелеген чишү өчен полиномиаль алгоритмнарның булу-булмавын ачыклау мәсьәләсен игътибар үзәгенә куйды. Бу бүгенге көндә дә информатиканың төп проблемаларының берсе булып тора.

Алгоритмнар теориясе

Карп алгоритмнарның эффективлыгын анализлау һәм оптимизацияләү өлкәсендә дә зур өлеш керткән. Ул графикларны эшкәртү, динамик программалаштыру һәм таркатылган системалар белән бәйле күп кенә алгоритмнарны эшләп чыгарган.

Аның Карп-Рабин алгоритмы, мәсәлән, текстларны эзләү һәм мәгълүматны тикшерү өчен кулланыла. Бу алгоритм гади һәм эффектив булуы белән киң танылган.

Тикшеренүләр һәм практик кулланышлар

Карпның тикшеренүләре:

- Геномика: Биологиядә ДНК эзлеклелеген анализлау өчен кулланыла.
- Оптимизация: Сәнәгать һәм икътисадтагы катлаулы мәсьәләләрне чишүдә кулланыла.
- Таркатылган системалар: Зур мәгълүматлар һәм челтәрләр белән эшләүдә кулланыла.

Аның эшләре теоретик аспекттан тыш, практик мәгънәгә дә ия, һәм күп тармакларда кулланылыш тапкан.

Бүләкләр һәм танылу

Ричард Карп күпсанлы бүләкләр һәм танылу алган, шул исәптән:

- Тьюринг премиясе (1985): NP-тулы мәсьәләләрне формалаштыру һәм алгоритмнар теориясенә керткән өлеше өчен.
- National Medal of Science (1996): АКШ президенты тарафыннан тапшырылган.
- Harvey Prize (1994): Теоретик һәм практик информатика өлкәсендәге казанышлары өчен.

Соңгы еллары һәм фәнгә йогынтысы

Карп Калифорния университетында (UC Berkeley) профессор булып эшләгән. Ул студентлар укытуга һәм алгоритмнар теориясен үстерүгә зур игътибар биргән. Карп фәндә яңа буын тикшеренүчеләрне тәрбияләгән һәм аларга зур илһам биргән.

Йомгак

Ричард Карп – информатика һәм математиканың чикләрен киңәйткән шәхес. Аның NP-тулы мәсьәләләр һәм алгоритмнар теориясендәге ачышлары фән

өчен фундаменталь әһәмияткә ия. Аның хезмәтләре заманча фәнни тикшеренүләргә һәм технологияләргә йогынты ясавын дәвам итә.

1.20. Кен Томпсон һәм Деннис Ритчи: Unix һәм C теленең авторлары

Кен Томпсон (сул якта) һәм Деннис Ритчи (уң якта)

Кен Томпсон

Балачак һәм белем алу

Кеннет Эверетт Томпсон 1943 елның 4 февралендә Луизиана штаты, Нов-Орлеан шәһәрендә туган. Аның әти-әнисе инженерлар булган һәм бу аны техник өлкәгә тарткан. Томпсон Калифорния университетында физика һәм электроника буенча белем алган һәм 1960-70нче елларда AT&T Bell Labs лабораториясендә эшли башлаган. Бу лабораториядә эшләү Томпсонның карьерасында зур роль уйнаган, чөнки ул монда компьютер фәнендә зур казанышларга ирешкән.

Unix операцион системасының үсеше

1970 елларда Кен Томпсон AT&T Bell Labs лабораториясендә Unix операцион системасын эшләүгә старт бирә. Бу вакытта, лабораториядә шулай ук, Деннис Ритчи белән берлектә, Tompson операцион система өчен җайланмалар уйлап таба. Unix операцион системасының төп идеясы – яңа, модульле һәм кулланучыларга уңайлы система булдыру.

Unіхның уникаль үзенчәлеге аның күп функцияле, берничә файдаланучы өчен ачык һәм яхшы структурланган булуы. Ул система берничә модульдән тора, һәрберсе үз бурычын үти һәм барлык башка модульләр белән бәйләнештә эшли. Шуның нәтиҗәсендә, Unix башка операцион системаларга караганда күбрәк мөмкинлекләргә ия була.

Unix һәм B теле

Unix операцион системасын эшләү өчен, Томпсон үзенең B телен булдырган. B теле – күп өлешләре буенча дәвамлы C теленә юл ачкан беренче программалаштыру теле була. B телен уйлап чыгаруның төп максаты – кечкенә һәм җиңел кулланыла торган операцион системаларны программалаштыру өчен кодлар язуда уңайлык тудыру. Tompsonның B телен эшләүе компьютер фәненә зур йогынты ясаган, чөнки ул системалы программалаштыруның үзенчәлекләрен ачты һәм киләчәктәге C теленең үсешенә нигез салды.

Кен Томпсонның мирасы

- Unix операцион системасы: Unixның төп үзенчәлекләре – модульлек, күп эшче режимы һәм масштаблылык – аның бүгенге көндә дөньякүләм популяр операцион системаларга, шул исәптән Linux һәм macOS, нигез булып торуын тәэмин итә.
- B теле: B теле әлеге заманның башлангыч программалаштыру телләреннән булган һәм C теленең үсеше өчен нигез булды.
- Ачык чыганак принциплары: Unix һәм B теленең ачык чыганаклы булуы бу системаларның киң таралуына ярдәм итте һәм ачык чыганаклы культураның үсешенә зур өлеш кертте.

Деннис Ритчи

Балачак һәм белем алу

Деннис МакКлинток Ритчи 1941 елның 9 сентябрендә Нью-Йорк шәһәрендә туган. Ул укуын Харвард университетында физика һәм математика буенча дәвам итә. Ритчи үзенең техник сәләтләрен иртә үк күрсәтә һәм 1960-70нче елларда AT&T Bell Labs лабораториясенә эшкә урнаша. Әлеге лабораториядә ул Кен Томпсон белән бергә Unix операцион системасын үстерү һәм C телен булдыруда катнаша.

C теленең үсеше

Ритчиның иң зур казанышларының берсе – C программалаштыру телен булдыруы. C теле 1970 елларда B теле нигезендә эшләнде. Ул программалаштыруның системалы һәм югары дәрәҗәдәге теленә әверелде. Ritchie һәм Томпсон C телен уйлап чыгарган вакытта, алар Unix системасын программалаштыру өчен уңайлы, көчле һәм универсаль телне эзләделәр.

C теле үзенең структурлыгы, бертөрле синтаксисы һәм башка телләргә караганда җиңел аңлаешлы булуы белән популярлашты. C программалаштыру

теле соңрак күп кенә операцион системаларның нигезенә әверелде һәм бүгенге көндә дә программалаштыру дөньясында төп роль уйный.

Unix операцион системасы

Деннис Ритчи шулай ук Unix операцион системасын булдыручы төп шәхесләрнең берсе булды. Ритчи һәм Томпсонның эшләве нәтиҗәсендә, Unix системасы берничә файлны эшкәртүгә һәм киң масштаблы мәгълүмат эшкәртүгә мөмкинлек бирә торган көчле һәм тотрыклы система булып үсеш алды. Unix ачык чыганаклы булуы аркасында тиз арада танылу яулады һәм соңрак күп кенә башка операцион системаларга нигез булды.

Деннис Ритчиның мирасы

- C теле: Ritchieның эшләгән C теле бүгенге көндә дә иң популяр һәм күп кулланыла торган программалаштыру телләренең берсе булып тора. Бу телгә нигезләнеп язылган программалар бүгенге көндә барлык платформаларда эшли.
- Unix операцион системасы: Ritchie һәм Томпсонның Unix системасы бүгенге көндә күп кенә үзенчәлекле һәм күп функцияле операцион системаларның нигезе булып тора, шулай ук Linux, macOS һәм башка Unix-подобные системалар.

Программалаштыру культурасы: C теле һәм Unixның ачык чыганаклы моделе программалаштыру дөньясында ачык чыганаклы сообществоның үсешенә зур йогынты ясаган.

Бүләкләр һәм танылу

- Тьюринг премиясе (1983): Деннис Ритчи һәм Кен Томпсонга Unix операцион системасын һәм C телен эшләп чыгару өчен бирелде.
- National Medal of Technology (1999): Технологияләр үсешенә керткән өлеше өчен Ритчи һәм Томпсонга Америка Хөкүмәте тарафыннан бирелгән.
- Милли премияләр һәм орденнар: Алар күп кенә халыкара бүләкләргә лаек булган, шул исәптән Американың техник прогресс өлкәсендә иң зур бүләкләренә.

Соңгы еллары һәм мирас

Деннис Ритчи 2011 елның 12 октябрендә вафат булды, әмма аның хезмәтләре бүгенге көндә дә кулланыла һәм компьютер фәненең киләчәген билгеләүче эшләр булып тора. Кен Томпсон бүгенге көндә дә бүгенге көннең иң алдынгы операцион системаларында һәм C телендә тәжрибәле инженер булып эшли.

Йомгак

Кен Томпсон һәм Деннис Ритчиның эшләре компьютер фәнендә зур революция булды. Алар Unix операцион системасын һәм C телен булдырулары белән бүгенге көннең иң күп кулланылган технологияләренә нигез салдылар. Аларның хезмәтләре компьютер системаларының эшен нигезле рәвештә үзгәртте һәм бүгенге көндә дә дөньяда күп кулланылучы техникалар булып тора.

Икенче бүлек.

Python телендә программалаштыру.

2.1. Кереш. Функцияләр һәм үзгәрешлеләр

Python телендә код язу

VS Code — код язу өчен махсус текст мөхәррире

Программалау серләренә төшенүне башлар алдыннан, сезгә VS Code кушымтасын урнаштыруны тәкъдим итәбез. VS Code — текст мөхәррире генә түгел, ул Python телендә язылган кодны эшләтеп җибәрергә мөмкинлек бирүче көчле инструмент. Башта VS Code программасын ачып, "Extensions" бүлегенә керегез һәм "Python" сүзен языгыз. Табылган Python өстәмәсен урнаштырыгыз. Бу эшне башкаргач, Python телендә үз программаларыгызны язып һәм эшләтеп җибәрә аласыз.

Программа язу өчен "Explorer" тәрәзәсенә кереп, яңа файл булдырыгыз. Мәсәлән, файлның исемен hello.py дип кертегез. Хәзер сез программалаштыру дөньясына аяк баса аласыз.

Файл эченә түбәндәге кодны языгыз:

```
print("hello, world")
```

Бу — программалаштыруны өйрәнүче һәркем язучы классик программа. Ул экранда "hello, world" хәбәрен чыгара.

Программаны эшләтеп җибәрү өчен терминалдан файдаланырга кирәк. Экранның аскы өлешендә урнашкан терминал тәрәзәсенә күчегез һәм анда түбәндәге команданы языгыз:

```
python hello.py
```

Шуннан соң клавиатурадагы Enter төймәсенә басыгыз.

Игътибар итегез: компьютерлар бары тик 0 һәм 1ләрне аңлый. Python телендәге командагызны эшләтү вакытында, интерпретатор сезнең текстны машина кодына, ягъни 0 һәм 1ләргә тәрҗемә итә.

Программа эшләп беткәч, терминалда нәтиҗә күренәчәк:
hello, world

Котлыйбыз! Сез үзегезнең беренче программаны яздыгыз!

Функцияләр

Функцияләр — бу компьютер яки программалаштыру теле белә торган гамәлләр җыелмасы.

`hello.py` программасында, мәсәлән, **`print`** функциясе терминал тәрәзәсенә мәгълүмат чыгарырга сәләтле.

`print` функциясе аргументлар кабул итә. Бу очракта, "hello, world" — әлеге функциянең аргументы.

Хаталар (Bugs)

Хаталар — программалаштыруда табигый күренеш. Алар — сез чишәргә тиешле проблемалар! Борчылмагыз — бу зур программист булуның мөһим бер өлеше.

Мәсәлән, **`hello.py`** программасында **`print("hello, world"`** дип язып, ахыргы кысалы җәяне (**`)`**) төшереп калдырсагыз, компилятор хатаны терминал тәрәзәсендә күрсәтер. Бу хата программаны эшләтергә мөмкинлек бирми.

Еш кына хаталар турында хәбәрләр сезнең ялгышларыгызны аңлата һәм аларны төзәтү өчен киңәшләр бирә. Әмма кайвакыт компилятор ул кадәр файдалы булмаска мөмкин.

Беренче Python программасын яхшырту

Беренче Python программасын *шәхси үзенчәлекле* итеп үзгәртеп була.

`hello.py` текст мөхәррирендә тагын бер функция өстәп карагыз.

`input` функциясе сорауны аргумент итеп ала. Мәсәлән:

```
input("Исемегез ничек? ")
print("hello, world")
```

Әмма әлеге үзгәртү генә кулланучы керткән мәгълүматны чыгару мөмкинлеген бирмәячәк. Моның өчен без сезне үзгәрешлеләр белән таныштырабыз.

Үзгәрешлеләр

Үзгәрешле — бу сезнең программаның эчендә мәгълүмат саклау өчен контейнер.

Программага үзгәрешлене мондый код өстәп булдырырга мөмкин:

```
name = input("Исемегез ничек? ")
print("hello, world")
```

Биредә, **name = input("Исемегез ничек? ")** өлешендә, **=** билгесе махсус роль башкара. Бу билге **input("Исемегез ничек? ")** функциясе кайтарган кыйммәтне **name** үзгәрешлеенә беркетә.

Әгәр кодыгызны түбәндәгечә үзгәртсәгез, хата килеп чыгачак:

```
name = input("Исемегез ничек? ")
print("hello, name")
```

Бу программа, кулланучы нәрсә генә язса да, терминалда һәрвакыт "hello, name" дип чыгарачак.

Кодны яңадан үзгәртсәгез, ул болайрак күренәчәк:

```
name = input("Исемегез ничек? ")
print("hello,")
print(name)
```

Терминалда күренәчәк нәтиҗә мондый булачак:

```
Исемегез ничек? Дәүләт
hello,
Дәүләт
```

Бу дөрес нәтиҗәгә якынаю булып тора!

Комментарийлар

Комментарийлар — программистлар үз кодлары турында язмалар калдыра алырлык урын. Алар үзләренә яки башка программистларга аңлатмалар бирү өчен файдалы.

Мәсәлән, кодыгызны түбәндәгечә үзгәртә аласыз:

```
# Кулланучының исемен сорау
name = input("Исемегез ничек? ")

# Сәламләүне чыгару
print("hello,")
print(name)
```

Комментарийлар сезгә кодны аңлауны җиңеләйтә.

Кодны камилләштерү

Тагын бер яңарту кертергә мөмкин:

```
# Кулланучының исемен сорау
name = input("Исемегез ничек? ")

# Сәламләү һәм исемне чыгару
print("hello,", name)
```

Терминалдан "**Дәүләт**" язылган очракта, нәтиҗә мондый: "**hello, Дәүләт**".

Юллар һәм параметрлар

Кыскача әйткәндә, юл (string) ул текст кисәге, Pythonда **str** дип билгеләнә.

Кодка яңадан әйләнеп кайтыйк:

```
# Кулланучының исемен сорау
name = input("Исемегез ничек? ")
print("hello,")
print(name)
```

Бу кодның нәтиҗәсе берничә юлга бүленә:

```
Исемегез ничек? Дәүләт
hello,
Дәүләт
```

Функцияләр үзләренең эшчәнлеген үзгәртә торган аргументлар кабул итә. Мәсәлән, **print** функциясе документлаштыруда **end='\n'** аргументын куллануын күрергә мөмкин. Бу **\n** символы яңа юлдан башлап ясарга мөмкинлек бирә.

Әмма без үзебез **end** өчен башка аргумент билгеләп, яңа юлга күчүне булдырмаска мөмкин. Кодны болай үзгәртегез:

```
# Кулланучының исемен сорау
name = input("Исемегез ничек? ")
print("hello,", end="")
print(name)
```

Мондый үзгәртү кертелгән очракта, "**Дәүләт**" исемен язганда, нәтиҗә **hello, Дәүләт** булачак.

Куш тырнаклар белән проблемалар

Юлга куш тырнаклар кертү кайвакыт авырлык тудыра. Мәсәлән:

```
print("hello,"friend"")
```

Бу хата чыгара. Әлеге проблеманы ике ысул белән төзәтергә була:

1. Яхшы вариант — тышкы куш тырнакларны бер-берсенә туры килми торган итеп үзгәртү:
2. `print('hello, "friend"')`
3. Икенче вариант — кире сызык (\) куллану: `python print("hello, \"friend\"")`

Бу очракта **** билгесе компиляторга кавычканы юлның бер өлеше итеп кабул итәргә ярдәм итә.

Форматланган юллар

Иң яхшы ысул — форматланган юллар куллану:

```
# Кулланучының исемен сорау
name = input("Исемегез ничек? ")
print(f"hello, {name}")
```

Биредә **f** символы Pythonга юлны махсус итеп эшкәртүне аңлата. Мондый формат сезнең өчен иң еш кулланыла торган формат булачак.

Юллар белән эшләүнең өстәмә мөмкинлекләре

Кулланучының керткән мәгълүматы еш кына хаталы була. Шуңа күрә бу мәгълүматны тикшерү һәм төзәтү мөһим.

Мәсәлән, кулланучы исеме алдыннан һәм артыннан буш урыннар булырга мөмкин. Аларны **strip** методы белән бетереп була:

```
# Кулланучының исемен сорау
name = input("Исемегез ничек? ")

# Буш урыннарны бетерү
name = name.strip()

# Нәтиҗә чыгару
```

```
print(f"hello, {name}")
```

Мәсәлән, әгәр кулланучы " **Дәүләт** " дип язса, программа аны "**Дәүләт**" итеп эшкәртәчәк.

Шулай ук исемнең беренче хәрефләрен баш хәрефкә әйләндерү өчен **title** методын кулланыгыз:

```
# Кулланучының исемен сорау
name = input("Исемегез ничек? ")

# Буш урыннарны бетерү һәм беренче хәрефләрне зур итеп язу
name = name.strip().title()

# Нәтиҗә чыгару
print(f"hello, {name}")
```

Үзгәрешлеләр белән эшне камилләштерү

Программа кодын тагын да кыскарту өчен, берничә методны берьюлы кулланып була:

```
# Кулланучының исемен сорау, буш урыннарны бетерү, баш хәрефләрне зур итү
name = input("Исемегез ничек? ").strip().title()

# Нәтиҗә чыгару
print(f"hello, {name}")
```

Бөтен үзгәрешлеләр (int)

Pythonда бөтен саннар **int** дип атала. Математикадагы гадәти гамәлләр (+, -, *, /) белән беррәттән, Pythonда **%** — модуль (калдык табу) операторы да бар.

Яңа калькулятор программасын ясап карыйк. Моны эшләү өчен терминалдан **calculator.py** файлын башлагыз:

```
x = int(input("x санын кертегез:"))
y = int(input("y санын кертегез:"))

print(x + y)
```

Бу программа кулланучыга ике сан кертергә һәм аларның суммасын күрсәтергә мөмкинлек бирә.

Вакланмалы саннар (float)

Pythonда вакланмалы саннар "floating-point numbers" яки "float" дип атала. Бөтен һәм вакланмалы өлешне аеру өчен нокта символын кертәләр: мәсәлән, безгә гадәти булган 2,5 саны программада **`2.5`** дип язылачак.

Мәсәлән, калькулятор программасын үзгәртеп карыйк:

```
x = float(input("x санын кертегез:"))
y = float(input("y санын кертегез:"))

print(x + y)
```

Әлеге код кулланучының саннарын кертергә мөмкинлек бирә. Мәсәлән, 1.2 һәм 3.4 саннары керткәндә, нәтиҗә 4.6 була.

Саннарны түгәрәкләү (round)

Әгәр саннарны якынча бер бөтен санга түгәрәкләргә теләсәгез, **`round`** функциясен куллана аласыз. Мәсәлән:

```
# Кулланучының мәгълүматын алу
x = float(input("x санын кертегез:"))
y = float(input("y санын кертегез:"))

# Суммаы түгәрәкләү
z = round(x + y)

# Нәтиҗәне чыгару
print(z)
```

Бу код суммаы иң якын булган бөтен санга кадәр түгәрәкли.

Саннарны түгәрәкләү һәм форматлау

Кайвакыт, калькулятор нәтиҗәләрен 2 билгеләмәле урынга кадәр күрсәтергә кирәк булырга мөмкин. Моның өчен **`round`** яки f-string кулланыла ала:

```
# Кулланучының мәгълүматын алу
x = float(input("x санын кертегез:"))
y = float(input("y санын кертегез:"))

# Нәтиҗәне түгәрәкләү
```

```
z = round(x / y, 2)

# Нәтиҗәне чыгару
print(z)
```

Яисә f-string кулланып:

```
# Кулланучының мәгълүматын алу
x = float(input("x санын кертегез:"))
y = float(input("y санын кертегез:"))

# Нәтиҗәне чыгару
print(f"{x / y:.2f}")
```

Функцияләр игълан итү

Үз функцияләрегезне игълан итү файдалы һәм мөһим күнекмә булып тора. Мәсәлән:

```
# Функция игълан итү
def hello(to="дөнья"):
    print(f"Сәлам, {to}")

# Функцияне куллану
name = input("Исемегез ничек? ")
hello(name)

# Стандарт аргумент куллану
hello()
```

Биредә, **hello** функциясе үзгәрешле "**to**" параметрын кабул итә. Әгәр параметр бирелмәсә, ул стандарт рәвештә "дөнья" дип билгеләнә.

main функциясен куллану

Pythonда программа логикасын ачык һәм төгәл итү өчен **main** функциясен кулланырга була. Мәсәлән:

```
# Төп функция
def main():
    name = input("Исемегез ничек? ").strip().title()
    hello(name)

# Сәламләү функциясе
def hello(to="дөнья"):
```

```
    print(f"Сәлам, {to}")

# Программаны эшләтү
main()
```

Бу структура программаны оештыруны җиңеләйтә.

Кыйммәт кайтару (Return values)

Функцияләр кайвакыт мәгълүматны үзгәртү өчен, яки исәпләү нәтиҗәләрен кире кайтарырга мөмкин. Мәсәлән:

```
# Төп функция
def main():
    x = int(input("x санын кертегез:"))
    print(f"x квадрат — {square(x)}")

# Квадрат исәпләү функциясе
def square(n):
    return n * n

# Программаны эшләтү
main()
```

Бу кодта **square** функциясе санның квадратын исәпли һәм аны төп функциягә кире кайтара.

2.2. Шартлы операторлар

Шартлы операторлар сезгә, программист буларак, программаны карарлар кабул итәргә мөмкинлек бирә: Мәсәлән, билгеле шартларга нигезләнеп, программаның сул юлга бару яки уң юлга бару.

Python эчендә математик сораулар бирү өчен кулланыла торган "операторлар" җыелмасы бар.

> һәм **<** символлары сезгә бик таныш булырга мөмкин.

>= "тик тигез яки зуррак" дигәнне аңлата.

<= "тик тигез яки кечкенәрәк" дигәнне аңлата.

== "тигез" дигәнне аңлата, ләкин икеле тигезлекне искә алыгыз! Бер генә тигезлек билгесе бәяне билгеләр иде. Икеле тигезлек билгеләре үзгәрешлеләрне чагыштыру өчен кулланыла.

!= "тигез түгел" дигәнне аңлата.

Шартлы операторлар сул яктан уң ягындагы терминны чагыштыралар.

if

Терминал тәрәзәсендә **compare.py** язып яңа файл булдырыгыз. Текст редакторында түбәндәге кодны языгыз:

```
x = int(input("x санын кертегез:"))
y = int(input("y санын кертегез:"))

if x < y:
    print("x, y тан кечкенәрәк")
```

Бу программа кулланучыдан x һәм y саннарын кертергә сорый, аларны бөтен саннарга әйләндерә һәм **x < y** шартын тикшерә. Әгәр шарт үтәлсә, программа "**x, y дан кечкенәрәк**" дигәнне бастыра.

if операторлары **bool** яки **boolean** кыйммәтләрен (дөрес яки ялган) кулланып, кодны эшкәртергә мөмкинлек бирә. Әгәр **x > y** шартын дөрес дип тапса, компилятор аны дөрес дип таный һәм кодны башкара.

Контроль агымы, elif һәм else

Кодыгызны түбәндәгечә яңартыгыз:

```
x = int(input("x санын кертегез:"))
y = int(input("y санын кертегез:"))

if x < y:
    print("x, y тан кечкенәрәк")
elif x > y:
    print("x, y тан зуррак")
elif x == y:
    print("x һәм y тигез")
```

Бу программа сериясе **if** операторларын кулланып, шартларны тикшерә. Беренче **if** шартын бәяли, аннары **elif** шартларын һәм ахырда **else** шартын тикшерә. Бу процессны "контроль агымы" дип атыйлар.

Бу программа өч шартны бер-бер артлы тикшерә, ләкин бары тик бер генә шарт үтәлә ала.

Программагызны түбәндәгечә яхшыртыгыз:

```
x = int(input("x санын кертегез:"))
y = int(input("y санын кертегез:"))

if x < y:
    print("x, y дан кечкенәрәк")
elif x > y:
    print("x, y дан зуррак")
else:
    print("x, y тигез")
```

Бу версиядә, соңгы elif шартын else белән алмаштырдык, чөнки әгәр x < y һәм x > y шартлары үтәлмәсә, x һәм y тигез булырга тиеш.

or

or операторын кулланып, программаны берничә альтернатива арасында сайларга мөмкинлек бирә. Мәсәлән, программаны түбәндәгечә үзгәртегез:

```
x = int(input("x санын кертегез:"))
y = int(input("y санын кертегез:"))

if x < y or x > y:
    print("x һәм y тигез түгел")
else:
    print("x һәм y тигез")
```

Бу программа шартны киметә, **"x < y яки x > y"** булса, **"x һәм y тигез түгел "** дип чыгара. **else** блогы калган очракны каплый.

Кодыгызны түбәндәгечә яңартыгыз:

```
x = int(input("x санын кертегез:"))
y = int(input("y санын кертегез:"))

if x != y:
    print("x һәм y тигез түгел")
else:
    print("x һәм y тигез")
```

Бу версиядә, **"x != y"** шартын кулланып, программа бер генә сорау бирә, бу исә кодның эффективлыгын арттыра.

Тагын бер мисал:

```
x = int(input("x санын кертегез:"))
y = int(input("y санын кертегез:"))

if x == y:
    print("x һәм y тигез")
else:
    print("x һәм y тигез түгел")
```

Бу код да бер үк нәтиҗәне бирә, ләкин ул **"x == y"** шартын беренче итеп тикшерә.

and

and операторын шартлы операторларда куллану мөмкинлеге бар.

КФУда кулланыла торган балл-рейтинг системасында кертелгән балларны билгеләргә алыштырыйк. **grade.py** исемле яңа файл булдырыгыз. Текст редакторында түбәндәге кодны языгыз:

```
score = int(input("Балл: "))

if score >= 86 and score <= 100:
    print("Билге: 5")
elif score >= 71 and score < 86:
    print("Билге: 4")
elif score >= 56 and score < 71:
    print("Билге: 3")
else:
```

```
    print("Билге: 2")
```

Бу программа кулланучыдан баллны кертергә сорый һәм аны тиешле билгегә алаштыра. Әмма, бу кодта хаталар мөмкинлеге бар.

Кодыгызны түбәндәгечә яхшыртыгыз:

```
score = int(input("Балл: "))

if 86 <= score <= 100:
    print("Билге: 5")
elif 72 <= score < 86:
    print("Билге: 4")
elif 56 <= score < 72:
    print("Билге: 3")
else:
    print("Билге: 2")
```

Pythonда операторларны *чылбырлап* куллану мөмкинлеге бар, бу башка телләрдәге программистлар өчен бик кызыклы.

Кодны тагы да яхшыртыйк:

```
score = int(input("Балл: "))

if score >= 86:
    print("Билге: 5")
elif score >= 72:
    print("Билге: 4")
elif score >= 56:
    print("Билге: 3")
else:
    print("Билге: 2")
```

Бу версиядә, һәр этапта гади генә шартлар кулланыла, бу кодны уку һәм аңлауны җиңеләйтә.

Модуло

Математикада паритет — санның җәп яки так булуын аңлата.

Программалаштыруда модуло `%` операторы ике санның тигез бүленүен яки калдык белән бүленүен тикшерергә мөмкинлек бирә.

Мәсәлән, 4 % 2 = 0, чөнки ул тигез бүленә. Ләкин, 3 % 2 ≠ 0, чөнки калдык бар.

parity.py исемле яңа файл булдырыгыз. Текст редакторында түбәндәге кодны языгыз:

```
x = int(input("x санын кертегез:"))

if x % 2 == 0:
    print("Пар")
else:
    print("Так")
```

Бу программа кулланучыдан сан кертергә сорый һәм аны пар яки тики дип билгели.

Үзебезнең Паритет Функциясен Булдыру

Беренче бүлектә өйрәнгән үрнәкләрне кулланып, үзегезнең яңа функцияләрегезне игълан итәргә мөмкин.

Кодыгызны түбәндәгечә яңартыгыз:

```
def main():
    x = int(input("x санын кертегез:"))
    if is_even(x):
        print("Җөп")
    else:
        print("Так")

def is_even(n):
    if n % 2 == 0:
        return True
    else:
        return False

main()
```

Бу код **is_even** функциясен кулланып, санның пармы яки тики булуын тикшерә.

“Pythonic”, яки “Питонча” код

Программалаштыру дөньясында, кайбер программалаштыру ысуллары "Pythonic" дип атала. Бу — Pythonның үзенчәлекле һәм уңайлы ысуллары.

Кодыгызны түбәндәгечә яңартыгыз:

```
def main():
```

```
    x = int(input("x санын кертегез:"))
    if is_even(x):
        print("Җөп ")
    else:
        print("Так")

def is_even(n):
    return True if n % 2 == 0 else False

main()
```

is_even функциясендә шартлы операторларны куллану ысулына игътибар итегез. Мондый язуы кыска һәм бик аңлаешлы.

Тагын да “Pythonic” итеп языйк:

```
def main():
    x = int(input("x санын кертегез:"))
    if is_even(x):
        print("Җөп")
    else:
        print("Так")

def is_even(n):
    return n % 2 == 0

main()
```

Бу версиядә, **return** операторын турыдан-туры кулланыла, ә функция кыскара.

match

if, elif, else операторларына охшаш, match операторлары билгеле кыйммәтләргә туры килгәндә кодны башкарырга мөмкинлек бирә.

Мәсәлән, түбәндәге программаны карагыз:

```
name = input("Исемегез ничек?")

if name == "Harry":
    print("Gryffindor")
elif name == "Hermione":
    print("Gryffindor")
elif name == "Ron":
    print("Gryffindor")
```

```
elif name == "Draco":
    print("Slytherin")
else:
    print("Кем?")
```

Бу кодны match операторын кулланып, түбәндәгечә яхшыртырга мөмкин:

```
name = input("Исемегез ничек?")

match name:
    case "Harry" | "Hermione" | "Ron":
        print("Gryffindor")
    case "Draco":
        print("Slytherin")
    case _:
        print("Кем?")
```

Бу версиядә, **match** операторын кулланып, берничә кыйммәтне бер шартта тикшердек. **_** символы else шартын кабатлый, ягъни башкача әйткәндә, калган барлык очракларны каплый. Шулай ук бу мисалда штрих **|** операторы **or** операторын алыштыра.

2.3 Цикллар, исемлекләр һәм сүзлекләр

Гадәттә, цикллар — нәрсәне дә булса берничә тапкыр кабатларга мөмкинлек бирә торган конструкция.

cat.py исмеле яңа файл булдырыгыз.

Текст редакторында түбәндәге кодны языгыз:

```
print("мырау")
print("мырау")
print("мырау")
```

Бу кодны эшләткәндә, программаның өч тапкыр "**мырау**" дигәнен бастыруын күрерсез.

Яхшы программист үз кодында кабатлана торган фрагментларны камилләштерергә омтыла. Мәсәлән, сез "**мырау**"ны 500 тапкыр әйтергә теләсәгез, **print("мырау")**ны һәр тапкыр кабатлап язарга дөрес булачакмы?

Мондый очракта цикллар сезгә ярдәмгә килә, алар берничә тапкыр башкарасы код блогын булдырырга мөмкинлек бирә.

While цикллары

while циклы барлык программалаштыру телләрендә киң таралган.

Шул тип цикл берничә тапкыр код блокын кабатлый.

Текст редакторында, кодыгызны түбәндәгечә үзгәртегез:

```
i = 3
while i != 0:
    print("мырау")
```

Бу код **print("мырау")** командасын эшли, ләкин туктамаячак. Ул өзлексез кабатланачак. **while** цикллары циклның шартын үтәлгәнен сорыйлар. Бу очракта, компилятор **"i нольгә тигез түгелме?"** дип сорый. Циклны туктату өчен сез клавиатурада **CTRL+C** төймәсенә басып циклдан чыга аласыз.

Бу өзлексез дәвам иткән циклны төзәтү өчен, кодыгызны түбәндәгечә үзгәртегез:

```
i = 3
```

```
while i != 0:
    print("мырау")
    i = i - 1
```

Хәзер код дөрес эшли, һәр цикл узганда **i** 1 гә кими. Итерация (iteration) термины программалаштыруда махсус мәгънәгә ия. Итерация — циклның бер әйләнеше. Беренче итерация "0-нче" итерация дип әйтәбез, икенчесе "1-нче" итерация, һәм шулай дәвам итәбез. Программалаштыруда саннарны 0-дән башлап саныйбыз.

Кодыгызны тагын да яхшыртып, 0-дән башлап карыйк:

```
i = 0
while i < 3:
    print("мырау")
    i += 1
```

Бу кодта **i < 3** операторын кулланып, циклның шартын дөрес итеп куябыз. **i += 1** - бу **i = i + 1** белән бер үк нәрсә, ләкин кыскарак язылыш. Хәзер **i** 0-дән башлап санала һәм цикл өч тапкыр эшли, нәтиҗәдә өч "мырау" басылачак.

Бу цикл **i** ны 3 кадәр санап, ләкин 3 гә җитмичә туктала.

For цикллары

for циклының эшләү принциба башкачарак.

for циклын яхшы аңлау өчен, Pythonдагы яңа үзгәрешле төре — исемлек (list) турында сөйләргә кирәк. Кешеләр тормышында кибет исемлеге, эшләр исемлеге кебек исемлекләр бар, шулай ук программалаштыруда да исемлекләр керткәннәр. Кайбер программалаштыру телләрендә исемлекләрне массив дип тә йөртәләр.

for циклы исемлек элементлары буенча әйләнә. Мәсәлән, текст редакторында **cat.py** кодыгызны түбәндәгечә үзгәртегез:

```
for i in [0, 1, 2]:
    print("мырау")
```

Бу код алдагы **while** циклына караганда чистарак. Бу кодта **i** 0-дән башлап, "**мырау**"ны бастыра, аннан **i** 1-гә, "**мырау**"ны бастыра, ахырда **i** 2-гә җиткәч, "**мырау**"ны бастыра һәм туктый.

Бу код безнең максатыбызны үти, ләкин экстремаль очракларны карау өчен кодны яхшыртырга мөмкин. Мәсәлән, циклны миллион тапкыр эшләргә

теләсәгез, кодны ничек игълан итү дөрес булыр иде. Шуңа күрә кодыгызны түбәндәгечә үзгәртегез:

```
for i in range(3):
    print("мырау")
```

range(3) автоматик рәвештә өч сан (0, 1, 2) кайтарачак. Бу код өч тапкыр **"мырау"**ны бастырачак.

Кодыгызны тагын да яхшыртып, **i** ны кулланмыйча, **_** билгеләмәсе куллана аласыз:

```
for _ in range(3):
    print("мырау")
```

Бу үзгәреш программа эшләүенә тәэсир итми, шул ук вакытта үзгәрешлесез цикл ясарга мөмкиинлек бирә.

Тагын да яхшыртып, түбәндәге кодны карагыз:

```
print("мырау\n" * 3, end="")
```

Бу код өч тапкыр **"мырау"**ны аерым юлларда бастырачак. **\n** символы юлны яңадан башларга мөмкинлек бирә, ә **end=""** параметры бастырудан соң өстәмә юл бушлыгы булдырмый.

Кертү ысулын яхшарту

Мөгаен, сез кулланучыдан мәгълүмат алырга теләрсез. Цикллар кулланучы кертемнәрен тикшерү өчен уңайлы ысул булып тора.

Pythonда киң кулланылган парадигма — кулланучының кертемнәрен тикшерү өчен **while** циклы куллану.

Мәсәлән, кулланучыдан 0-дән зуррак сан сорап, түбәндәге кодны языгыз:

```
while True:
    n = int(input("n санын кертегез:"))
    if n < 0:
        continue
    else:
        break
```

Бу кодта ике яңа сүз — **continue** һәм **break** кулланылган. **continue** — циклның киләсе итерациясенә күчәргә кирәклеген белдерә, ә **break** — циклдан иртәрәк чыгуны тәэмин итә. Бу очракта, әгәр **n** 0-дән кечкенә булса,

циклны дәвам итәбез, ягъни кулланучыга яңадан сорап, әгәр **n** зуррак яки тигез булса, циклны туктатабыз һәм программа дәвам итә.

Ләкин бу очракта **continue** сүзен куллану кирәкми. Кодыгызны түбәндәгечә үзгәртегез:

```
while True:
    n = int(input("n санын кертегеҙ:"))
    if n > 0:
        break

for _ in range(n):
    print("мырау")
```

Бу цикл һәрчак эшләп тора, тик **n** 0-дән зуррак булганда гына туктый.

Алдагы өйрәнүләрне кулланып, кодыгызны тагын да яхшыртырга мөмкин:

```
def main():
    meow(get_number())

def get_number():
    while True:
        n = int(input("n санын кертегеҙ:"))
        if n > 0:
            return n

def meow(n):
    for _ in range(n):
        print("мырау")

main()
```

Бу код күп яхшы үзенчәлекләргә ия. Без сан алу мөмкинлеген **get_number** функциясендә яздык, һәм төп программабыз өч юлга кыскарды. Шулай ук, **return** кулланып, санны кайтарып бирәбез.

Исемлекләр турында өстәмә мәгълумат

Терминал тәрәзәсендә **hogwarts.py** языгыз.

Текст редакторында түбәндәге кодны языгыз:

```
students = ["Hermoine", "Harry", "Ron"]

print(students[0])
```

```
print(students[1])
print(students[2])
```

Бу кодта без студентлар исемлеген саклыйбыз һәм аларның һәрберсен аерым язабыз. Беренче позициядәге студент — “Hermoine”, икенчесе “Harry”, өченчесе “Ron”.

Шулай ук, цикл кулланып, исемлек буенча әйләнергә мөмкин:

```
students = ["Hermoine", "Harry", "Ron"]

for student in students:
    print(student)
```

Бу кодта цикл исемлекнең һәр элементын алу һәм бастыру өчен кулланыла. Бу юлы, **_** билгеләмәсе кулланмыйбыз, чөнки **student** үзгәрешлеен код эчендә кулланабыз.

Без **len** функциясен кулланып, исемлекнең озынлыгын тикшерә алабыз.

Мәсәлән, исемлекнең һәр студентның позициясен һәм исемен бастырырга теләсәгез, түбәндәге кодны кулланыгыз:

```
students = ["Hermoine", "Harry", "Ron"]

for i in range(len(students)):
    print(i + 1, students[i])
```

Бу кодта **len(students)** исемлекнең озынлыгын кайтара, һәм **range(len(students))** циклы аша бара. **i + 1** белән без позицияне 1-дән башлап саныйбыз, чөнки кешеләр гадәттә 1-дән саныйлар.

Сүзлекләр

dicts яки сүзлекләр — ачкычларны кыйммәтләргә бәйләүче мәгълүмати структура.

Мәсәлән, Гарри Поттерның өйләрен һәм студентларын ассоциацияләү:

Исемлекләр белән:

```
students = ["Hermoine", "Harry", "Ron", "Draco"]
houses = ["Gryffindor", "Gryffindor", "Gryffindor",
"Slytherin"]

print(students[0], houses[0])
```

```
print(students[1], houses[1])
print(students[2], houses[2])
print(students[3], houses[3])
```

Бу ысул белән без ике исемлекне бер-беренә бәйләргә тиешбез, ләкин зур исемлекләр белән эш иткәндәмәшәкатькә китерергә мөмкин.

Сүзлекләрнең үрнәге түбәндә күрсәтелгән:

```
students = {
    "Hermoine": "Gryffindor",
    "Harry": "Gryffindor",
    "Ron": "Gryffindor",
    "Draco": "Slytherin",
}

print(students["Hermoine"])
print(students["Harry"])
print(students["Ron"])
print(students["Draco"])
```

Бу ысул белән без һәр студентка туры килгән өйне саклыйбыз. Сүзлекләр **{}** фигурлы кыстырмалар кулланып төзелә.

Текст редакторында, сүзлекне цикл белән әйләндереп, барлык студентларны һәм өйләрен бастырырга мөмкин:

```
students = {
    "Hermoine": "Gryffindor",
    "Harry": "Gryffindor",
    "Ron": "Gryffindor",
    "Draco": "Slytherin",
}

for student in students:
    print(student, students[student], sep=", ")
```

Бу код һәр студентның исемен һәм туры килгән өйне бастырачак. **sep=", "** параметры белән һәр элемент арасында буш урын куябыз.

2.4. Искәрмәләр (Exceptions)

Искәрмәләр — безнең кодта хаталар килеп чыккан вакытта килеп чыга торган хәлләр.

hello.py исемле файл булдырыгыз. Текст редакторында, түбәндәге кодны языгыз (*игътибар, кодка хаталар кертелгән*):

```
print("сәлам, дөнья)
```

Бу кодта без бер куш тырнакны калдырып киттек.

Терминал тәрәзәсендә **python hello.py** язып эшләтегез, һәм сез хата хәбәрен күрерсез. Компилятор **" syntax error "** дип белдерәчәк. Синтаксис хаталары — сезнең кодны дөрес язып булмавын таләп итә торган хаталар.

Pythonның **Errors and Exceptions** документациясен укып күбрәк белә аласыз.

Эшләү вакытында киләп чыгучы хаталар (Runtime Errors)

Эшләү вакытында киләп чыгучы хаталар — сезнең кодта көтелмәгән галәм аркасында килеп чыккан хаталар. Мәсәлән, сез кулланучыдан сан кертүне өмет итәсез, ләкин ул символ яки хәреф кертә. Программагыз бу көтелмәгән кертү аркасында хата чыгарырга мөмкин.

number.py исемле файл булдырыгыз. Текст редакторында, түбәндәге кодны языгыз:

```
x = int(input("x санын кертегез:"))
print(f"x {x}")
```

Монда **f** префиксы белән без Pythonга фигуралы сызыклар (**{}**) эчендәге мәгълүматны *интерполяцияли*. Кодыгызны тестлап караганда, кулланучы хәреф яки символ керткәндә (мәсәлән, **"cat"**), программа **ValueError: invalid literal for int() with base 10: 'cat'** хата бирәчәк. Бу, Python интерпретаторының **int** функциясенә **"cat"** ны сан итеп кабул итмәвен күрсәтә.

Программист буларак, без кулланучыларның көтелгәнчә мәгълүмат кертүләрен тәэмин итәргә тиешбез.

try

Pythonда **try** һәм **except** командаларын куллану — хаталар килеп чыкканда аларны тоту һәм эшкәртү ысулы. Кодыгызны түбәндәгечә үзгәртегез:

```
try:
    x = int(input("x санын кертегеҙ:"))
    print(f"x {x}")
except ValueError:
    print("x сан түгел")
```

Бу кодны эшләткәндә, 50 кертсәгез, ул кабул ителәчәк. Әмма, **"cat"** кертсәгез, программа хатаны тотып, **"x сан түгел"** дип хәбәр итәчәк.

Бу әле дә иң яхшы ысул түгел, чөнки без берничә юлны берьюлы **try** блокына керттек. Иң яхшы практика буенча, без мөмкин кадәр аз юлны **try** блокына кертергә тиешбез, бу юлларда хата килеп чыгу ихтималы бар. Кодыгызны түбәндәгечә үзгәртегез:

```
try:
    x = int(input("x санын кертегеҙ:"))
except ValueError:
    print("x сан түгел")
else:
    print(f"x {x}")
```

Бу очракта, әгәр хата килеп чыкмаса, **else** блогы эшлиячәк. Шулай итеп, без **x** ны игълан итүны **try** блокына куйдык, һәм **print** юлны **else** блокына күчердек.

else

else блогы **try** блокында хата килеп чыкмаса эшләнә. Кодыгызны яңадан карагыз:

```
try:
    x = int(input("x санын кертегеҙ:"))
except ValueError:
    print("x сан түгел")
else:
    print(f"x {x}")
```

Бу кодны эшләткәндә, 50 кертсәгез, **"x 50"** дип бастырылачак. **"cat"** кертсәгез, **"x сан түгел"** дип хәбәр ителәчәк.

Кодыгызны тагын да яхшыртырга теләсәк, кулланучы дөрес мәгълүмат кертмичә, программаны туктатмыйк. Аның өчен цикл кулланырга мөмкин. Кодыгызны түбәндәгечә үзгәртегез:

```
while True:
    try:
        x = int(input("x санын кертегез:"))
    except ValueError:
        print("x сан түгел")
    else:
        break

print(f"x {x}")
```

Бу код, кулланучы дөрес сан керткәнче, аны кире сорар. Дөрес сан керткәндә, программа циклднан чыгачак һәм нәтиҗәне бастырачак.

Мөгаен, программистлар тарафыннан языла торган һәм иң киң кулланыла торган функция — ул кулланучылардан сан алу функциясе. Кодыгызны түбәндәгечә үзгәртегез:

```
def main():
    x = get_int()
    print(f"x {x}")

def get_int():
    while True:
        try:
            x = int(input("x санын кертегез:"))
        except ValueError:
            print("x сан түгел")
        else:
            break
    return x

main()
```

Бу код күп яхшы үзенчәлекләргә ия. Без сан алу мөмкинлеген **get_int** функциясенә *абстрактлаштырдык*, һәм төп программабыз өч юлга кыскартылды.

Бу программаны тагын да яхшыртырга мөмкин. **get_int** функциясе нәтиҗәне циклдан чыгу белән генә түгел, аны кайтару (**return**) белән дә эшли. Кодыгызны түбәндәгечә үзгәртегез:

```
def main():
```

```
    x = get_int()
    print(f"x {x}")

def get_int():
    while True:
        try:
            x = int(input("x санын кертегез:"))
        except ValueError:
            print("x сан түгел")
        else:
            return x

main()
```

Бу версиядә, **return** функциясе циклны туктатып, санны кайтара.

Кайберәүләр түбәндәге кодны язарга мөмкиннәр:

```
def main():
    x = get_int()
    print(f"x {x}")

def get_int():
    while True:
        try:
            return int(input("x санын кертегез:"))
        except ValueError:
            print("x сан түгел")

main()
```

Бу да алдагы версия кебек эшли, ләкин юллар саны кимегән.

pass

Программа белән эшләгән вакытта хата килеп чыккан очракта, без аны кулланучыга әйтмичә кире баштан мүгъламт кертүне оештыра алабыз. Кодыгызны түбәндәгечә үзгәртегез:

```
def main():
    x = get_int()
    print(f"x {x}")

def get_int():
    while True:
        try:
```

```
            return int(input("x санын кертегез:"))
        except ValueError:
            pass

main()
```

Бу код элеккеге функцияләр кебек эшли, ләкин кулланучыга хата турында хәбәр итми. Кайбер очракларда, сез кулланучыга хаталар турында хәбәр итәргә теләрсез, кайбер очракларда исә гади генә кире сорарга теләрсез.

Сез бу функцияне тагын да яхшыртып, кертү соравын функциягә күчерә аласыз:

```
def main():
    x = get_int("x санын кертегез:")
    print(f"x {x}")

def get_int(prompt):
    while True:
        try:
            return int(input(prompt))
        except ValueError:
            pass

main()
```

Бу версиядә, **get_int** функциясе сорау текстын параметр буларак ала, бу аны тагын да уңайлырак итә.

2.5. Модульләр

Модульләр, гадәттә, сезнең үзегез яки башка кешеләр тарафыннан язылган код өлешләре булып тора, һәм аларны сез үз программаларыгызда куллана аласыз. Python модульләре ярдәмендә функцияләрне һәм мөмкинлекләрне башкалар белән уртаклашу бик җиңел. Искергән проекттагы файдалы кодны күчереп, аерым модуль итеп ясап, аны яңа проектларда кабат куллану өчен әзерләргә мөмкин.

Random

random — Python белән бергә килгән модуль, аны сез үз проектларыгызга җиңел генә импортлап куллана аласыз. Кодер буларак, элекке эшләнмәләргә таянып эшләү сезнең эшегезне җиңеләйтә. Ә ничек итеп модульне үз программагызга өстәргә? Моның өчен **import** сүзен кулланырга кирәк.

random модулендә **random.choice(seq)** дип аталган файдалы функция бар. **random** — сез импортлый торган модуль, һәм бу модульдә **choice** функциясе урнаштырылган. Әлеге функциягә **seq** дип аталган тәртипкә салынган исемлек бирәп, шуннан бер элемент сайлап алырга мөмкин.

generate.py исемле файл булдырыгыз. Шуннан соң текст редакторында түбәндәге кодны өстәгез:

```
import random

# Тәртипкә салынган исемлек
items = ['алма', 'банан', 'җиләк', 'карбыз']

# Очраклы элементны сайлау
selected_item = random.choice(items)
print(f"Сайланган элемент: {selected_item}")
```

Шушы кодны саклап, **generate.py** исемендәге файлда эшләвегезне тикшерегез. Терминалда **python generate.py** командасын язып, нәтиҗәне карагыз.

choice функциясе белән эш иткәндә, квадрат кыстырмалар, куш тырнаклар һәм өтерләр ярдәмендә исемлек төзелгәнен күрәсез. Әгәр ике генә элемент тапшырылган булса, Python автомат рәвештә һәр элементка тигез шанс (бирелгән мисалда — 25%) билгели. Кодыгызны эшләткәндә, бу очраклы сайлауның дөрес эшләвен күреп була.

Кодны эффективрак итеп үзгәртү өчен, **from** операторын кулланырга мөмкин. Ул безгә конкрет модульдән конкрет функцияне импортларга мөмкинлек бирә.

Мәсәлән, элек без **random** модуленең бөтен эчтәлеген импортлый идек, әмма модульнең кечкенә өлешен генә кулланырга теләгәндә, импорт юлларын оптимальләштерергә кирәк. Кодыгызны түбәндәгечә үзгәртегез:

```
from random import choice

# Очраклы сайлау
selected_item = choice(['алма', 'банан', 'җиләк',
'карбыз'])
print(тавыш)
```

Хәзер без **random** модуленең бары тик **choice** функциясен генә импорт итәбез. Шул вакыттан башлап, безгә **random.choice** дип язу кирәкми. Без хәзер бары тик **choice** дип яза алабыз. **choice** функциясе турыдан-туры программага йөкләнгән. Бу систем ресурсларын саклый һәм кодның тизрәк эшләвен тәэмин итә ала!

Алга таба, **random.randint(a, b)** функциясен карагыз. Бу функция **a** һәм **b арасында очраклы сан генерацияли. Кодыгызны түбәндәгечә үзгәртегез:**

```
import random

сан = random.randint(1, 10)
print(сан)
```

Әлеге код 1 һәм 10 арасында очраклы сан генерацияләячәк.

Шулай ук, безгә **random.shuffle(x)** функциясен кертеп була, ул исемлекне очраклы тәртипкә салыр. Мәсәлән:

```
import random

исемлек = [1, 2, 3, 4, 5]
random.shuffle(исемлек)
print(исемлек)
```

Бу код исемлек элементларын очраклы тәртиптә чыгарачак.

```
import random

карточкалар = ["валет", "дама", "король"]
random.shuffle(карточкалар)
for карточка in карточкалар:
    print(карточка)
```

random.shuffle функциясе исемлекне урынында үзгәртә, башка функцияләрдән аермалы буларак, ул бернинди кыйммәт кайтармый. Аның урынына, ул **карточкалар** исемлеген турыдан-туры үзгәртә һәм аны шул ук исемлек эчендә shuffle итә. Кодыгызны берничә тапкыр эшләтеп карагыз, һәм һәр тапкырда исемлекнең тәртибе төрле булуын күрерсез.

Хәзер бездә очраклы мәгълүмат генерацияләү өчен өч ысул бар:

1. **random.choice(seq)**: Тәртипкә салынган исемлектән очраклы элементны сайлый.
2. **random.randint(a, b)**: a һәм b арасында очраклы сан генерацияли.
3. **random.shuffle(x)**: Исемлекне урынында очраклы тәртипкә салу өчен кулланыла.

Statistics

Python үз эченә төзелешле **statistics** модулен ала. Бу модуль саннар белән эш иткәндә төрле статистик исәп-хисапларны башкару өчен бик уңайлы. **mean** — әлеге модульнең иң файдалы функцияләренең берсе, һәм ул уртача мәгънәне табу өчен кулланыла.

code average.py исемле файл булдырыгыз. Аннары текст редакторында түбәндәге кодны языгыз:

```
import statistics

саннар = [100, 90]
уртача = statistics.mean(саннар)
print(уртача)
```

Бу кодта без **statistics** дип аталган модульне импорт итәбез. Ул мәгълүматны эшкәртү өчен күп кенә файдалы функцияләр тәкъдим итә. **mean** функциясе бирелгән саннар исемлеген ала һәм аларның уртача бәясен исәпли. Безнең очракта, ул **[100, 90]** исемлеген эшкәртәчәк һәм нәтиҗәдә бу саннарның уртача мәгънәсен табачак.

Программа эшләгәннән соң, терминалда 95.0 саны чыгарылачак — бу исемлектәге саннарның уртача бәясе.

statistics модуле Python программаларына төзелгән мәгълүматны тиз һәм җиңел эшкәртү мөмкинлеге бирә. Аның башка функцияләрен дә кулланып, мәгълүматларны тагын да тирәнрәк анализларга мөмкин. Әлеге мисал — **statistics.mean** функциясен куллануның гади, әмма нәтиҗәле үрнәге.

Командалар юлы аргументлары

sys — Python-да кулланучының командалар юлында аргументлар кертүен һәм аларны эшкәртүен тәэмин итүче модуль. **sys.argv** исемле функция модульнең иң файдалы элементларының берсе булып тора. Ул командалар юлында кертелгән барлык аргументларны исемлек рәвешендә саклый.

Башта sys.argv белән берничә гади мисал карап чыгыйк. **name.py** исемле файл булдырыгыз һәм текст редакторына түбәндәге кодны өстәгез:

```
import sys

print("сәлам, минем исемем", sys.argv[1])
```

Программа **sys.argv[1]** ярдәмендә командалар юлында кертелгән беренче аргументны укый. Мәсәлән, терминалда **python name.py Давид** дип язсагыз, программа сезгә:

```
сәлам, минем исемем Давид
```

дип җавап бирәчәк. Әгәр сез **sys.argv[0]** ни саклаганын тикшерсәгез, анда **name.py** исеме сакланачагын күрерсез, чөнки ул программа эшләтелгән файлның исемен күрсәтә.

Ләкин бу программаның бер җитешсезлеге бар. Әгәр кулланучы аргументлар кертмәсә, мәсәлән, **python name.py** дип кенә язса, программа **list index out of range** хата хәбәрен бирәчәк. Бу проблеманы түбәндәгечә чишеп була:

```
import sys

try:
    print("сәлам, минем исемем", sys.argv[1])
except IndexError:
    print("Артык аз аргументлар")
```

Мондый код кулланучыга хата турында мәгълүмат бирә һәм аны ничек төзәтергә икәнен күрсәтә. Шулай да, хаталарны эшкәртүне тагын да камилләштереп була:

```
import sys

if len(sys.argv) < 2:
    print("Артык аз аргументлар")
elif len(sys.argv) > 2:
    print("Артык күп аргументлар")
else:
```

```
    print("сәлам, минем исемем", sys.argv[1])
```

Бу код программаны күбрәк яки азрак аргументлар белән эшләтергә тырышканда, кулланучыга төгәлрәк күрсәтмәләр бирә. Хәзер без хата килеп чыккан очракта программа эшен туктатыр өчен **sys.exit** куллана алабыз:

```
import sys

if len(sys.argv) < 2:
    sys.exit("Артык аз аргументлар")
elif len(sys.argv) > 2:
    sys.exit("Артык күп аргументлар")

print("сәлам, минем исемем", sys.argv[1])
```

Бу версиядә, программа хатаны күрсәтеп тукталачак, һәм ахыргы юл (мәсәлән, **print**) башкарылмаячак. Шулай итеп, **sys.argv** кулланучыдан командалар юлы аша мәгълүмат алуны җиңеләйтә, ә **sys.exit** хата килеп чыккан очракта программа эшен туктатырга мөмкинлек бирә. Бу программаны төгәлрәк һәм ышанычлырак итә.

slice

slice — Python-да исемлекнең бер өлешен сайлап алырга мөмкинлек бирә торган механизм. Ул безгә исемлекнең кайсы элементыннан башларга һәм кайсы элементында туктарга кирәклеген күрсәтергә мөмкинлек бирә. Мәсәлән, **sys.argv** белән эшләгәндә, **slice** ярдәме белән командалар юлында кергән барлык аргументларны эшкәртү уңайлырак була ала.

Башта, исемлекнең бөтен элементларын бастырырга тырышкан гади кодны карагыз:

```
import sys

if len(sys.argv) < 2:
    sys.exit("Артык аз аргументлар")

for arg in sys.argv:
    print("сәлам, минем исемем", arg)
```

Бу кодны терминалда **python name.py Давид Картер Ронгшин** дип эшләтсәгез, программа, көтелгән исемнәрдән тыш, **name.py** файлының исемен дә бастырачак. Моның сәбәбе — **sys.argv** исемлегенең беренче элементы һәрвакыт программа файлының исеме була.

Шушы проблеманы чишү өчен, **slice** кулланып, исемлекнең башлангыч ноктасын үзгәртә алабыз. Кодыгызны түбәндәгечә үзгәртегез:

```
import sys

if len(sys.argv) < 2:
    sys.exit("Артык аз аргументлар")

for arg in sys.argv[1:]:
    print("сәлам, минем исемем", arg)
```

Монда **sys.argv[1:]** исемлекнең беренче элементын (файл исемен) төшереп калдыра һәм калган элементларны эшкәртә. Мәсәлән, терминалда **python name.py Давид Картер Ронгшин** дип язсагыз, программа түбәндәгечә эшләячәк:

```
сәлам, минем исемем Давид
сәлам, минем исемем Картер
сәлам, минем исемем Ронгшин
```

slice кулланып, программа логикасын гади, төгәл һәм җиңел аңлашыла торган итеп үзгәртә аласыз. Бу ысул командалар юлында кертелгән барлык файдалы мәгълүматларны дөрес итеп эшкәртү мөмкинлеге бирә.

Пакетлар

Python популярлыгының сәбәпләренең берсе — аның өченче як модульләр һәм пакетлар белән баетылуы. Әлеге модульләр программаларга өстәмә функциональлек өстәргә ярдәм итә. Пакетлар папка рәвешендә оештырылган, ә PyPI (Python Package Index) — аларның иң зур репозиториясе.

cowsay — популяр өченче як пакетларның берсе, ул кулланучы белән сөйләшә торган "сыер" функциясен тәкъдим итә. Пакетларны урнаштыру өчен Python-ның **pip** дигән пакет менеджеры кулланыла.

cowsay пакетын урнаштыру өчен терминалда түбәндәгечә команда языгыз:

```
pip install cowsay
```

Урнаштыру тәмамланганнан соң, сез бу пакетны үз кодыгызда куллана аласыз.

say.py исемле файл булдырыгыз һәм текст редакторына түбәндәге кодны өстәгез:

```
import cowsay
```

```
import sys

if len(sys.argv) == 2:
    cowsay.cow("сәлам, " + sys.argv[1])
```

Бу кодта программа иң беренче чиратта командалар юлында ике аргумент булу-булмавын тикшерә. Әгәр шарт үтәлсә, **`cowsay.cow`** функциясе сыер аша "сәлам" хәбәрен бастыра. Мәсәлән, терминалда **`python say.py Давид`** дип язсагыз, экранда сөтке "сәлам, Давид" дип "әйтәчәк".

Кодыгызны түбәндәгечә үзгәртегез, һәм хәзер T-Rex сөйләшәчәк:

```
import cowsay
import sys

if len(sys.argv) == 2:
    cowsay.trex("сәлам, " + sys.argv[1])
```

Бу кодта **`cowsay.trex`** функциясе кулланыла. Терминалда **`python say.py Картер`** дип язсагыз, T-Rex экранда "сәлам, Картер" дип чыгарачак.

APIлар

API (application program interface) — программалар арасында мәгълүмат һәм функциональлек алмашуны тәэмин итә торган интерфейс. Python-да **`requests`** модуле бу эшне җиңеләйтә, веб-браузер кебек эшләргә мөмкинлек бирә. Әйдәгез, **`requests`** кулланып, Apple iTunes API-сы белән эшләп карыйк.

Башта терминалда түбәндәгечә команда язып, **`requests`** пакетын урнаштырыгыз:

```
pip install requests
```

`itunes.py` файлын булдыру өчен, терминалда **`itunes.py`** исемле файл булдырыгыз һәм түбәндәге кодны өстәгез:

```
import requests
import sys

if len(sys.argv) != 2:
    sys.exit()
```

```
response =
requests.get("https://itunes.apple.com/search?entity=so
ng&limit=1&term=" + sys.argv[1])
print(response.json())
```

Бу программа кулланучының командалар юлында тапшырган терминен кулланып, iTunes API-сына җыр эзләү соравы җибәрә. Мәсәлән, **python itunes.py weezer** дип язсагыз, iTunes тарафыннан JSON форматына әйләндерелгән җавапны алачаксыз. **response.json()** мәгълүматны Python сүзлегенә әйләндерә.

Чыгыш күп мәгълүматны үз эченә ала һәм катлаулы булырга мөмкин. JSON форматын аңлаешлырак итү өчен, **json** модулен кулланып, кодыгызны үзгәртегез:

```
import json
import requests
import sys

if len(sys.argv) != 2:
    sys.exit()

response =
requests.get("https://itunes.apple.com/search?entity=so
ng&limit=1&term=" + sys.argv[1])
print(json.dumps(response.json(), indent=2))
```

json.dumps функциясе JSON файлын укылырлык итеп чыгара. Мәсәлән, **python itunes.py weezer** эшләткәндә, сез ясалма рәвештә форматланган JSON файлын күрерсез. Нәтиҗәдә, сез **results** исемле сүзлекне һәм аның эчендәге ачкычларны күрәчәксез.

JSON җавабыннан **trackName** кыйммәтен генә чыгару өчен, кодыгызны түбәндәгечә үзгәртегез:

```
import json
import requests
import sys

if len(sys.argv) != 2:
    sys.exit()

response =
requests.get("https://itunes.apple.com/search?entity=so
ng&limit=50&term=" + sys.argv[1])
```

```
o = response.json()
for result in o["results"]:
    print(result["trackName"])
```

Бу кодта:

- **`o = response.json()`** — JSON җавабын **`o`** исемле сүзлеккә саклый.
- **`o["results"]`** — җаваптагы **`results`** исемле ачкыч эчендәге мәгълүматка мөрәҗәгать итә.
- **`for result in o["results"]`** — җаваптагы һәр нәтиҗәне эшкәртә.
- **`result["trackName"]`** — һәр нәтиҗәнең җыр исемен бастыра.

Мәсәлән, **`python itunes.py weezer`** дип эшләткәндә, сез **`trackName`** кыйммәтләренең исемлеген күрәчәксез.

`requests` модуле һәм **`json`** турында тулырак мәгълүматны аларның документациясеннән алырга мөмкин. Бу инструментлар белән танышу API-ларны эффектив куллану мөмкинлеген арттыра.

Үзегезнең Модульләрегезне Булдыру

Сез Python программисты буларак үзегезнең модульләрегезне булдыра аласыз! Карагыз, кайбер очракларда сез код өлешләрен кабатлап кулланырга яки башкалар белән уртаклашырга теләрсез!
Бу курс дәвамында без күп тапкыр "сәлам" дип әйтә торган кодлар яздык. Әйдәгез, "сәлам" һәм "хәйерле" дип әйтергә мөмкинлек бирә торган модуль булдырыйк. **`sayings.py`** исемле файл булдырыгыз. Текст редакторында түбәндәге кодны языгыз:

```
def сәламләү(исем):
    print(f"сәлам, {исем}")

def хәйерле(исем):
    print(f"хәйерле, {исем}")
```

Бу код үзе генә эшләми. Ләкин программист бу модульне үзенең программасына импорт итсә, өстәге функцияләрне куллана алачак.

Кодыгызны бу модульне кулланып ничек файдаланырга мөмкин икәнен күрсәтү өчен, **`say.py`** исемле файл булдырыгыз. Яңа файлда түбәндәге кодны языгыз:

```
import sys
from sayings import хәйерле

if len(sys.argv) == 2:
    хәйерле(sys.argv[1])
```

Бу код **sayings** модуленнән **хәйерле** функциясен импорт итә. Әгәр кулланучы командалар юлында кимендә ике аргумент кертә икән, ул "хәйерле" дип, командалар юлында керткән исем белән әйтәчәк.

2.6. Берәмлек тестлары

Әлегә кадәр сез, мөгаен, үз кодыгызны **`print`** операторларын кулланып тикшереп карагансыз. Ләкин сәнәгатьтә иң киң таралган ысул — программаларны тестлау өчен махсус код язу.

Башта **`calculator.py`** исемле файл булдырыгыз. Әгәр бу файлны алдагы дәресләрдә язган булсагыз, аны яңадан булдыру кирәк түгел. Текст редакторында, сезнең код түбәндәгечә булырга тиеш:

```
def main():
    x = int(input("What's x? "))
    print("x squared is", square(x))


def square(n):
    return n * n


if __name__ == "__main__":
    main()
```

Бу кодны эшләтеп карап була, мәсәлән, 2 санын кертеп. Ләкин, программа дөрес эшлиме икәнлеген автоматик рәвештә тикшерү өчен тест язу кирәклеге турында уйлап карагыз.

Конвенция буенча, яңа тест программасы язу өчен **`test_calculator.py`** исемле файл булдырыгыз һәм текст редакторында түбәндәге кодны кертегез:

```
from calculator import square


def main():
    test_square()


def test_square():
    if square(2) != 4:
        print("2 squared was not 4")
    if square(3) != 9:
        print("3 squared was not 9")


if __name__ == "__main__":
    main()
```

Бу кодның беренче юлында **calculator.py** файлыннан **square** функциясе импорт ителә. Конвенциягә туры китереп, без **test_square** исемле функция ясыйбыз, ул кайбер шартларны тикшерә.

Терминалда **python test_calculator.py** командасын язып карагыз. Әгәр тестлар уңышлы үтә, бернинди хәбәр дә күренмәячәк. Бу код дөрес эшли дигәнне аңлата. Әмма шуны да истә тотыгыз: тест функциясе кайбер "почмак очракларын" тапмаган булырга мөмкин, алар хаталарга китерергә мөмкин.

Әлеге код хәзергә бары тик ике шартны гына тикшерә. Әгәр тестларны күбрәк шартларны колачларлык итәргә теләсәгез, аларның коды артык катлаулы булып китәргә мөмкин. Бу очракта, тестлау мөмкинлекләрен киңәйтү өчен автоматлаштырылган тестлау инструментлары, мәсәлән, **pytest** китапханәсе, кулланырга мөмкин.

Мәсәлән, pytest кулланып тестларыгыз түбәндәгечә булырга мөмкин:

```
import pytest
from calculator import square

def test_square():
    assert square(2) == 4
    assert square(3) == 9
    assert square(-1) == 1
    assert square(0) == 0
```

Бу кодны башкару өчен, терминалда **pytest** командасын языгыз. Pytest тестларны автоматик рәвештә ачыклый һәм нәтиҗәләрне күрсәтә.

assert

Pythonның **assert** командасы безгә программага билгеле бер шартның дөрес булуын тикшерергә мөмкинлек бирә. Тест кодында аны түбәндәгечә кулланырга мөмкин:

```
from calculator import square


def main():
    test_square()


def test_square():
    assert square(2) == 4
    assert square(3) == 9
```

```
if __name__ == "__main__":
    main()
```

Бу кодта без ачык итеп **square(2)** һәм **square(3)** функцияләренең кайсы нәтиҗәләргә тигез булырга тиешлеген күрсәтәбез. Бу якын килү тест юлларының санын дүрттән икегә киметә.

Әгәр без калькулятор кодын махсус хаталы итеп үзгәртергә теләсәк, аны түбәндәгечә үзгәртеп була:

```
def main():
    x = int(input("What's x? "))
    print("x squared is", square(x))


def square(n):
    return n + n


if __name__ == "__main__":
    main()
```

Монда ***** операторын **+** белән алыштырдык. Хәзер, **python test_square.py** командасын эшләткәндә, программа **AssertionError** чыгарачак. Бу шартларның берсе үтәлмәгәнлеген күрсәтә.

Әгәр кулланучыларга хата турында тулырак мәгълүмат бирергә теләсәк, кодны түбәндәгечә үзгәртеп була:

```
from calculator import square


def main():
    test_square()


def test_square():
    try:
        assert square(2) == 4
    except AssertionError:
        print("2 squared is not 4")
    try:
        assert square(3) == 9
```

```
    except AssertionError:
        print("3 squared is not 9")
    try:
        assert square(-2) == 4
    except AssertionError:
        print("-2 squared is not 4")
    try:
        assert square(-3) == 9
    except AssertionError:
        print("-3 squared is not 9")
    try:
        assert square(0) == 0
    except AssertionError:
        print("0 squared is not 0")


if __name__ == "__main__":
    main()
```

Бу код күп очракларны тестлый һәм берничә хата турында хәбәр бирә. Шулай да, ул барлык хаталарны күрсәтә алмаска мөмкин. Бу тестлауның төп әһәмиятен искә төшерә: программа хаталарын табу өчен мөмкин кадәр күбрәк шартларны тикшерергә кирәк.

Шулай ук, код зуррак проблема күрсәтә: тестлар артык озын һәм катлаулы була ала. Кодны җиңел һәм төгәл итеп саклау өчен, автоматлаштырылган тестлау инструментларын куллану тәкъдим ителә, мәсәлән, **pytest** яки **unittest** модулен. Бу инструментлар тест структурасын оптимальләштерергә һәм кодны тәртипкә салырга ярдәм итә.

pytest

pytest — бу программаларыгызны берәмлек тестлау өчен кулланыла торган өченче як китапханә. Гади сүзләр белән әйткәндә, ул функцияләрегезнең дөрес эшләвен тикшерергә мөмкинлек бирә.

pytestны куллану өчен, консольдә **`pip install pytest`** командасын языгыз. Аннары, тест файлын (**`test_calculator.py`**) түбәндәгечә үзгәртегез:

```
from calculator import square


def test_assert():
    assert square(2) == 4
```

```
    assert square(3) == 9
    assert square(-2) == 4
    assert square(-3) == 9
    assert square(0) == 0
```

Бу код барлык тест шартларын ассызыклый. pytest тест нәтиҗәләрен аңлаешлырак форматта күрсәтә һәм программаны турыдан-туры аның аша эшләтә.

Консольдә **pytest test_calculator.py** командасын языгыз. Әгәр кодта хата бар икән, экранда кызыл F билгесе күренәчәк, бу тестларның уңышсыз узуын аңлата. Ә кызыл E хатасы сезнең **calculator.py** файлыгызда җитди хата булуын күрсәтә.

Тест нәтиҗәләренә карап, **calculator.py** кодын төзәтәбез:

```
def main():
    x = int(input("What's x? "))
    print("x squared is", square(x))


def square(n):
    return n * n


if __name__ == "__main__":
    main()
```

***** операторын кире кайтарып, программа эшли торган хәлгә китердек. Хәзер **pytest test_calculator.py** командасын яңадан эшләткәндә, хаталар чыкмый.

pytest беренче хатадан соң туктый, бу җитешсезлек. Шуңа күрә, кодны махсус җимереп карыйбыз:

```
def main():
    x = int(input("What's x? "))
    print("x squared is", square(x))


def square(n):
    return n + n


if __name__ == "__main__":
```

```
    main()
```

***** операторын **+** белән алыштырып, программа эшләмә торган хәлгә китердек.

Тест кодын оптимальләштерү өчен, аны түбәндәгечә үзгәртәбез:

```
from calculator import square


def test_positive():
    assert square(2) == 4
    assert square(3) == 9


def test_negative():
    assert square(-2) == 4
    assert square(-3) == 9


def test_zero():
    assert square(0) == 0
```

Бер үк биш тестны өч функциягә бүлдек. pytest кебек тестлау каркаслары һәр функцияне аерым эшләтә, хәтта берсендә хата чыкса да.

Хәзер **pytest test_calculator.py** командасын яңадан эшләткәндә, күбрәк хаталар күренергә тиеш. Бу чыганак проблемаларын табуда ярдәм итә.

Тест кодын төгәлләгәннән соң, **calculator.py** кодын тулы эшли торган хәлгә кайтарыгыз:

```
def main():
    x = int(input("What's x? "))
    print("x squared is", square(x))


def square(n):
    return n * n


if __name__ == "__main__":
    main()
```

pytest test_calculator.py командасын яңадан эшләтегез, хаталар чыкмый.

Программа искәрмәләрне ничек эшкәртүен тикшерү өчен, тест кодын түбәндәгечә үзгәртәбез:

```
import pytest

from calculator import square


def test_positive():
    assert square(2) == 4
    assert square(3) == 9


def test_negative():
    assert square(-2) == 4
    assert square(-3) == 9


def test_zero():
    assert square(0) == 0


def test_str():
    with pytest.raises(TypeError):
        square("cat")
```

Монда **assert** урынына **pytest** китапханәсендәге **raises** функциясен кулланабыз. Бу тест көтелгән хатаны тикшерә. Программа башына **import pytest** өстәп, **pytest.raises** ярдәмендә хата төрен билгеләдек.

pytest test_calculator.py командасын яңадан эшләтеп карагыз. Әгәр барлык тестлар уңышлы үтә икән, хаталар чыкмый. Бу сезнең кодның тулысынча дөрес эшләвен аңлата!

Йомгаклап әйткәндә, сез код яза торганчы, күптөрле тест шартларын билгеләү сезнең өстегездә!

pytest турында күбрәк мәгълүмат алу өчен **pytest** документациясен укыгыз.

Юлларны тестлау

Вакытка кире кайтып, түбәндәге hello.py кодын карыйк:

```
def main():
    name = input("What's your name? ")
    hello(name)


def hello(to="world"):
    print("hello,", to)


if __name__ == "__main__":
    main()
```

Әгәр hello функциясенең нәтиҗәсен тестларга теләсәк, түбәндәге test_hello.py кодын карыйк:

```
from hello import hello


def test_hello():
    assert hello("David") == "hello, David"
    assert hello() == "hello, world"
```

Бу кодны караганда, тестлау ысулы эшләр дип уйлыйсызмы? Ни өчен бу тест дөрес эшләми? Сәбәбе шунда: hello функциясе hello.py файлы эчендә текстны чыгара, ләкин ул бернинди дә кыйммәт кайтармый! Тестның эшләве өчен функциядән кыйммәтне кайтару кирәк.

hello.py файлын түбәндәгечә үзгәртик:

```
def main():
    name = input("What's your name? ")
    print(hello(name))


def hello(to="world"):
    return f"hello, {to}"


if __name__ == "__main__":
    main()
```

Бу кодта hello функциясе хәзер текстны чыгармый, ә кыйммәт кайтара. Мондый үзгәрешләр безгә pytest кулланып, hello функциясен дөрес итеп тестларга мөмкинлек бирә.

Терминалда pytest test_hello.py командасын эшләткәндә, барлык тестлар үтәр.

Тест очракларын аерым функцияләргә бүлеп алсак, алар тагын да җайлырак булыр:

```
from hello import hello


def test_default():
    assert hello() == "hello, world"


def test_argument():
    assert hello("David") == "hello, David"
```

Бу код тестларны аерым функцияләргә бүлә. Әгәр бер тестта хата чыкса да, калганнары барыбер үтәчәк. Бу тест структурасы кодны тикшерүне җайлы һәм төгәл итә.

Тестларны директорияләрдә оештыру

Берәмлек тестлау кодны күп тестлар белән тикшерү өчен бик гади ысул, һәм сез бер команда ярдәмендә бөтен директориядәге тестларны эшләтеп җибәрә аласыз.

Башта, терминалда **`mkdir test`** командасын языгыз. Бу test исемле директория булдырачак.

Аннары, бу директория эчендә тест файллары булдырырга кирәк. Мәсәлән, **`test/test_hello.py`** исемле файл булдырыгыз. Бу команда test/ директориясе эчендә test_hello.py файлын булдыруны күрсәтә.

Текст редакторында файлны түбәндәгечә үзгәртегез:

```
from hello import hello


def test_default():
    assert hello() == "hello, world"


def test_argument():
    assert hello("David") == "hello, David"
```

Бу код, алдагыча, hello функциясен тестларга мөмкинлек бирә.

pytest бөтен директориядәге тестларны автоматик рәвештә эшләтергә мөмкин. Ләкин моның өчен директориядә **`__init__.py`** файлы булырга тиеш. **`test/__init__.py`** исемле файл булдырыгыз. Әлеге файлны буш калдырырга мөмкин. Ул директорияне Python модуле итеп билгели һәм pytestка бу директориядә тестлар барлыгын аңлата.

Хәзер, терминалда **`pytest test`** командасын языгыз. Бу команда test директориясендәге барлык тест файлларын табып эшләтәчәк.

Бу процесс сезгә барлык тестларны тиз һәм уңайлы итеп тикшерергә мөмкинлек бирә. Кодыгызда үзгәрешләр керткәндә, аларның дөреслеген шушы структура ярдәмендә җиңел тикшереп була.

2.7. Файллар белән эшләү

Хәзерге вакытка кадәр без язган барлык программалар мәгълүматны бары тик оператив хәтердә саклады. Башкача әйткәндә, программа туктатылганнан соң, кулланучыдан җыелган яки программа тарафыннан эшләнгән барлык мәгълүмат юкка чыга.

Файллар белән эшләү мөмкинлеге программага мәгълүматны файлга сакларга һәм кире укырга ярдәм итә. Башта, **names.py** исемле файл булдырыгыз, түбәндәге кодны өстәгез:

```
name = input("What's your name?" )
print(f"hello, {name}")
```

Бу код кулланучыдан исем сорый һәм экранда "hello" дип каршы ала. Нәтиҗә көтелгәнчә чыга.

Ләкин, берничә исемне сакларга теләсәк, моңа ничек ирешергә мөмкин? Исегездә тотыгыз, исемлек (list) — бер үк үзгәрешле эчендә берничә кыйммәтне сакларга мөмкинлек бирә торган мәгълүмат структурасы.

```
names = []

for _ in range(3):
    name = input("What's your name?" )
    names.append(name)
```

Бу код кулланучыга өч тапкыр исем кертү мөмкинлеген бирә. **append** методы һәр кертелгән исемне **names** исемлегенә өсти.

Кодны түбәндәгечә кыскарак итеп язырга мөмкин:

```
names = []

for _ in range(3):
    names.append(input("What's your name?" ))
```

Бу код шул ук нәтиҗәне бирә.

Хәзер, кертелгән исемнәрне алфавит тәртибендә бастырырга мөмкинлек бирик:

```
names = []

for _ in range(3):
    names.append(input("What's your name?" ))
```

```
for name in sorted(names):
    print(f"hello, {name}")
```

Бу программа эшләтелгәндә, исемнәр алфавит тәртибендә экранда күрсәтеләчәк. Ләкин, программа туктагач, барлык мәгълүмат юкка чыгачак. Файллар белән эшләү программаның бу мәгълүматны саклап калырга һәм соңрак яңадан кулланырга мөмкинлек бирә.

sorted функциясе турында күбрәк мәгълүматны Pythonның документациясеннән укып белә аласыз.

ачу

open — Python'ның эченә кергән функциясе, ул файлларны ачу һәм алар белән эшләү мөмкинлеге бирә. **open** функциясе файлларны уку яки язу режимында ачарга мөмкинлек бирә.

Программаның файл белән эшләү мөмкинлеген күрсәтү өчен, түбәндәге кодны языгыз:

```
name = input("What's your name? ")

file = open("names.txt", "w")
file.write(name)
file.close()
```

open функциясе **names.txt** исемле файлны язу режимында (**w**) ача. Код ачыкланган файлны **file** исемле үзгәрешлегә билгели.
file.write(name) юлы кулланучыдан алынган исемне текст файлына яза. Соңгы юлда файл ябыла.

Кодны сынап карагыз: терминалда **python names.py** языгыз һәм исем керегез. Бу исем **names.txt** файлына сакланачак. Ләкин, программа берничә тапкыр эшләтелсә, **names.txt** файлы тулысынча яңадан язылуын күрерсез.

Идеаль рәвештә, һәр исемне текст файлына өстәргә мөмкинлек булырга тиеш. Моның өчен терминалда **rm names.txt** командасы белән файлны бетерегез һәм кодны түбәндәгечә үзгәртегез:

```
name = input("What's your name? ")

file = open("names.txt", "a")
file.write(name)
```

```
file.close()
```

Бу кодта **w** урынына **a** (append) кулланыла, ул файлга язу урынына мәгълүматны өсти. Программаны берничә тапкыр эшләтеп, исемнәр файлга өстәләчәген күрерсез. Ләкин бу очракта тагын бер проблема килеп чыга: исемнәр бер-берсенә кушылып языла, арада буш урын булмый.

Мәсьәләне чишү өчен, текст файлын яңадан бетерегез (**rm names.txt**) һәм кодны түбәндәгечә үзгәртегез:

```
name = input("What's your name? ")

file = open("names.txt", "a")
file.write(f"{name}\n")
file.close()
```

file.write юлы һәр исемнең ахырында юл өзеге (**\n**) өсти, бу аларны аерым юлларга яза. Хәзер программа дөрес эшләячәк.

Ләкин, бу кодны тагын да яхшыртырга мөмкин. Файлны ябуны оныту бик җиңел, шуңа күрә **with** операторы куллану хәерле:

```
name = input("What's your name? ")

with open("names.txt", "a") as file:
    file.write(f"{name}\n")
```

with операторы файлны автоматик рәвештә яба, бу программаны уңайлырак һәм ышанычлырак итә.

with

with ачкыч сүзе файл белән эшләгәндә аны автоматик рәвештә ябарга мөмкинлек бирә. Бу кодны кыскартып, эшләү процессын ышанычлырак итә.

Файлга язу өчен кодны түбәндәгечә үзгәртегез:

```
name = input("What's your name? ")

with open("names.txt", "a") as file:
    file.write(f"{name}\n")
```

with эчендәге юлның керемгә (indent) ия булуын күрергә мөмкин. Бу язма режимында (append) файлга язу өчен.

Әгәр файлдан укырга теләсәк, кодны түбәндәгечә үзгәртәбез:

```
with open("names.txt", "r") as file:
    lines = file.readlines()

for line in lines:
    print("hello,", line)
```

readlines функциясе файлдагы барлык юлларны укып, **lines** исемле исемлеккә саклый. Ләкин, программа эшләгәндә, һәр юлның ахырында артык юл өзеге күренәчәк.

Бу мәсьәләне төзәтү өчен, **rstrip** функциясен кулланыгыз:

```
with open("names.txt", "r") as file:
    lines = file.readlines()

for line in lines:
    print("hello,", line.rstrip())
```

rstrip функциясе юлның ахырындагы артык **\n** билгеләрен бетерә. Әмма бу кодны тагын да гади итәргә мөмкин:

```
with open("names.txt", "r") as file:
    for line in file:
        print("hello,", line.rstrip())
```

Бу версия бер үк эшне башкара, ләкин артык исемлекләр кулланмый.

Файлдан укыганда, исемнәрне тәртипкә салырга теләсәгез, түбәндәгечә эшләгез:

```
names = []

with open("names.txt") as file:
    for line in file:
        names.append(line.rstrip())

for name in sorted(names):
    print(f"hello, {name}")
```

Бу код түбәндәгечә эшли:

1. **names** исемле буш исемлек булдырыла.
2. Файлдагы һәр исем укыла һәм **rstrip** функциясе ярдәмендә артык билгеләрдән арындырыла.
3. Исем **names** исемлегенә өстәлә.
4. **sorted** функциясе исемнәрне тәртипкә сала.

5. Тәртипләнгән исемнәр экранга бастырыла.

Бу кодны эшләтеп карагыз, һәм исемнәрнең дөрес тәртиптә күрсәтелүен күрерсез. Шул рәвешле, программа мәгълүматны саклый, укый һәм эшкәртә ала.

CSV

CSV (Comma-Separated Values) — кыйммәтләрне өтер белән аерып язылган форматны аңлата. Бу форматны Python-да укып эшкәртү һәм язу өчен күп мөмкинлекләр бар.

CSV файлны уку ученә**students.csv** исемле файл булдырыгыз һәм түбәндәге мәгълүматларны файлга өстәгез:

```
Hermione,Gryffindor
Harry,Gryffindor
Ron,Gryffindor
Draco,Slytherin
```

Аннары **students.py** исемле файл булдырыгыз һәм шушы кодны өстәгез:

```
with open("students.csv") as file:
    for line in file:
        row = line.rstrip().split(",")
        print(f"{row[0]} is in {row[1]}")
```

rstrip функциясе юл ахырындагы артык символларны (мәсәлән, **\n**) бетерә, ә **split** кыйммәтләрне аера. **row[0]** — беренче багана, **row[1]** — икенче багана. Бу код һәр юлны аерым эшкәртә.

Кодны җиңелрәк аңларлык итеп язырга мөмкин:

```
with open("students.csv") as file:
    for line in file:
        name, house = line.rstrip().split(",")
        print(f"{name} is in {house}")
```

Мәгълүматны тәртипкә салу Кулланучының мәгълүматларын тәртипкә китерү өчен исемлек кулланырга була:

```
students = []

with open("students.csv") as file:
    for line in file:
        name, house = line.rstrip().split(",")
```

```
        students.append(f"{name} is in {house}")

for student in sorted(students):
    print(student)
```

Бу код һәр юлны тәртипкә салынган исемлеккә өсти һәм аннары экранга бастыра.

Сүзлекләр куллану Python сүзлекләре мәгълүматны яхшырак оештырырга ярдәм итә:

```
students = []

with open("students.csv") as file:
    for line in file:
        name, house = line.rstrip().split(",")
        students.append({"name": name, "house": house})

for student in students:
    print(f"{student['name']} is in
{student['house']}")
```

Бу код һәр студентны сүзлек итеп саклый, анда **`name`** һәм **`house`** ачкычлары кулланыла.

`csv.DictReader` куллану **`csv`** модуле мәгълүматны җиңелрәк эшкәртү өчен **`DictReader`** тәкъдим итә:

```
import csv

students = []

with open("students.csv") as file:
    reader = csv.DictReader(file)
    for row in reader:
        students.append({"name": row["name"], "house":
row["house"]})

for student in sorted(students, key=lambda student:
student["name"]):
    print(f"{student['name']} is in
{student['house']}")
```

`DictReader` һәр юлны автомат рәвештә сүзлеккә әйләндерә. Бу саклык һәм сыгылмалылык өсти.

CSV файлга язу Файлга язу өчен башта иске файлны бетерегез:

```
rm students.csv
```

Шуннан соң, students.py файлын түбәндәгечә үзгәртегез:

```
import csv

name = input("What's your name? ")
home = input("Where's your home? ")

with open("students.csv", "a") as file:
    writer = csv.DictWriter(file, fieldnames=["name",
"home"])
    writer.writerow({"name": name, "home": home})
```

Бу код:

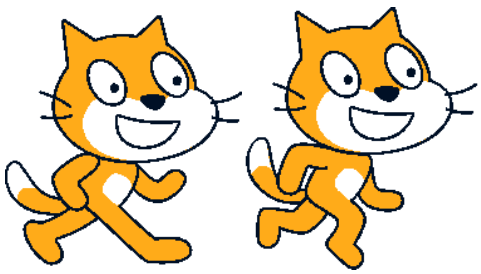
- **`DictWriter`** куллана, ул язуларны автоматик рәвештә CSV форматында саклый.
- **`fieldnames`** параметры кырга исемнәр бирә.
- **`writerow`** функциясе бер сүзлекне яза.

Бинар файллар һәм PIL

Бүген без бинар файллар турында сөйләшәчәкбез. Бинар файллар — берләр һәм нульләр җыелмасыннан торган мәгълүмат. Алар теләсә нинди мәгълүматны, шул исәптән музыка һәм рәсемнәрне дә саклый ала.

Python өчен PIL (Pillow) китапханәсе рәсем файллары белән эшләү өчен киң кулланыла. Мәсәлән, анимацияле GIF — популяр рәсем форматы, анда берничә рәсем бар, алар чиратлап уйнатылып, гади анимация яки видео эффекты тудыра.

Әйдәгез, costume1.gif һәм costume2.gif дип аталган ике рәсем файлы бар дип күз алдына китерик, аларны анимацияле GIF итеп берләштерергә телибез.



Әлеге файлларны сез түбәндәге сылтамалар аша йөкли аласыз:

https://s.kpfu.ru/3f

https://s.kpfu.ru/3g

costumes.py исемле файл булдырыгыз һәм шушы кодны өстәгез:

```
import sys
from PIL import Image

images = []

for arg in sys.argv[1:]:
    image = Image.open(arg)
    images.append(image)

images[0].save(
    "costumes.gif", save_all=True,
append_images=[images[1]], duration=200, loop=0
)
```

Терминалда түбәндәге команданы языгыз:

```
python costumes.py costume1.gif costume2.gif
```

Бу программа costumes.gif исемле яңа анимацияле GIF файлы тудырачак. Аннары, анимацияне карау өчен түбәндәге команданы кертергә кирәк:

```
code costumes.gif
```

2.8. Регуляр аңлатмалар

Регуляр аңлатмалар яки "регекслар" безгә код эчендә үрнәкләрне тикшерергә мөмкинлек бирә. Мәсәлән, без электрон почта адресының дөрес форматта булуын тикшерергә теләр идек. Регуляр аңлатмалар шундый ысул белән экспрессияләрне тикшерергә мөмкинлек бирә.

Башлау өчен, **validate.py** исемле файл булдырыгыз. Аннан соң, текст мөхәррирендә түбәндәге кодны языгыз:

```
email = input("Электрон почтагызны языгыз? ").strip()

if "@" in email:
    print("Дөрес")
else:
    print("Дөрес түгел")
```

strip методы керүнең башындагы яки ахырындагы бушлыкны бетерә. Бу программаны эшләтеп җибәргәндә, @ символы керткәннән соң, программа керүне дөрес дип санаячак.

Ләкин, сез, мәсәлән, @@ гына кертергә мөмкин, һәм программа керүне дөрес дип кабул итәр иде. Электрон почта адресын кимендә бер @ һәм ниндидер урында . булуын тикшерергә мөмкин. Кодны түбәндәгечә үзгәртегез:

```
email = input("Электрон почтагызны языгыз? ").strip()

if "@" in email and "." in email:
    print("Дөрес")
else:
    print("Дөрес түгел")
```

Бу көтелгәнчә эшли, ләкин кулланучы гади генә @. кертергә мөмкин, һәм программа аны дөрес дип кабул итәр иде.

Программаның логикасын түбәндәгечә яхшыртырга мөмкин:

```
email = input("Электрон почтагызны языгыз? ").strip()

username, domain = email.split("@")

if username and "." in domain:
    print("Дөрес")
else:
    print("Дөрес түгел")
```

Бу код **username** булуын һәм домен өлешендә **.** символы барлыгын тикшерә. Программаны эшләткәндә, сез керткән стандарт электрон почта адресы дөрес дип саналачак. Мәсәлән, **malan@kpfu** адресы дөрес түгел дип бәяләнәчәк.

Кодны тагын да төгәлрәк итү өчен түбәндәгечә үзгәртегез:

```
email = input("Электрон почтагызны языгыз? ").strip()

username, domain = email.split("@")

if username and domain.endswith(".ru"):
    print("Дөрес")
else:
    print("Дөрес түгел")
```

endswith методы доменның **.ru** белән тәмамлануын тикшерә. Шулай да, начар ниятле кулланучы **malan@.ru** кертеп, аны дөрес дип кабул иттерә ала.

Чыннан да, бу кодны камилләштереп дәвам итәргә мөмкин. Ләкин, Python-ның **re** исемле китапханәсе регуляр аңлатмалар белән эшләү өчен күп функцияләр тәкъдим итә.

re китапханәсе белән танышу

re китапханәсендәге иң күп кулланыла торган функцияләрнең берсе — **search**.

search функциясенең сигнатурасы: **re.search(pattern, string, flags=0)**. Бу функцияне кулланып, кодны түбәндәгечә үзгәртергә мөмкин:

```
import re

email = input("Электрон почтагызны языгыз? ").strip()

if re.search("@", email):
    print("Дөрес")
else:
    print("Дөрес түгел")
```

Бу программа функциональлекне арттырмый, ә бары тик регуляр аңлатмалар куллану юнәлешендә бер адым ясый.

Программаны тагын да камилләштерү өчен, регуляр аңлатмалар билгеле символларны кулланып тикшерү мөмкинлеген киңәйтергә мөмкин. Түбәндәге символлар регуляр аңлатмаларда киң кулланыла:

- `.` — теләсә нинди символ (яңа юлдан башка)
- `*` — 0 яки күбрәк кабатлау
- `+` — 1 яки күбрәк кабатлау
- `?` — 0 яки 1 кабатлау
- `{m}` — m тапкыр кабатлау
- `{m,n}` — m-нче тапкырдан башлап n тапкыр кабатлау

Бу символларны кулланып, кодны түбәндәгечә үзгәртергә мөмкин:

```
import re

email = input("Электрон почтагызны языгыз? ").strip()

if re.search(".+@.+", email):
    print("Дөрес")
else:
    print("Дөрес түгел")
```

Бу код `.+` ярдәмендә адреснының `@` символының сул ягында һәм уң ягында берәр нәрсә булуын тикшерә. Мәсәлән, `malan@` керткәндә, программа аны дөрес түгел дип бәяләячәк.

`.+@.+` регуляр аңлатмасын түбәндәгечә аңлатырга мөмкин:

- **q1** — кулланучы тарафыннан теләсә нинди символның 1 яки күбрәк кабатлануын аңлата.
- **q2** — `@` символына туры килә.
- **q3** — уң яктагы теләсә нинди символларны тикшерә.

Кодны тагын да камилләштереп:

```
import re

email = input("Электрон почтагызны языгыз? ").strip()

if re.search(".+@.+\.ru", email):
    print("Дөрес")
else:
    print("Дөрес түгел")
```

Бу код **.ru** белән тәмамланган адресларны дөрес дип бәяли. Ләкин, кулланучы **malan@kpfu?edu** дип керткәндә дә, программа аны дөрес дип кабул итәчәк, чөнки **.** теләсә нинди символны аңлата.

Escape символын кулланып, кодны төзәтергә мөмкин:

```
import re

email = input("Электрон почтагызны языгыз? ").strip()

if re.search(r".+@.+\.ru", email):
    print("Дөрес")
else:
    print("Дөрес түгел")
```

**** символы **.** ны гади символ итеп карарга мәҗбүр итә. Хәзер **malan@kpfu.ru** дөрес дип саналачак, ә **malan@kpfu?edu** дөрес түгел дип бәяләнәчәк.

Python-да raw string (чиста юллар) куллану шулай ук махсус символларны үзгәртмичә калдырырга мөмкинлек бирә. Кодны түбәндәгечә камилләштерергә мөмкин:

```
import re

email = input("Электрон почтагызны языгыз? ").strip()

if re.search(r"^.+@.+\.ru$", email):
    print("Дөрес")
else:
    print("Дөрес түгел")
```

^ — юлның башын, **$** — юлның ахырын аңлата. Бу код **.ru** белән тәмамланган, дөрес форматтагы электрон почта адресларын тикшерәчәк.

Тулырак регуляр аңлатмалар

Шулай да, кулланучылар безгә проблемалар тудыруны дәвам итә ала! Мәсәлән, My email address is **malan@kpfu.ru**. кебек җөмлә кертергә мөмкин һәм бу бөтен җөмлә дөрес дип саналачак. Программалауны тагын да төгәлләштерергә мөмкин.

Валидациядә кулланыла торган махсус символлар:

- ^ — юлның башы белән туры килә.

- **$** — юлның ахыры белән яки юлның ахырындагы яңа юлдан алда туры килә.

Кодны бу символларны кулланып түбәндәгечә үзгәртергә мөмкин:

```
import re

email = input("Электрон почтагызны языгыз? ").strip()

if re.search(r"^.+@.+\.ru$", email):
    print("Дөрес")
else:
    print("Дөрес түгел")
```

Бу регуляр аңлатма тикшерелгән юлның башыннан ахырына кадәр туры килүен тәэмин итә. Мәсәлән, My email address is **malan@kpfu.ru**. кертсәгез, ул дөрес түгел дип саналачак.

Без тагын да яхшырак эшли алабыз! Хәзер юл башында кулланучы исемен, `@` символын һәм ахырда домен исемен тикшерәбез, ләкин кулланучы теләсә кайсы җирдә берничә `@` символы кертә ала! Мәсәлән, malan@@@kpfu.ru дөрес дип санала.

Моның өчен сүзлеккә түбәндәге символлар өстәлә:

- **[]** — символлар җыелмасы.
- **[^]** — җыелманың кире кагылышы.

Бу яңа мөмкинлекләрне кулланып, кодны түбәндәгечә үзгәртә алабыз:

```
import re

email = input("Электрон почтагызны языгыз? ").strip()

if re.search(r"^[^@]+@[^@]+\.ru$", email):
    print("Дөрес")
else:
    print("Дөрес түгел")
```

`[^@]+` — `@` символыннан башка теләсә нинди символлар. `@` туры символ булып тикшерелә, аннан соң **`[^@]+\.ru`** домен өлешен билгели. Мәсәлән, malan@@@kpfu.ru хәзер дөрес түгел дип санала.

Электрон почта адресының гадәти таләпләрен исәпкә алып, кодны түбәндәгечә камилләштерергә мөмкин:

```
import re

email = input("Электрон почтагызны языгыз? ").strip()

if re.search(r"^[a-zA-Z0-9_]+@[a-zA-Z0-9_]+\.ru$",
email):
    print("Дөрес")
else:
    print("Дөрес түгел")
```

[a-zA-Z0-9_] регуляр аңлатмасы символларның **a**-дан **z**-гә, **A**-дан **Z**-гә, **0**-дән **9**-га кадәр булуын һәм **_** символын рөхсәт итә.

Шулай да, регуляр аңлатмаларда гомуми үрнәкләрне тырыш программистлар инде эшләгән. Кодны тагын да гади итәргә мөмкин:

```
import re

email = input("Электрон почтагызны языгыз? ").strip()

if re.search(r"^\w+@\w+\.ru$", email):
    print("Дөрес")
else:
    print("Дөрес түгел")
```

\w — **[a-zA-Z0-9_]** белән бертигез.

Өстәмә үрнәкләр:

- **\d** — саннар.
- **\D** — саннар түгел.
- **\s** — бушлык символлары.
- **\S** — бушлык символлары түгел.
- **\W** — сүз символлары түгел.

Әгәр электрон почта адресының төрле доменнарын (мәсьәлән, **.com**, **.org**) рөхсәт итәргә теләсәгез, кодны түбәндәгечә үзгәртә аласыз:

```
import re

email = input("Электрон почтагызны языгыз? ").strip()

if re.search(r"^\w+@\w+\.(com|edu|gov|net|org)$",
email):
    print("Дөрес")
else:
```

```
    print("Дөрес түгел")
```

символы "яисә" мәгънәсенә ия, һәм бу код төрле доменнарны тикшерә ала.

Сүзлеккә тагын да күбрәк символлар өстәү өчен түбәндәгеләрне карагыз:

- **A|B** — A яки B.
- **(...)** — төркем.
- **(?:...)** — төркемнең тотылмый торган версиясе.

Хәрефләрнең зур-кичек булуы мәсьәләсен ничек хәл итәргә мөмкин икәнлеген күрсәтү өчен кодны кире кайтарыйк:

```
import re

email = input("Электрон почтагызны языгыз? ").strip()

if re.search(r"^\w+@\w+\.ru$", email):
    print("Дөрес")
else:
    print("Дөрес түгел")
```

Алдагы кодта кулланылган **|** операторларын бетердек.

re.search функциясендә **flags** параметры бар. Кайбер корылган флаг үзгәрешләре:

- **re.IGNORECASE**
- **re.MULTILINE**
- **re.DOTALL**

Бу флагларны кодта түбәндәгечә кулланырга мөмкин:

```
import re

email = input("Электрон почтагызны языгыз? ").strip()

if re.search(r"^\w+@\w+\.ru$", email, re.IGNORECASE):
    print("Дөрес")
else:
    print("Дөрес түгел")
```

Өченче параметр итеп **re.IGNORECASE** өстәдек. Бу программаны MALAN@KPFU.ru кертсәгез, керү дөрес дип саналачак.

Менә бер электрон почта адресы: malan@cs50.kpfu.ru. Югарыдагы кодны кулланганда, бу дөрес түгел дип саналачак. Ни өчен?
Өстәмә `.` символы керүне дөрес түгел дип санарга сәбәп булып тора.

Без сүзлекне кулланып, идеяләрне төркемләп, кодны түбәндәгечә үзгәртә алабыз:

```
import re

email = input("Электрон почтагызны языгыз? ").strip()

if re.search(r"^\w+@(\w+\.)?\w+\.ru$", email,
re.IGNORECASE):
    print("Дөрес")
else:
    print("Дөрес түгел")
```

`(\w+\.)?` компиляторга бу экспрессиянең бер тапкыр керергә мөмкин яки бөтенләй булмаска мөмкин икәнен аңлата. Шулай итеп, malan@cs.kpfu.ru һәм malan@kpfu.ru адреслары дөрес дип саналачак.

Чыннан да, кодны тулысынча дөрес электрон почта адресын тикшерү өчен киңәйтергә мөмкин:

```
^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9](?:[a-zA-
Z0-9-]{0,61}[a-zA-Z0-9])?(?:\.[a-zA-Z0-9](?:[a-zA-Z0-9-
]{0,61}[a-zA-Z0-9])?)*$
```

Кулланучылар һәрвакыт чиста керүләр кертәчәкләр дип өмет итмәгез. Чыннан да, кулланучылар еш кына программистның ниятләрен бозалар.

Мәгълүматларны чистарту ысуллары бар.

`format.py` исемле файл булдырыгыз. Аннан соң текст мөхәррирендә түбәндәге кодны языгыз:

```
name = input("Исемегезне языгыз? ").strip()
print(f"Сәлам, {name}")
```

Бу "hello world" программасы кебек. Программаны эшләткәндә David керткәндә, ул дөрес эшли. Ләкин Malan, David керткәндә программа көтелгәнчә эшләми.

Бу керүне ничек чистарта алабыз? Кодны түбәндәгечә үзгәртә алабыз:

```
name = input("Исемегезне языгыз? ").strip()
```

```
if "," in name:
    last, first = name.split(", ")
    name = f"{first} {last}"
print(f"Сәлам, {name}")
```

Әгәр исемдә , булса, **`last, first = name.split(", ")`** башкарыла, һәм исем стандартлаштырыла. Мәсәлән, Malan, David керткәндә программа бу керүне дөрес итеп эшкәртә.

Ләкин, Malan,David (буш арасыз) керткәндә компилятор хата бирә. Регуляр аңлатмалар синтаксисын кулланып, кодны түбәндәгечә үзгәртә алабыз:

```
import re

name = input("Исемегезне языгыз? ").strip()
matches = re.search(r"^(.+), (.+)$", name)
if matches:
    last, first = matches.groups()
    name = first + " " + last
print(f"Сәлам, {name}")
```

`re.search` кулланучының керүеннән матчларны кайтара. Әгәр матчлар булса, программа аларны дөрес итеп эшкәртә. Мәсәлән, David Malan керткәндә if шарт үтәлми, һәм исем шул килеш кире кайтарыла. Malan, David керткәндә исем дөрес итеп эшкәртелә. 1111

Кушымтаны конкрет төркемнәрне кире кайтару өчен түбәндәгечә үзгәртәбез:

```
import re

name = input("Исемегезне языгыз? ").strip()
matches = re.search(r"^(.+), (.+)$", name)
if matches:
    name = matches.group(2) + " " + matches.group(1)
print(f"Сәлам, {name}")
```

Монда **`group`** берлек формасында кулланыла.

Кодны тагын да ныклы итү өчен:

```
import re

name = input("Исемегезне языгыз? ").strip()
matches = re.search(r"^(.+), (.+)$", name)
if matches:
    name = matches.group(2) + " " + matches.group(1)
```

```
print(f"Сәлам, {name}")
```

group(2) һәм **group(1)** арасында пробел куелган. Беренче төркем **,** символы сул ягында, икенче төркем уң ягында урнашкан.

Пробелларны рөхсәт итү

Ләкин, **Malan,David** (пробелсыз) керткәндә программа әле дә хата бирәчәк. Шунлыктан, түбәндәгечә үзгәртәбез:

```
import re

name = input("Исемегезне языгыз: ").strip()
matches = re.search(r"^(.+), *(.+)$", name)
if matches:
    name = matches.group(2) + " " + matches.group(1)
print(f"Сәлам, {name}")
```

***** символы пробелларның булуын яки булмавын рөхсәт итә. Бу код хәзер **Malan,David** һәм **David, Malan** кебек керүләрне дөрес итеп эшкәртәчәк.

Walrus операторын куллану

Куллану шартында **re.search** куллану еш очрый, ләкин бу операторны түбәндәгечә берләштерергә мөмкин:

```
import re

name = input("Исемегезне языгыз? ").strip()
if matches := re.search(r"^(.+), *(.+)$", name):
    name = matches.group(2) + " " + matches.group(1)
print(f"Сәлам, {name}")
```

Монда **:=** операторын кулланып, ике юлны берләштерәбез. Бу оператор уңнан сулга кыйммәт бирә һәм бер үк вакытта буле шартын сорауга мөмкинлек бирә. Бу "walrus operator" дип атала.

Кулланучы кертүен алу

Хәзерге вакытка кадәр, без кулланучының керүләрен валидацияләдек һәм тазарттык. Хәзер, кулланучының керүеннән конкрет мәгълүматларны алыйк. **twitter.py** исемле файл булдырыгыз һәм текст мөхәррирендә түбәндәге кодны языгыз:

```
url = input("URL: ").strip()
print(url)
```

2.9. Өстәмә мәгълумат

Күп кенә алдагы дәресләрдә без Python белән бәйле бик күп нәрсәләрне өйрәндек! Бу дәрестә без элеккә карап китмәгән күп кенә "эт сетэра" элементларына игътибар итәчәкбез. "Эт сетэра" турыдан-туры "һәм калганнары" дигәнне аңлата!

Чыннан да, әгәр сез Python документациясенә карасагыз, сез башка функцияләрнең "калганы"ны табачаксыз.

Set

Математикада, сет (җыелма) саннарның бертөрле булмаган җыелмасы булып санала. Текст редакторында түбәндәгечә кодлагыз:

```
студентлар = [
    {"исем": "Гөлназ", "төркем": "Казан"},
    {"исем": "Айдар", "төркем": "Казан"},
    {"исем": "Рөстәм", "төркем": "Казан"},
    {"исем": "Дамир", "төркем": "Уфа"},
    {"исем": "Ләйсән", "төркем": "Чаллы"},
]

төркемнәр = []
for студент in студентлар:
    if студент["төркем"] not in төркемнәр:
        төркемнәр.append(студент["төркем"])

for төркем in sorted(төркемнәр):
    print(төркем)
```

Бу программаның ничек эшләвен күзәтегез: бездә һәрберсе студент булган сүзлекләр исемлеге бар. Буш исемлек **төркемнәр** буларак булдырыла. Без һәр студентны тикшерәбез, әгәр студентның төркеме **төркемнәр** исемлегендә юк икән, без аны өстибез.

Чыннан да, без билгеле бер функцияләрне кулланып, кабатланмаучыларны бетерү өчен **set** объектларын куллана алабыз. Текст редакторында түбәндәгечә кодлагыз:

```
студентлар = [
    {"исем": "Гөлназ", "төркем": "Казан"},
    {"исем": "Айдар", "төркем": "Казан"},
    {"исем": "Рөстәм", "төркем": "Казан"},
    {"исем": "Дамир", "төркем": "Уфа"},
```

```
    {"исем": "Ләйсән", "төркем": "Чаллы"},
]

төркемнәр = set()
for студент in студентлар:
    төркемнәр.add(студент["төркем"])

for төркем in sorted(төркемнәр):
    print(төркем)
```

Бу очракта, кабатланмаучыларны тикшерү кирәкми. **set** объекты моны автоматик рәвештә башкара.

Pythonның **set** турында документациясендә күбрәк өйрәнергә була.

Глобаль үзгәрмәләр

Башка программалаштыру телләрендә, глобаль үзгәрмәләр бар, алар теләсә кайсы функциягә кереп була. Без бу мөмкинлекне Pythonда куллана алабыз. Текст редакторында түбәндәгечә кодлагыз:

```
баланс = 0


def төп():
    print("Баланс:", баланс)


if __name__ == "__main__":
    төп()
```

баланс дип аталган глобаль үзгәрмә, функцияләрнең тышында барлыкка килә.

Әгәр дә без шушы кодны эшләтсәк, барысы да яхшы дип уйларсыз. Ләкин, бу юк! Текст редакторында түбәндәгечә кодлагыз:

```
баланс = 0


def төп():
    print("Баланс:", баланс)
    депозит(100)
    чыгым(50)
    print("Баланс:", баланс)
```

```
def депозит(n):
    баланс += n


def чыгым(n):
    баланс -= n


if __name__ == "__main__":
    төп()
```

Бу очракта, без **баланс** үзгәрмәсенә өстәмә һәм чыгымнар кертәбез. Ләкин, шушы кодны эшләткәндә, **UnboundLocalError** хата белән очрашырсыз. Бу хата **баланс**ны функция эчендә яңадан билгеләү мөмкин түгеллеген күрсәтә.

Глобаль үзгәрмә белән эшләү өчен, без **global** ачкыч сүзен кулланырга тиешбез. Текст редакторында түбәндәгечә кодлагыз:

```
баланс = 0


def төп():
    print("Баланс:", баланс)
    депозит(100)
    чыгым(50)
    print("Баланс:", баланс)


def депозит(n):
    global баланс
    баланс += n


def чыгым(n):
    global баланс
    баланс -= n


if __name__ == "__main__":
    төп()
```

global ачкыч сүзе һәр функциягә **баланс**ның локаль түгел, глобаль булуын аңлата. Шул рәвешле, безнең код хәзер дөрес эшли!

Объектка юнәлтелгән программалаштыру тәҗрибәбезне кулланып, без глобаль үзгәрмәне класс кулланып алмаштыра алабыз. Текст редакторында түбәндәгечә кодлагыз:

```
class Аккаунт:
    def __init__(self):
        self._баланс = 0

    @property
    def баланс(self):
        return self._баланс

    def депозит(self, n):
        self._баланс += n

    def чыгым(self, n):
        self._баланс -= n


def төп():
    аккаунт = Аккаунт()
    print("Баланс:", аккаунт.баланс)
    аккаунт.депозит(100)
    аккаунт.чыгым(50)
    print("Баланс:", аккаунт.баланс)


if __name__ == "__main__":
    төп()
```

Бу очракта, без **Аккаунт** классын булдырабыз, ул **баланс**ны саклый һәм аңа керү мөмкинлеге бирә торган методлар белән тәэмин итә.

Гомумән алганда, глобаль үзгәрмәләрне бик сак итеп кулланырга кирәк, әгәр дә бар да!

Константалар

Кайбер телләрдә үзгәрми торган үзгәрмәләр булдырырга мөмкин, алар "константалар" дип атала. Константалар программаны саклы язуга ярдәм итә һәм мөһим кыйммәтләрнең үзгәрмәслеген тәэмин итә.

Текст редакторында түбәндәгечә кодлагыз:

```
мыраулар = 3
```

```
for _ in range(мыраулар):
    print("мырау")
```

Бу очракта, **мыраулар** — безнең константа. Константалар гадәттә зур хәрефләр белән язылалар һәм коднының башында урнаштырылалар. Ләкин Pythonда чыннан да константаларны үзгәртүне булдырмас өчен механизм юк! Шунлыктан, сез "ихтыяр системасында" эшлисез: әгәр дә үзгәрмә исемнәре бөтен хәрефләр белән язылган булса, аларны үзгәртмәгез!

Без класс "константа" булдыра алабыз, хәзерге вакытта "константа" дип әйткәндә Python чыннан да аны тулысынча якламавын белгән булсак та. Текст редакторында түбәндәгечә кодлагыз:

```
class Мияу:
    мыраулар = 3

    def мыраулау(self):
        for _ in range(Мияу. мыраулар):
            print("мырау")


песи = Мияу()
песи.мыраулау()
```

Бу очракта, **мыраулар** класс методларыннан тыш билгеләнгән, һәм алар **Мияу. мыраулар** аша кулланылалар.

Тип белгеләре (Type Hints)

Башка телләрдә, сез үзгәрмә типларын ачык рәвештә белдерәсез. Курсның алдагы өлешләрендә күргәнчә, Pythonда типларны ачык рәвештә игълан итү таләп ителми. Ләкин, барлык үзгәрмәләрегезнең дөрес типта булуын тәэмин итү яхшы практика булып тора. **mypy** — бу сезгә үзгәрмәләрегезнең дөрес типта булуын тикшерергә ярдәм итә торган программа. Сез терминалда түбәндәге команданы эшләтеп, **mypy**ны урнаштыра аласыз: **pip install mypy**. Текст редакторында түбәндәгечә кодлагыз:

```
def мыраулау(n):
    for _ in range(n):
        print("мырау")


сан = input("Ничә тапкыр мыраулауны телисең? ")
мыраулар = мыраулау(сан)
```

Сез инде күрәсез ки, **сан = input("Ничә тапкыр мыраулауны телисең? ")** юлында **сан** — **str** типында кайта, ләкин **мыраулау** функциясе **int** таләп итә!

Тип белгеләре функцияләргә нинди типлар көтелгәнен аңлатырга ярдәм итә. Текст редакторында түбәндәгечә кодлагыз:

```
def мыраулау(n: int):
    for _ in range(n):
        print("мырау")


сан = input("Ничә тапкыр мыраулауны телисең? ")
мыраулау(сан)
```

Ләкин, программа әле дә хата бирәчәк.

муруны урнаштыргач, терминалда **муру meows.py** командасын эшләтегез. **муру** сезгә бу хатаны ничек төзәтергә кирәклеге турында киңәшләр бирәчәк. Сез барлык үзгәрмәләрегезгә тип белгеләрен өсти аласыз. Текст редакторында түбәндәгечә кодлагыз:

```
def мыраулау(n: int):
    for _ in range(n):
        print("мырау")


сан: int = input("Ничә тапкыр мыраулауны телисең? ")
мыраулау(сан)
```

Бу очракта, **сан** өчен тип белгесе өстәлгән.

Соңгы хата мына мондый:

```
def мыраулау(n: int) -> None:
    for _ in range(n):
        print("мырау")


сан: int = int(input("Ничә тапкыр мыраулауны телисең?
"))
мыраулау(сан)
```

-> None билгесе **мыраулау** функциясенең бернәрсә дә кайтармавын күрсәтә.

Без үз кодларыбызны тагын да төзәтеп була:

```
def мыраулау(n: int) -> str:
    return "мырау\n" * n


сан: int = int(input("Ничә тапкыр мыраулауны телисең? "))
мыраулар: str = мыраулау(сан)
print(мыраулар, end="")
```

Бу очракта, **мыраулау** функциясе **str** кайтара, һәм **муру** бу хаталарны күрсәтми.

Докстринглар (Docstrings)

Функциянең максатын аңлатуның стандарт ысулы — докстринг куллану. Текст редакторында түбәндәгечә кодлагыз:

```
def мыраулау(n):
    """Мяуны n тапкыр бастыру."""
    return "мырау\n" * n


сан = int(input("Ничә тапкыр мыраулауны телисең? "))
мыраулар = мыраулау(сан)
print(мыраулар, end="")
```

Өчле куш тырнаклар функциянең нәрсә эшләвен күрсәтә.

Функцияләрнең үзенчәлекләрен документлаштыру өчен докстрингларны куллану яхшы практика. Текст редакторында түбәндәгечә кодлагыз:

```
def мыраулау(n: int) -> str:
    """
    Мяуны n тапкыр кайтару.

    :param n: Мяуның саны
    :type n: int
    :raise TypeError: Әгәр n int булмаса
    :return: Бер юлга бер мырауны керткән Юл
    :rtype: str
    """
    return "мырау\n" * n
```

```
сан = int(input("Ничә тапкыр мыраулауны телисең? "))
мыраулар = мыраулау(сан)
print(мыраулар, end="")
```

Докстрингларның күп параметрларын куллана аласыз. Мәсәлән, функциянең кабул иткән параметрларын һәм нәрсә кайтара икәнен аңлаталар.

Сәламәт инструментлар, мәсәлән Sphinx, докстрингларны парслау һәм автоматик рәвештә веб битләре һәм PDF файллары рәвешендә документация булдырырга ярдәм итә ала.

argparse

Әгәр дә безнең программа командалык юл аргументларын кулланырга тиеш икән, ничек эшләргә? Текст редакторында түбәндәгечә кодлагыз:

```
import sys

if len(sys.argv) == 1:
    print("мырау")
elif len(sys.argv) == 3 and sys.argv[1] == "-n":
    n = int(sys.argv[2])
    for _ in range(n):
        print("мырау")
else:
    print("кулланылыш: meows.py [-n САН]")
```

Бу программаның ничек эшләвен күзәтегез: **sys** импортлана, **sys.argv** — программага тапшырылган аргументлар массивы. Без берничә **if** шартларын кулланып, программа дөрес кулланылганмы икәнен тикшерәбез.

Әгәр дә безнең программа бик катлаулы булса, барлык аргументларны тикшерү кыенлаша ала. Монда **argparse** китапханәсе ярдәм итә.

Текст редакторында түбәндәгечә кодлагыз:

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("-n")
args = parser.parse_args()

for _ in range(int(args.n)):
    print("мырау")
```

argparse **sys** урынына
импортлана. **ArgumentParser** классыннан **parser** объекты булдырыла. **add_argument** методын кулланып, без кулланучының нинди аргументлар бирәчәген аңлатабыз. Соңыннан, **parse_args** методын эшләтеп, аргументларны анализлыйбыз.

Без шулай ук, кулланучыга программа дөрес кулланылышын күрсәтү өчен өстәмә мөмкинлекләр бирә алабыз. Текст редакторында түбәндәгечә кодлагыз:

```
import argparse

parser = argparse.ArgumentParser(description="Мяулау
программасы")
parser.add_argument("-n", help="мыраулауның саны",
type=int, default=1)
args = parser.parse_args()

for _ in range(args.n):
    print("мырау")
```

Бу очракта, кулланучы **python meows.py --help** яки **-h** командасын эшләткәндә, программа кулланылыш турында мәгълүмат күрсәтәчәк.

Бу программаны тагын да яхшыртырга була:

```
import argparse

parser = argparse.ArgumentParser(description="Мяулау
программасы")
parser.add_argument("-n", help="мыраулауның саны",
type=int, default=1)
args = parser.parse_args()

for _ in range(args.n):
    print("мырау")
```

Бу очракта, **-n** аргументы өчен дефолт кыйммәт билгеләнгән, ягъни кулланучы аны бирмәсә, программа бер тапкыр мыраулап чыгарачак.

Unpacking

Бер үзгәрмәгә берничә үзгәрмә сакларга мөмкинлек бирү ничек? Текст редакторында түбәндәгечә кодлагыз:

```
беренче, _ = input("Сезнең исемегез ничек? ").split("
")
```

```
print(f"Исәнмесез, {беренче}")
```

Бу программа кулланучының исемен ике өлешкә бүлә, әмма икенче өлешне игътибарсыз калдыра.

Без үзгәрмәләрне аерып саклауны тагын да үстерә алабыз. Текст редакторында түбәндәгечә кодлагыз:

```
def җыелма(алтын, көмеш, күмеc):
    return (алтын * 17 + көмеш) * 29 + күмеc


print(җыелма(100, 50, 25), "Күмеc")
```

Бу программа алтын, көмеш һәм күмиснең гомуми кыйммәтен исәпли.

Әгәр без үзебезнең акчаларны исемлеккә сакласак, ничек эшләргә? Текст редакторында түбәндәгечә кодлагыз:

```
def җыелма(алтын, көмеш, күмеc):
    return (алтын * 17 + көмеш) * 29 + күмеc


акчалар = [100, 50, 25]

print(җыелма(акчалар[0], акчалар[1], акчалар[2]),
"Күмис")
```

Бу очракта, без исемлекне индексләү белән кулланабыз, ләкин бу бик озын булырга мөмкин.

Көтелмәгәнчә, без исемлекне турыдан-туры функциягә тапшырмыйча, unpacking куллана алабыз. Текст редакторында түбәндәгечә кодлагыз:

```
def җыелма(алтын, көмеш, күмеc):
    return (алтын * 17 + көмеш) * 29 + күмеc


акчалар = [100, 50, 25]

print(җыелма(*акчалар), "Күмис")
```

***акчалар** исемлекне unpacking ярдәмендә функциягә бирә, һәм һәр элемент аерым аргумент буларак тапшырылган.

Әгәр без акчаларны теләсә нинди тәртиптә исемләп тапшырсаң, ничек эшләргә? Текст редакторында түбәндәгечә кодлагыз:

```
def җыелма(алтын, көмеш, күміс):
    return (алтын * 17 + көмеш) * 29 + күміс


print(җыелма(алтын=100, көмеш=50, күміс=25), "Күмис")
```

Бу очракта, без функциягә исемле аргументлар тапшырабыз, һәм программа дөрес итеп исәпли.

Сүзлекләр турында уйлагач, без unpackingны ничек кулланырга икәнен аңлый алабыз! Текст редакторында түбәндәгечә кодлагыз:

```
def җыелма(алтын, көмеш, күміс):
    return (алтын * 17 + көмеш) * 29 + күміс


акчалар = {"алтын": 100, "көмеш": 50, "күмис": 25}

print(җыелма(**акчалар), "Күмис")
```

Бу очракта, **`**акчалар`** сүзлеген unpacking ярдәмендә функциягә тапшыра, һәм ачкычлар белән кыйммәтләр туры килүен тәэмин итә.

args һәм kwargs

Алдагы курсның өлешләрендә карыйккан **`print`** функциясендә:

```
print(*objects, sep=' ', end='\n', file=sys.stdout,
flush=False)
```

`args` — позицион аргументлар, мәсәлән **`print("Исәнмесез", "Дөнья")`**. **`kwargs`** — исемле аргументлар, мәсәлән **`print(end="")`**.

Функцияләрегезгә билгеле бер саннан артык позицион һәм исемле аргументлар тапшыру мөмкинлеге бирергә теләсәгез, сез **`*args`** һәм **`**kwargs`** куллана аласыз. Текст редакторында түбәндәгечә кодлагыз:

```
def f(*args, **kwargs):
    print("Позицион:", args)


f(100, 50, 25)
```

Бу очракта, функция **args** буларак позицион аргументларны алып, аларны бастыра.

Без шулай ук исемле аргументларны тапшыра алабыз. Текст редакторында түбәндәгечә кодлагыз:

```
def f(*args, **kwargs):
    print("Исемле:", kwargs)


f(алтын=100, көмеш=50, күміс=25)
```

Бу очракта, функция **kwargs** буларак исемле аргументларны алып, аларны бастыра.

print функциясе мисалында, ***objects** күпсанлы позицион аргументларны алырга мөмкинлек бирә.

Pythonның print документациясендә күбрәк өйрәнергә була.

map

Алдагы дәресләрдә без процедура буенча программалаштыру белән башладык. Соңыннан Pythonның объектка юнәлтелгән программалаштыру теле икәнен күрдек. Функциональ программалаштыруның кайбер билгеләрен дә күрдек, монда функцияләрнең кайтару кыйммәте булмыйча, берничә нәтиҗәсе бар. Буны текст редакторында түбәндәгечә күрсәтәбез:

```
def төп():
    мыраулау("Бу CS50")


def мыраулау(сүз):
    print(сүз.upper())


if __name__ == "__main__":
    төп()
```

Бу программа гади генә бер сүзне зур хәрефләр белән бастыра.

Күпсанлы сүзләрне бастыру уңайлы булыр иде. Текст редакторында түбәндәгечә кодлагыз:

```
def төп():
    мыраулау(["Бу", "CS50"])
```

```
def мыраулау(сүзләр):
    югары_хәрефле = []
    for сүз in сүзләр:
        югары_хәрефле.append(сүз.upper())
    print(*югары_хәрефле)


if __name__ == "__main__":
    төп()
```

Бу очракта, без **югары_хәрефле** исемле исемлеккә сүзләрне зур хәрефләргә әйләндерәбез һәм бастырабыз.

Бу бик озын икән! Исемлекне турыдан-туры функциягә тапшырсаң, ничек эшләр иде? Unpacking кулланып, бу мөмкинлекне куллана алабыз. Текст редакторында түбәндәгечә кодлагыз:

```
def төп():
    мыраулау("Бу", "CS50")


def мыраулау(*сүзләр):
    югары_хәрефле = []
    for сүз in сүзләр:
        югары_хәрефле.append(сүз.upper())
    print(*югары_хәрефле)


if __name__ == "__main__":
    төп()
```

Бу очракта, ***сүзләр** күпсанлы аргументларны кабул итә һәм аларны берәм-берәм эшкәртә.

map функциясе сезгә функцияне мәгълүматларның бер төркеменә кулланырга мөмкинлек бирә. Практикада, түбәндәгечә кодлагыз:

```
def төп():
    мыраулау("Бу", "CS50")


def мыраулау(*сүзләр):
    югары_хәрефле = map(str.upper, сүзләр)
```

```
    print(*югары_хәрефле)


if __name__ == "__main__":
    төп()
```

Бу очракта, **map** ике аргумент ала: беренчесе — һәр элементка кулланылырга тиешле функция (**str.upper**), икенчесе — шушы функциянең кулланылырга тиешле мәгълүматлар (**сүзләр**). Шунлыктан, барлык сүзләргә **str.upper** функциясе кулланыла һәм нәтиҗә **югары_хәрефле** исемле map объектына саклана.

Pythonның map документациясендә күбрәк өйрәнергә була.

List Comprehensions

List comprehensions сезгә исемлекне бер генә, матур юлда булдырырга мөмкинлек бирә. Буны түбәндәгечә күрсәтәбез:

```
def төп():
    мыраулау("Бу", "CS50")


def мыраулау(*сүзләр):
    югары_хәрефле = [сүз.upper() for сүз in сүзләр]
    print(*югары_хәрефле)


if __name__ == "__main__":
    төп()
```

Бу очракта, **map** урынына, без бер юллы list comprehension кулландык, ул һәр сүзгә **.upper()** методын куллана.

Бу концепцияне тагын да алга китереп, башка программаны карап чыгыйк. Текст редакторында **gryffindors.py** исемле файлны ачып, түбәндәгечә кодлагыз:

```
студентлар = [
    {"исем": "Гөлназ", "төркем": "Казан"},
    {"исем": "Айдар", "төркем": "Казан"},
    {"исем": "Рөстәм", "төркем": "Казан"},
    {"исем": "Дамир", "төркем": "Уфа"},
]
```

```
казаннар = []
for студент in студентлар:
    if студент["төркем"] == "Казан":
        казаннар.append(студент["исем"])

for казан in sorted(казаннар):
    print(казан)
```

Бу программада без шартлы конструкция кулланып, **Казан** төркемендәге студентларның исемнәрен туплыйбыз һәм аларны тәртипләп бастырабыз.

Бу кодны list comprehension кулланып түбәндәгечә гадиәйтә алабыз:

```
студентлар = [
    {"исем": "Гөлназ", "төркем": "Казан"},
    {"исем": "Айдар", "төркем": "Казан"},
    {"исем": "Рөстәм", "төркем": "Казан"},
    {"исем": "Дамир", "төркем": "Уфа"},
]

казаннар = [студент["исем"] for студент in студентлар
if студент["төркем"] == "Казан"]

for казан in sorted(казаннар):
    print(казан)
```

Бу очракта, list comprehension барысын да бер юлда башкарырга мөмкинлек бирә.

filter

Pythonның **filter** функциясе сезгә шартка туры килгән мәгълүматларны сайларга мөмкинлек бирә. Текст редакторында түбәндәгечә кодлагыз:

```
студентлар = [
    {"исем": "Гөлназ", "төркем": "Казан"},
    {"исем": "Айдар", "төркем": "Казан"},
    {"исем": "Рөстәм", "төркем": "Казан"},
    {"исем": "Дамир", "төркем": "Уфа"},
]


def Казан_студенты(c):
    return s["төркем"] == "Казан"
```

```
казаннар = filter(Казан_студенты, студентлар)

for казан in sorted(казаннар, key=lambda s: s["исем"]):
    print(казан["исем"])
```

Бу очракта, без **Казан_студенты** исемле функция булдырабыз, ул студентның төркемен тикшерә. **filter** функциясе бу функцияне һәр студентка куллана һәм **Казан** төркемендәге студентларны сайлый.

filter шулай ук lambda функцияләрен кулланып тагын да гадиәйтергә мөмкин:

```
студентлар = [
    {"исем": "Гөлназ", "төркем": "Казан"},
    {"исем": "Айдар", "төркем": "Казан"},
    {"исем": "Рөстәм", "төркем": "Казан"},
    {"исем": "Дамир", "төркем": "Уфа"},
]


казаннар = filter(lambda s: s["төркем"] == "Казан",
студентлар)

for казан in sorted(казаннар, key=lambda s: s["исем"]):
    print(казан["исем"])
```

Бу очракта, lambda функциясе **Казан** төркемендәге студентларны сайлый.

Pythonның filter документациясендә күбрәк өйрәнергә була.

Dictionary Comprehensions

List comprehensions концепциясен сүзлекләргә дә куллана алабыз. Текст редакторында түбәндәгечә кодлагыз:

```
студентлар = ["Гөлназ", "Айдар", "Рөстәм"]

казаннар = []

for студент in студентлар:
    казаннар.append({"исем": студент, "төркем":
"Казан"})

print(казаннар)
```

Бу очракта, без шартлы конструкция кулланмыйча, list comprehension кулланып сүзлекләр ясыйбыз:

```
студентлар = ["Гөлназ", "Айдар", "Рөстәм"]

казаннар = [{"исем": студент, "төркем": "Казан"} for
студент in студентлар]

print(казаннар)
```

Моннан тыш, без түбәндәгечә сүзлек comprehension куллана алабыз:

```
студентлар = ["Гөлназ", "Айдар", "Рөстәм"]

казаннар = {студент: "Казан" for студент in студентлар}

print(казаннар)
```

Бу очракта, сүзлек ачкычлары итеп студент исемнәрен, ә кыйммәтләре итеп "Казан"ны кулланабыз.

enumerate

Без һәр студентка саннар бирергә теләсәк, ничек эшләргә? Текст редакторында түбәндәгечә кодлагыз:

```
студентлар = ["Гөлназ", "Айдар", "Рөстәм"]

for i in range(len(студентлар)):
    print(i + 1, студентлар[i])
```

Бу программаның ничек эшләвен күзәтегез: без индекслар кулланып, һәр студентка сан бирәбез.

enumerate функциясен кулланып, бу эшне җиңеләйтә алабыз:

```
студентлар = ["Гөлназ", "Айдар", "Рөстәм"]

for i, студент in enumerate(студентлар):
    print(i + 1, студент)
```

Бу очракта, **enumerate** функциясе һәр студентның индексын һәм исемен кайтара, һәм без аларны турыдан-туры бастырабыз.

Generators һәм Iterators

Pythonда, сезнең системаның ресурсларыннан исраф булдырмас өчен, мәгълүматларны җиңелрәк эшкәртү ысуллары бар. Күрәсең, АКШта, йокларга авырлык кичергәндә, кешеләр күңеле белән "кәҗәләр санау" гадәте белән шөгыльләнә.

Текст редакторында **sleep.py** исемле файлны ачып, түбәндәгечә кодлагыз:

```
n = int(input("Ничә кәҗә санауны телисең? "))
for i in range(n):
    print("🐑" * i)
```

Бу программа сезнең сораган кәҗәләр санын санауны башлый.

Без үзебезнең программаны катлауландырырга һәм **main** функциясен өстәргә телибез. Текст редакторында түбәндәгечә кодлагыз:

```
def төп():
    n = int(input("Ничә кәҗә санауны телисең? "))
    for i in range(n):
        print("🐑" * i)


if __name__ == "__main__":
    төп()
```

Бу очракта, без **main** функциясен кулланып, кодны абстракцияләдек.

Без бу кодны тагын да яхшыртырга телибез. Мәсәлән, **кажә** функциясен булдырыйк:

```
def төп():
    n = int(input("Ничә кәҗә санауны телисең? "))
    for s in кажә(n):
        print(s)


def кажә(n):
    flock = []
    for i in range(n):
        flock.append("🐑" * i)
    return flock


if __name__ == "__main__":
    төп()
```

Бу очракта, **кажә** функциясе **flock** исемле исемлеккә кәҗәләрне өсти һәм аларны кайтара.

Әгәр сез зур саннарны, мәсәлән, 1000000 кәҗә санарга тырышсагыз, программа тукталып калырга яки төшәргә мөмкин. Бу очракта, ресурсларны исраф итмичә эшләү өчен **yield** генераторын куллана алабыз. Текст редакторында түбәндәгечә кодлагыз:

```
def төп():
    n = int(input("Ничә кәҗә санауны телисең? "))
    for s in кажә(n):
        print(s)


def кажә(n):
    for i in range(n):
        yield "🐑" * i


if __name__ == "__main__":
    төп()
```

Бу очракта, **yield** һәр тапкыр бер кәҗәне кайтара, һәм программа ресурсларны исраф итмичә эшли.

Өченче бүлек.
JavaScript телендә программалаштыру элементлары

3.1. Танышу (Introduction)

JavaScript — веб-браузерларда эшләүче, динамик һәм интерактив веб-элементларны булдырырга мөмкинлек бирүче югары дәрәҗәдәге интерпретацияләнгән программалаштыру теле. Ул шулай ук сервер ягы технологияләре белән дә эшли ала (мәсәлән, Node.js).

JavaScript ярдәмендә сез веб-страницаларга слайдерлар, формаларны тикшерү, динамик контент кебек төрле интерактив элементлар өсти аласыз. Ул объектка юнәлешле һәм функциональ программалаштыру парадигмаларын да хуплый, бу исә аны киң файдалы корал итә.

Беренче программа: "Сәлам, Дөнья!"
JavaScript программасын язар өчен HTML документында **`<script>`** тегы эчендә кодны урнаштырырга кирәк.

```
<!DOCTYPE html>
<html lang="tt">
<head>
    <meta charset="UTF-8">
    <title>JavaScript дәреслеге</title>
</head>
<body>
    <h1>JavaScript белән танышу</h1>
    <script>
        console.log("Сәлам, Дөнья!");
    </script>
</body>
</html>
```

Төшенчәләр:

- **`<!DOCTYPE html>`** — документның HTML5 стандартын куллануын күрсәтә.
- **`<html lang="tt">`** — битнең татар телендә булуын билгели.
- **`<script>`** — JavaScript кодын урнаштыру өчен файдаланыла.
- **`console.log`** — браузерның консольгә хәбәр чыгару функциясе, отладка өчен уңайлы.

Бу мисал JavaScriptны өйрәнүне башлау өчен төп адымнарны күрсәтә һәм телнең нигезен аңларга ярдәм итә.

3.2. Үзгәрешлеләр (Variables)

JavaScriptта үзгәрешлеләрне билгеләү өчен **let**, **const**, һәм элеккеге **var** ачкыч сүзләре кулланыла. Бүгенге көндә **let** һәм **const** куллану киңәш ителә, чөнки алар кодны укуны һәм хаталарны булдырмауны җиңеләйтә.

- **let** — үзгәрешле кыйммәтләрне саклау өчен кулланыла.
- **const** — бер тапкыр бирелгән һәм үзгәрми торган кыйммәтләр (константалар) өчен кулланыла.

Үзгәрешле билгеләү мисалы:

```
let исем = "Айдар";
const КӨМЕШ = 50;

console.log(исем); // "Айдар"
console.log(КӨМЕШ); // 50

исем = "Гөлназ";
console.log(исем); // "Гөлназ"

// КӨМЕШ = 60; // Хата: КӨМЕШ константа, аны үзгәртеп булмый
```

Төшенчәләр:

- `let исем = "Айдар";` — **исем** исемле үзгәрешле булдырабыз һәм аңа `"Айдар"` кыйммәтен бирәбез.
- `const КӨМЕШ = 50;` — **КӨМЕШ** исемле константа булдырабыз һәм аңа `50` кыйммәтен бирәбез.
- `console.log` — консольгә хәбәр чыгара.

Исемнәр кагыйдәләре:

- Исемнәр хәреф, сан, `$` яки `_` символы белән башланырга мөмкин.
- Исемнәр эчендә хәрефләр, саннар, `$` яки `_` кулланыла ала.
- JavaScriptта зур һәм кечкенә хәрефләр аерыла: мәсәлән, **исем** һәм **Исем** төрле үзгәрешлеләр.

var белән let аермасы:
Элекке вакытта үзгәрешле билгеләү өчен **var** кулланылган. Ул функциональ

масштабка ия, һәм кайбер очракларда көтелмәгән нәтиҗәләр бирергә мөмкин. Шуңа күрә хәзер **let** һәм **const** файдалану киңәш ителә.

Мисал:

```
function мисал() {
    if (true) {
        var x = 10; // var: функциянең бөтен мәйданында күренә
        let y = 20; // let: блок эчендә генә күренә
    }
    console.log(x); // 10
    // console.log(y); // Хата: y билгеләнмәгән
}

мисал();
```

Бу мисалда:

- **x var** белән билгеләнгәнлектән, ул функциянең бөтен мәйданында күренә.
- **y let** белән билгеләнгәнлектән, ул тик үзенең блогы эчендә генә күренә.

3.3. Арифметик операторлар (Arithmetic Operators)

JavaScript арифметик операцияләр өчен киң мөмкинлекләр бирә, төрле операторлар ярдәмендә төрле математик гамәлләр башкарырга мөмкин. Аларны куллану аңлаешлы һәм нәтиҗәле код язарга ярдәм итә.

Арифметик операторлар:

Оператор	Куллану үрнәге	Аңлатма
+	a + b	Саннарны куша
-	a - b	Бер санны икенчесеннән чыгара
*	a * b	Саннарны күпәйтә
/	a / b	Саннарны бүлә
%	a % b	Калдыкны таба (модуль)
**	a ** b	Санны дәрәҗәгә күтәрә

Мисаллар:

```
let a = 10;
let b = 3;

console.log(a + b); // 13
console.log(a - b); // 7
console.log(a * b); // 30
console.log(a / b); // 3.3333333333333335
console.log(a % b); // 1
console.log(a ** b); // 1000
```

Аңлатмалар:

- a + b — a һәм b саннарын куша.
- a - b — a саныннан b санын чыгара.
- a * b — a санын b санына күпәйтә.
- a / b — a санын b санына бүлә.
- a % b — a санын b санына бүлгәндә калганны таба.
- a ** b — a санын b дәрәҗәсенә күтәрә.

Практик мисал: катлаулы исәпләүләр

```
let алтын = 100;
let көмеш = 50;
let энҗе = 25;

// Акчаларның гомуми кыйммәтен исәпләү
let гомумиКыйммәт = (алтын * 17 + көмеш) * 29 + энҗе;

console.log("Гомуми Кыйммәт:", гомумиКыйммәт, "Көмеш");
// Гомуми Кыйммәт: 5150 Көмеш
```

Төшенчәләр:

- Монда алтын, көмеш, һәм энҗе саннары кулланылган. Аларны Harry Potter китапларындагы акчалар төркеменә охшатып мисал буларак карарга мөмкин.
- Исәпләү:
  - 1 алтын = 17 көмеш.
  - 1 көмеш = 29 энҗе.
  - Барлык кыйммәтләр берәмлеккә күчерелеп исәпләнә.

Өстәмә мәгълүмат: инкремент һәм декремент

JavaScript шулай ук саннарны берәмлеккә арттыру яки киметү өчен махсус операторлар тәкъдим итә:

- `++` — санны 1гә арттыра (инкремент).
- `--` — санны 1гә киметә (декремент).

Мисаллар:

```
let x = 5;

x++;
console.log(x); // 6

x--;
console.log(x); // 5
```

Постфикс һәм префикс аермасы:

- Постфикс (x++) — операциядән соң арттыра.
- Префикс (++x) — операциядән алда арттыра.

Өстәмә чагыштырулар:

JavaScript арифметик операторларның комбинацияләрен кулланып катлаулы исәпләүләр ясауны гадиләштерә. Формулалар һәм аңлатмаларны кодка өстәү үз программаларыгызны укыла торган итә һәм нәтиҗәле исәпләүләр ясарга мөмкинлек бирә.

3.4. Шартлы операторлар (Conditional Operators)

Шартлы операторлар программа логикасын төрле юлларга бүлү мөмкинлеге бирә һәм JavaScriptта иң еш кулланыла торган коралларның берсе булып тора. Алар ярдәмендә программа төрле шартларга карап төрле гамәлләр башкара ала.

Шартлы операторлар: `if`, `else if`, `else`

Мисал:

```
let балансы = 1000;

if (балансы > 500) {
    console.log("Баланс зур.");
} else if (балансы > 100) {
    console.log("Баланс уртача.");
```

```
} else {
    console.log("Баланс аз.");
}
```

Аңлатмалар:

- **if** — шарт дөрес булганда, кодны эшли.
- **else if** — алдагы шартлар дөрес булмаса, икенче шартны тикшерә.
- **else** — алдагы барлык шартлар дөрес булмаса, эшли торган блок.

Төгәлрәк мисал:

```
let яше = 20;

if (яше >= 18) {
    console.log("Сез өлкән.");
} else {
    console.log("Сез яшь.");
}
```

Нәтиҗә:

- Әгәр яше үзгәрешле 18 яки аннан зуррак булса, программа "Сез өлкән." дип чыгара.
- Киресенчә, "Сез яшь." дип җавап бирә.

Чагыштыру операторлары:

JavaScriptта чагыштыру операторлары шартларны тикшерү өчен кулланыла.

Оператор	Аңлатма
===	Тигез (тигезлек һәм типны тикшерә)
!==	Тигез түгел
>	Зуррак
<	Кечкенәрәк
>=	Зуррак яки тигез
<=	Кечкенәрәк яки тигез

Мисаллар:

```
let a = 10;
```

```
let b = "10";

console.log(a === b); // false (типлар төрле)
console.log(a == b);  // true (типлар үзгәртелә)
console.log(a !== b); // true (тип һәм кыйммәт төрле)
console.log(a > 5);   // true
console.log(a <= 10); // true
```

Төшенчәләр:

- === — икесе дә бер үк типта һәм кыйммәттә булырга тиеш.
- == — JavaScript автоматик рәвештә типларны үзгәртә һәм тигезлекне тикшерә.
- !== — типлар яки кыйммәтләр төрле булганда true чыгара.

Кушылган шартлар: &&, ||

- **&&** (һәм): Барлык шартлар да дөрес булганда **true** чыгара.
- **||** (яки): Бер шарт дөрес булса да **true** чыгара.

Мисал:

```
let температура = 25;

if (температура > 20 && температура < 30) {
    console.log("һава җылы.");
} else if (температура <= 20 || температура > 30) {
    console.log("һава салкын яки бик эссе.");
}
```

Кыска форма: Тернар операторы

JavaScript шартлы операторларын кыскартылган формада кулланырга мөмкин.

Синтаксис:

```
шарт ? шарт_дөрес_булганда : шарт_ялгыш_булганда;
```

Мисал:

```
let яше = 17;
let статус = яше >= 18 ? "Өлкән" : "Яшь";
console.log(статус); // "Яшь"
```

Практик куллану:

```
let кулланучы = "Айдар";
let пароль = "1234";

if (кулланучы === "Айдар" && пароль === "1234") {
    console.log("Керү рөхсәт ителә.");
} else {
    console.log("Керү тыелган.");
}
```

Бу мисал, кулланучының исемен һәм паролен тикшереп, керү мөмкинлеген билгели.

Өстәмә киңәшләр:

- Кодны укыла торган итү: Озын шартларны аңлатма белән бүлегез яки функциягә чыгарыгыз.
- Тигезлекне тикшергәндә: **===** операторын кулланыгыз, чөнки ул типларны да тикшерә һәм көтелмәгән хаталардан саклый.
- Кушылган шартларны куллану: **&&** һәм **||** операторларын куллануны оптимальләштерегез, артык катлаулы структуралардан сакланыгыз.

Шулай итеп, JavaScriptта шартлы операторлар код логикасын җайлы һәм төгәл итеп оештырырга ярдәм итә.

3.5. Цикллар (Loops)

Цикллар JavaScriptта кодны кабатлап эшләтү өчен файдалы корал булып тора. Алар программа логикасын оптимальләштерергә, кайбер операцияләрне берничә тапкыр эшләргә мөмкинлек бирә. JavaScriptта төп цикллар булып for, while, һәм do...while тора.

for циклы

Синтаксис:

```
for (инициализация; шарт; итерация) {
    // Эшләнәчәк код
}
```

Мисал:

```
for (let i = 0; i < 5; i++) {
    console.log("Сан:", i);
}
```

Аңлатмалар:

- let i = 0; — циклның башлангыч ноктасын билгели.
- i < 5; — циклның эшләү шартын билгели. Әгәр шарт дөрес булса, цикл эшләвен дәвам итә.
- i++ — һәр цикл үткәннән соң i ны 1 гә арттыра.

Нәтиҗә:

```
Сан: 0
Сан: 1
Сан: 2
Сан: 3
Сан: 4
```

`while` циклы

Синтаксис:

```
while (шарт) {
    // Эшләнәчәк код
}
```

Мисал:

```
let i = 0;
while (i < 5) {
    console.log("Сан:", i);
    i++;
}
```

Аңлатмалар:

- while циклы шарт дөрес булган вакытта эшли.
- Циклның эчендә i ны арттыруны онытмаска кирәк, югыйсә цикл туктамыйча дәвам итәчәк (мәңгелек цикл).

Нәтиҗә:

```
Сан: 0
Сан: 1
Сан: 2
Сан: 3
Сан: 4
```

do...while циклы

Синтаксис:

```
do {
    // Эшләнәчәк код
} while (шарт);
```

Мисал:

```
let i = 0;
do {
    console.log("Сан:", i);
    i++;
} while (i < 5);
```

Аңлатмалар:

- **do...while** циклы башта бер тапкыр эшли, аннары шартны тикшерә. Шарт дөрес булса, цикл кабатлана.
- Бу цикл шарт дөрес булмаса да, иң кимендә бер тапкыр үтәлә.

Нәтиҗә:

```
Сан: 0
Сан: 1
Сан: 2
Сан: 3
Сан: 4
```

Цикллар белән эш итүгә киңәшләр:

Цикларны вакытында туктату:

Мәңгелек циклдан саклану өчен һәр циклда туктату шартын тикшерергә онытмагыз. Мәңгелек цикл программа тукталуына китерә.

Циклны вакытыннан алда туктату:

- break операторы циклны тулысынча туктата.
- continue операторы циклның хәзерге итерациясен төшереп калдыра һәм яңа итерациягә күчә.

Мисал:

```
for (let i = 0; i < 10; i++) {
```

```
    if (i === 5) {
        break; // Цикл 5 кә җиткәч туктатыла
    }
    console.log(i);
}
```

Нәтиҗә:

```
0
1
2
3
4
```

`for...in` һәм `for...of` циклларын

`for...in`

Объектның барлык ачкычларын (ключларын) кабатлый.

Мисал:

```
let кеше = { исем: "Айдар", яшь: 25 };

for (let ачкыч in кеше) {
    console.log(ачкыч + ": " + кеше[ачкыч]);
}
```

Нәтиҗә:

```
исем: Айдар
яшь: 25
```

`for...of`

Массив яки башка итерацияләнә торган объектларның элементларын кабатлый.

Мисал:

```
let саннар = [10, 20, 30];

for (let сан of саннар) {
    console.log(сан);
}
```

Нәтиҗә:

```
10
20
30
```

Йомгак:

Цикллар JavaScript программаларын төгәл һәм нәтиҗәле итә. Цикларны дөрес сайлап, логиканы оптимальләштерү һәм кодны кыскарту мөмкинлеге бар. Әгәр цикл эчендәге логика катлаулы булса, аңлатмалар язуны һәм эш этапларын төгәл билгеләүне онытмагыз.

3.6. Функцияләр (Functions)

JavaScript функцияләре кодны модульләштерү, структуралау һәм кабат куллану мөмкинлеген бирә. Аларны төрле стильдә һәм ситуациягә яраклы итеп билгеләргә мөмкин.

Гадәти функция билгеләү

Синтаксис:

```
function функцияИсеме(параметр1, параметр2, ...) {
    // Эшләнәчәк код
    return нәтиҗә;
}
```

Мисал:

```
function исәплә(a, b) {
    return a + b;
}

let нәтиҗә = исәплә(5, 3);
console.log("Нәтиҗә:", нәтиҗә); // Нәтиҗә: 8
```

Аңлатмалар:

- **function исәплә(a, b)** — исәплә исемле функция булдырабыз, ул ике аргумент кабул итә.
- **return** — функциянең нәтиҗәсен кайтара.
- **исәплә(5, 3)** — функцияне чакыру, нәтиҗә нәтиҗә үзгәрешлегенә саклана.

Аноним функцияләр

Аноним функцияләр исемсез, һәм гадәттә үзгәрешлегә билгеләнә.

Мисал:

```
let исәплә = function(a, b) {
    return a + b;
};

console.log("Нәтиҗә:", исәплә(10, 15)); // Нәтиҗә: 25
```

Аңлатмалар:

- Аноним функцияне **function(a, b)** формасында билгелибез.
- Ул исәплә үзгәрешлегенә саклана һәм кирәк булганда чакырыла.
- Объектка юнәлтелгән программалаштыруда һәм callback функцияләрендә еш кулланыла.

Уклы функцияләр (Arrow Functions)

Уклы функцияләр — кыскарак һәм җиңел язылышлы функцияләр.

Мисал:

```
let исәплә = (a, b) => a + b;

console.log("Нәтиҗә:", исәплә(7, 2)); // Нәтиҗә: 9
```

Аңлатмалар:

- => — уклы функцияләрне билгели.
- Әгәр функция бер генә операция ясый икән, **{}** һәм **return** язу кирәк түгел.
- Бердәнбер параметр булганда, параметрларны түгәрәк скобкаларсыз язып була.

Өстәмә мисал:

```
let квадрат = x => x * x;

console.log("Квадрат:", квадрат(5)); // Квадрат: 25
```

Функцияләрнең төрле кулланылышлары

Callback функцияләр

Callback функцияләр башка функцияләргә аргумент буларак тапшырыла.

Мисал:

```
function үтәү(гамәл, a, b) {
    return гамәл(a, b);
}

let кушу = (a, b) => a + b;
let тапкырлау = (a, b) => a * b;

console.log("Кушу:", үтәү(кушу, 5, 3)); // Кушу: 8
console.log("Тапкырлау:", үтәү(тапкырлау, 5, 3)); //
Тапкырлау: 15
```

Функцияләрне Default параметрлар белән билгеләү

Функция параметрлары өчен стандарт кыйммәтләр биреп була.

Мисал:

```
function исәнләшү(исем = "Дус") {
    console.log(`Исәнмесез, ${исем}!`);
}

исәнләшү(); // Исәнмесез, Дус!
исәнләшү("Айдар"); // Исәнмесез, Айдар!
```

IIFE (Immediately Invoked Function Expression)

IIFE — функцияне аны язган шунда ук эшләтеп җибәрү.

Мисал:

```
(function() {
    console.log("Бу IIFE функция.");
})();
```

Нәтиҗә:

```
Бу IIFE функция.
```

Күренешләр һәм контекст (Scope һәм `this`)

- Күренеш (Scope): Функцияләр үзләренең локаль күренешләрендә эшли, ә тышкы күренешкә турыдан-туры керә ала.
- **this**: Гадәти функцияләрдә this функция чакырылган объектка карый, ә уклы функцияләрдә this өске күренешкә бәйле.

Мисал:

```
let объект = {
    исем: "Гөлназ",
    күрсәт: function() {
        console.log(this.isem); // Гөлназ
    },
    уклы: () => {
        console.log(this.isem); // `this` өске
күренешкә карый, бу очракта undefined
    }
};

объект.күрсәт();
объект.уклы();
```

JavaScript функцияләре күп яклы кулланылыш мөмкинлекләре бирә. Аларны төрле стильдә язырга мөмкин: гадәти, аноним, уклы, IIFE. Контекст (**this**) һәм күренешләр (scope) турында аңлау функцияләрне дөрес итеп кулланырга ярдәм итә.

3.7. Массивлар (Arrays)

Массивлар JavaScriptта бертөрле (һәм кайчак төрле) мәгълүматларны тәртипле рәвештә саклау өчен кулланыла. Алар санлы индексләү ярдәмендә элементларга керергә мөмкинлек бирә һәм төрле операцияләрне тиз башкарырга мөмкин.

Массивларны билгеләү һәм эшкәртү

Мисал:

```
let исемнәр = ["Гөлназ", "Айдар", "Рөстәм"];

console.log(исемнәр[0]); // "Гөлназ"
console.log(исемнәр.length); // 3

исемнәр.push("Дамир");
console.log(исемнәр); // ["Гөлназ", "Айдар", "Рөстәм",
"Дамир"]
```

Аңлатмалар:

- **[]** — массивны билгели.
- **исемнәр[0]** — массивның беренче элементын ала.
- **.length** — массивтагы элементларның санын күрсәтә.
- **.push** — массив ахырына яңа элемент өсти.

Массив элементларын үзгәртү

Мисал:

```
let саннар = [1, 2, 3, 4, 5];
саннар[2] = 10;
console.log(саннар); // [1, 2, 10, 4, 5]
```

Аңлатмалар:

- Массив элементлары индексы ярдәмендә үзгәртелә.
- Индексация 0 дан башлана, шуңа күрә **саннар[2]** өченче элементны үзгәртә.

Массивлар белән эшләү өчен цикллар

`for` циклы белән эш итү

```
let исемнәр = ["Гөлназ", "Айдар", "Рөстәм"];

for (let i = 0; i < исемнәр.length; i++) {
    console.log(`Сезнең исемегез: ${исемнәр[i]}`);
}
```

Нәтиҗә:

```
Сезнең исемегез: Гөлназ
Сезнең исемегез: Айдар
Сезнең исемегез: Рөстәм
```

`for...of` циклы

```
for (let исем of исемнәр) {
    console.log(`Сезнең исемегез: ${исем}`);
}
```

Нәтиҗә:

```
Сезнең исемегез: Гөлназ
Сезнең исемегез: Айдар
Сезнең исемегез: Рөстәм
```

Массивларның киң таралган ысуллары

1. Элемент өстәү һәм бетерү:
    - **push** — массив ахырына элемент өсти.
    - **pop** — массивның ахыргы элементын бетерә.
    - **unshift** — массив башына элемент өсти.
    - **shift** — массивның беренче элементын бетерә.

    Мисал:

```
let саннар = [1, 2, 3];

саннар.push(4);
console.log(саннар); // [1, 2, 3, 4]

саннар.pop();
console.log(саннар); // [1, 2, 3]
```

2. Массивны кисү:
    - **slice** — массивның бер өлешен ала.
    - **splice** — массивка элемент өсти яки аннан элементларны бетерә.

    Мисал:

```
let саннар = [1, 2, 3, 4, 5];

let киселгән = саннар.slice(1, 4);
console.log(киселгән); // [2, 3, 4]

саннар.splice(2, 1, 10); // 2нче индекстагы элементны
бетереп, урынына 10 куя
console.log(саннар); // [1, 2, 10, 4, 5]
```

3. Массив элементларын эзләү:
    - **indexOf** — элементның индексын кайтара (әгәр тапмаса, -1).
    - **includes** — массивта элемент барлыгын тикшерә.

    Мисал:

```
let исемнәр = ["Гөлназ", "Айдар", "Рөстәм"];

console.log(исемнәр.indexOf("Айдар")); // 1
console.log(исемнәр.includes("Дамир")); // false
```

Массивлар белән фильтрлау һәм үзгәртү

1. **filter** — массив элементларын шарт буенча сайлый.
2. **map** — массив элементларын үзгәртеп, яңа массив ясый.

Мисал:

```
let саннар = [1, 2, 3, 4, 5];

let зурСаннар = саннар.filter(сан => сан > 2);
console.log(зурСаннар); // [3, 4, 5]

let квадратлар = саннар.map(сан => сан * сан);
console.log(квадратлар); // [1, 4, 9, 16, 25]
```

Массивлар JavaScriptта көчле һәм күп яклы корал. Аларны җиңел эшкәртергә, үзгәртергә һәм фильтрларга мөмкин. Цикллар һәм ысуллар ярдәмендә массивларны эшкәртү процессын нәтиҗәле итәргә була. Массивлар белән эшләгәндә функцияләрнең эшләү тәртибен яхшы аңлау мөһим.

3.8. Объектлар (Objects)

Объектлар JavaScriptта мәгълүматны ачкыч-мәгънә парлары рәвешендә сакларга мөмкинлек бирә, бу исә катлаулы структуралар белән эш иткәндә уңайлы.

Объектны билгеләү һәм аның белән эш итү

Мисал:

```
let студент = {
    исем: "Айдар",
    төркем: "Казан",
    яше: 20
};

console.log(студент.исем); // "Айдар"
console.log(студент["төркем"]); // "Казан"

студент.яше = 21;
console.log(студент.яше); // 21
```

Аңлатмалар:

- **{}** — объектны билгели.
- **.** һәм **[]** — объектның атрибутларына керү ысуллары.

- **.** — билгеле атрибутка керү өчен турыдан-туры ысул.
- **[]** — атрибут исеме үзгәрешле яки динамик булган очракта кулланыла.

Объектларның функционаллыгын арттыру

Объектлар функцияләрне саклый ала, бу аларга үз-үзләрен тасвирлый торган функционаллык өсти.

Мисал:

```
let студент = {
    исем: "Айдар",
    төркем: "Казан",
    яше: 20,
    танышу: function() {
        console.log(`Исәнмесез, минем исемем
${this.исем}, мин ${this.төркем} төркемендә укыйм.`);
    }
};

студент.танышу();
// Исәнмесез, минем исемем Айдар, мин Казан төркемендә
укыйм.
```

Аңлатмалар:

- **function()** — объектка функция өсти.
- **this** — объектның үз атрибутларына мөрәҗәгать итү өчен кулланыла.

Объектлар белән эш итүнең киң таралган ысуллары

1. Атрибутлар өстәү:

```
студент.телефон = "+123456789";
console.log(студент.телефон); // "+123456789"
```

2. Атрибутларны бетерү:

```
delete студент.телефон;
console.log(студент.телефон); // undefined
```

3. Барлык атрибутларны карау:

```
for (let ачкыч in студент) {
    console.log(`${ачкыч}: ${студент[ачкыч]}`);
```

```
}
```

Нәтиҗә:

```
исем: Айдар
төркем: Казан
яше: 20
```

Объектларны клонлау һәм берләштерү

1. Объектны копияләү:

```
let яңаСтудент = { ...студент };
яңаСтудент.исем = "Гөлназ";
console.log(яңаСтудент.исем); // "Гөлназ"
console.log(студент.исем); // "Айдар"
```

2. Объектларны берләштерү:

```
let өстәмәМәгълүмат = { адрес: "Казан", телефон:
"+123456789" };
let тулыМәгълүмат = { ...студент, ...өстәмәМәгълүмат };
console.log(тулыМәгълүмат);
// { исем: "Айдар", төркем: "Казан", яше: 20, адрес:
"Казан", телефон: "+123456789" }
```

JSON: Объектларны саклау һәм тапшыру

JavaScript объектлары JSON форматына күчереп саклана һәм тапшырыла ала.

1. Объектны JSON форматына күчерү:

```
let json = JSON.stringify(студент);
console.log(json);
// {"исем":"Айдар","төркем":"Казан","яше":20}
```

2. JSONны объектка әйләндерү:

```
let яңаданОбъект = JSON.parse(json);
console.log(яңаданОбъект);
// { исем: "Айдар", төркем: "Казан", яше: 20 }
```

Комплексы структуралар

Объектлар массивларны һәм башка объектларны саклый ала.

Мисал:

```
let университет = {
    факультетлар: [
        { исем: "Математика", студентлар: 120 },
        { исем: "Информатика", студентлар: 80 }
    ],
    урнашу: "Казан"
};

console.log(университет.факультетлар[0].исем); //
"Математика"
console.log(университет.факультетлар[1].студентлар); //
80
```

Объектлар JavaScriptта мәгълүматны оештыру һәм функционаллык белән баету өчен төп коралларның берсе. Алар функцияләр, массивлар, башка объектлар белән бергә кулланылганда катлаулы мәгълүмат структураларын эшкәртүне җиңеләйтә. Объектларның мөмкинлекләрен тулысынча куллану программаларның укучылылыгын һәм нәтиҗәлелеген арттыра.

3.9. Массивлар белән эш итү (Working with Arrays)

JavaScript массивлар белән эшләү өчен күптөрле методлар тәкъдим итә, алар мәгълүматны эшкәртү, фильтрлау һәм трансформацияләү процессларын җиңеләйтә. Бу методлар кодны кыскарту һәм укучылылыкны арттыру өчен файдалы.

forEach методы

Синтаксис:

```
массив.forEach(функция(элемент, индекс, массив));
```

Мисал:

```
let исемнәр = ["Гөлназ", "Айдар", "Рөстәм"];

исемнәр.forEach(function(исем, индекс) {
    console.log(`${индекс + 1}. ${исем}`);
});
```

Нәтиҗә:

```
1. Гөлназ
```

```
2. Айдар
3. Рөстәм
```

Аңлатмалар:

- **forEach** һәр массив элементына бер тапкыр функцияне куллана.
- Функция өч параметр кабул итә:
  - элемент — массивның агымдагы элементы.
  - индекс — элементның индексы.
  - массив (өченче параметр) — бөтен массив.

map методы

Синтаксис:

```
яңаМассив = массив.map(функция(элемент, индекс, массив));
```

Мисал:

```
let саннар = [1, 2, 3, 4, 5];
let квадратлар = саннар.map(function(сан) {
    return сан * сан;
});

console.log(квадратлар); // [1, 4, 9, 16, 25]
```

Уклы функция белән мисал:

```
let квадратлар = саннар.map(сан => сан * сан);
console.log(квадратлар); // [1, 4, 9, 16, 25]
```

Аңлатмалар:

- **map** массив элементларын трансформацияли һәм яңа массив кайтара.
- Эшкәртү өчен функция һәр элементка кулланыла.

filter методы

Синтаксис:

```
яңаМассив = массив.filter(функция(элемент, индекс, массив));
```

Мисал:

```
let саннар = [1, 2, 3, 4, 5];
```

```
let зурраклар = саннар.filter(function(сан) {
    return сан > 3;
});

console.log(зурраклар); // [4, 5]
```

Lambda функция белән:

```
let зурраклар = саннар.filter(сан => сан > 3);
console.log(зурраклар); // [4, 5]
```

Аңлатмалар:

- **filter** массив элементларын шартка карап сайлый һәм яңа массив кайтара.
- Шартка туры килмәгән элементлар нәтиҗә массивына керми.

reduce методы

Синтаксис:

```
җыелма = массив.reduce(функция(acc, элемент, индекс, массив), башлангычКыйммәт);
```

Мисал:

```
let саннар = [1, 2, 3, 4, 5];
let сумма = саннар.reduce(function(acc, сан) {
    return acc + сан;
}, 0);

console.log("Сумма:", сумма); // Сумма: 15
```

Уклы функция белән:

```
let сумма = саннар.reduce((acc, сан) => acc + сан, 0);
console.log("Сумма:", сумма); // Сумма: 15
```

Аңлатмалар:

- **reduce** массив элементларын бер кыйммәткә җыя.
- **acc** — аккумулятор, нәтиҗәне саклау өчен кулланыла.
- **сан** — массивның агымдагы элементы.
- **0** — башлангыч аккумулятор кыйммәте.

Методларны берләштерү

Массив методларын бергә кулланып, катлаулы эшкәртүләр ясарга мөмкин.

Мисал:

```
let саннар = [1, 2, 3, 4, 5];

let нәтиҗә = саннар
    .filter(сан => сан > 2)        // [3, 4, 5]
    .map(сан => сан * 2)           // [6, 8, 10]
    .reduce((acc, сан) => acc + сан, 0); // 24

console.log("Нәтиҗә:", нәтиҗә); // Нәтиҗә: 24
```

JavaScript массив методлары мәгълүматларны эшкәртүне гади һәм нәтиҗәле итә. Методлар:

- **forEach** — массив элементларын кабатлау өчен.
- **map** — элементларны трансформацияләү өчен.
- **filter** — шарт буенча элементларны сайлау өчен.
- **reduce** — элементларны бер кыйммәткә җыю өчен кулланыла.

Бу методларны аңлау һәм дөрес куллану программаны укучыл һәм нәтиҗәле итә.

3.10. Асинхрон программалаштыру (Asynchronous Programming)

Асинхрон программалаштыру JavaScriptның көчле якларыннан берсе булып тора. Promises һәм async/await программаның нәтиҗәле эшләвен һәм укыла торган код язуны тәэмин итә.

Promises

Promise — асинхрон операциянең киләчәктәге нәтиҗәсен белдерүче объект. Ул өч халәттә булырга мөмкин:

1. Pending (Көтү): Операция тәмамланмаган.
2. Fulfilled (Уңышлы): Операция уңышлы тәмамланган.
3. Rejected (Хаталы): Операция хата белән тәмамланган.

Мисал:

```
let промис = new Promise(function(resolve, reject) {
    let уңыш = true; // яки false
```

```
    if (уңыш) {
        resolve("Нәтиҗә уңышлы!");
    } else {
        reject("Нәтиҗә хаталы!");
    }
});

промис
    .then(function(мәгълүмат) {
        console.log(мәгълүмат); // "Нәтиҗә уңышлы!"
    })
    .catch(function(хата) {
        console.error(хата); // "Нәтиҗә хаталы!" (әгәр
уңыш === false)
    });
```

Уклы функция белән:

```
let промис = new Promise((resolve, reject) => {
    let уңыш = true;

    уңыш ? resolve("Нәтиҗә уңышлы!") : reject("Нәтиҗә
хаталы!");
});

промис
    .then(мәгълүмат => console.log(мәгълүмат))
    .catch(хата => console.error(хата));
```

Төшенчәләр:

- **`Promise`** — асинхрон операциянең үтәлеш хәлен тасвирлый.
- **resolve** — уңышлы операция өчен.
- **reject** — хаталы операция өчен.
- **then** — уңышлы нәтиҗә өчен.
- **catch** — хата булганда.

async/await

async/await — Promise белән эшләүне җиңеләйтү өчен кертелгән. Ул кодны укыла торган синхрон структурага охшата.

Мисал:

```
function эшләү() {
    return new Promise((resolve, reject) => {
```

```
        setTimeout(() => {
            resolve("Эш тәмамланды!");
        }, 2000);
    });
}

async function төп() {
    try {
        let нәтиҗә = await эшләү();
        console.log(нәтиҗә); // "Эш тәмамланды!" (2
секундтан соң)
    } catch (хата) {
        console.error(хата);
    }
}

төп();
```

Уклы функция белән:

```
const эшләү = () => {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve("Эш тәмамланды!");
        }, 2000);
    });
};

const төп = async () => {
    try {
        let нәтиҗә = await эшләү();
        console.log(нәтиҗә); // "Эш тәмамланды!"
    } catch (хата) {
        console.error(хата);
    }
};

төп();
```

Төшенчәләр:

- **async** — функция асинхрон булуын билгели һәм Promise кайтара.
- **await** — промисның үтәлешен көтә.
- **try...catch** — хаталарны тоту өчен кулланыла.

Promises һәм async/await чагыштыру

Үзенчәлек	Promises	async/await
Кодның укучылылыгы	Катлаулырак, төзү өчен күбрәк `.then`	Гади, синхрон структура кебек
Хата белән эш итү	`.catch`	`try...catch`
Куллану	Асинхрон операцияләрне аерым эшкәртү	Күп асинхрон операцияләрне эшкәртү җиңел

Асинхрон программалаштыруда катлаулы мисал

Берничә асинхрон операция башкару:

```
function мәгълүматАлу(id) {
    return new Promise(resolve => {
        setTimeout(() => {
            resolve(`Мәгълүмат ${id}`);
        }, 1000);
    });
}
```

```
async function барысынБашкару() {
    try {
        let мәгълүмат1 = await мәгълүматАлу(1);
        console.log(мәгълүмат1);

        let мәгълүмат2 = await мәгълүматАлу(2);
        console.log(мәгълүмат2);

        let мәгълүмат3 = await мәгълүматАлу(3);
        console.log(мәгълүмат3);
    } catch (хата) {
        console.error("Хата:", хата);
    }
}

барысынБашкару();
```

Нәтиҗә:

```
Мәгълүмат 1
Мәгълүмат 2
Мәгълүмат 3
```

- Promises һәм async/await асинхрон операцияләрне җиңел башкару өчен кулланыла.
- Promises берәм-берәм эшкәртү өчен яхшы.
- async/await кодны укучыл һәм структуралы итә.
- Хата белән эш иткәндә try...catch куллану уңайлы.

Асинхрон операцияләрне дөрес куллану, программа нәтиҗәлелеген арттыра һәм кулланучыларның тәҗрибәсен яхшырта.

3.11.DOM белән эш итү

DOM (Document Object Model)

DOM — веб-битнең структурасын тасвирлый торган API, ул HTML һәм XML документлары белән эш итәргә мөмкинлек бирә. DOM ярдәмендә JavaScript веб-биттәге элементларны таба, үзгәртә, яңаларын өсти һәм бетерә ала.

DOM белән эш итүнең төп методлары

```
Мисал:
<!DOCTYPE html>
<html lang="tt">
<head>
    <meta charset="UTF-8">
    <title>DOM Мисалы</title>
</head>
<body>
    <h1 id="исем">Исәнмесез!</h1>
    <button id="күзгәк">Төзәтергә</button>

    <script>
        let элемент = document.getElementById("исем");
        let төзәтергә =
document.getElementById("күзгәк");

        төзәтергә.addEventListener("click", function()
{
            элемент.textContent = "JavaScript белән
исәнмесез!";
        });
    </script>
</body>
</html>
```

Төшенчәләр:

- **`document.getElementById("исем")`** — DOM элементын аның id атрибуты буенча таба.
- **`addEventListener`** — DOM элементына вакыйга тыңлаучы өсти. Монда "click" вакыйгасы өчен функция эшләтелә.
- **`textContent`** — DOM элементының текстын үзгәртә.

DOM элементларын табу ысуллары

1. **`getElementById`**
    - **`id`** буенча элементны таба.

```
2. let элемент = document.getElementById("мисал");
```

3. **`querySelector`**
    - CSS селекторы ярдәмендә беренче туры килгән элементны таба.

```
4. let элемент = document.querySelector(".контент p");
```

5. **`querySelectorAll`**
    - CSS селекторы ярдәмендә туры килгән барлык элементларны массивсыман структурада кайтара.

```
6. let элементлар = document.querySelectorAll("div
   p");
```

7. **`getElementsByClassName`**
    - Класс исеме буенча элементларны массивсыман структурада кайтара.

```
8. let элементлар =
   document.getElementsByClassName("контент");
```

9. **`getElementsByTagName`**
    - Тэг исеме буенча элементларны таба.

```
10. let элементлар =
    document.getElementsByTagName("p");
```

DOM элементлары белән эш итү

Эчтәлекне үзгәртү

1. **`textContent`**
    - Текстны гади текст рәвешендә үзгәртә.

```
2. элемент.textContent = "Яңа текст!";
```

3. **`innerHTML`**
    - Элементның эчке HTML кодын үзгәртә.

```
4. элемент.innerHTML = "<strong>Калын текст</strong>";
```

Элемент өстәү

Мисал:

```
<!DOCTYPE html>
<html lang="tt">
<head>
    <meta charset="UTF-8">
    <title>DOM өстәмәләр</title>
</head>
<body>
    <div class="контент">
        <p>Бу бер абзац.</p>
    </div>
    <button id="өстәү">Абзац өстәү</button>

    <script>
        let төзәтергә =
document.getElementById("өстәү");

        төзәтергә.addEventListener("click", function()
{
            let яңаАбзац = document.createElement("p");
            яңаАбзац.textContent = "Яңа абзац
өстәлде!";

document.querySelector(".контент").appendChild(яңаАбзац
);
        });
    </script>
</body>
</html>
```

Элементны бетерү

1. **remove** — элементны бетерә.
2. `элемент.remove();`
3. **removeChild**
    - Ата элементтан баласын бетерә.
4. `document.querySelector(".контент").removeChild(элем ент);`

DOM вакыйгалары (Events)

Вакыйгалар тыңлау

addEventListener DOM элементлары белән төрле вакыйгалар өчен эшли.

Мисал:

```
элемент.addEventListener("click", function() {
    console.log("Элементка басылды!");
});
```

Вакыйгаларның төрләре:

- **click** — элементка басканда.
- **mouseover** — курсор элемент өстенә килгәндә.
- **keyup** — клавишаны җибәргәндә.
- **change** — форма элементының кыйммәте үзгәргәндә.

Өстәмә мисал: DOM элементлары белән эш

Элемент төсләрен үзгәртү:

```
<!DOCTYPE html>
<html lang="tt">
<head>
    <meta charset="UTF-8">
    <title>Төсләр үзгәртү</title>
</head>
<body>
    <div id="төс">Бу текстның төсен үзгәртегез</div>
    <button id="яңаТөс">Төсен үзгәртү</button>

    <script>
        let төсЭлемент =
document.getElementById("төс");
        let төймә = document.getElementById("яңаТөс");

        төймә.addEventListener("click", function() {
            төсЭлемент.style.color = "кызыл";
        });
    </script>
</body>
</html>
```

DOM — JavaScript белән веб-битләрнең структурасын идарә итү өчен төп корал.

- Элементларны табу: getElementById, querySelector.
- Эчтәлекне үзгәртү: textContent, innerHTML.
- Элементлар белән эш: өстәү, бетерү, стильләрне үзгәртү.
- Вакыйгалар: addEventListener кулланып интерактив элементлар булдыра аласыз.

DOM белән эш иткәндә, веб-битне динамик һәм кулланучылар белән үзара эш итә торган итәргә мөмкин.

3.12. Тәртипкә китерү (Sorting)

JavaScriptта массивларны тәртипләү (`sort` методы)

`sort` методы массивны урынында (in-place) тәртипкә китерә һәм шул ук массивны кайтара. Ул текстларны һәм саннарны тәртипкә китерү өчен кулланыла, ләкин саннарны дөрес тәртипләү өчен өстәмә функция таләп ителергә мөмкин.

Гади кулланылыш

Мисал: Текстларны тәртипләү

```
let исемнәр = ["Гөлназ", "Айдар", "Рөстәм", "Дамир"];
исемнәр.sort();
console.log(исемнәр); // ["Айдар", "Дамир", "Гөлназ",
"Рөстәм"]
```

Мисал: Саннарны тәртипләү (текст буларак)

```
let саннар = [4, 2, 5, 1, 3];
саннар.sort();
console.log(саннар); // [1, 2, 3, 4, 5]
```

Саннарны дөрес тәртипкә китерү

`sort` методы саннарны тәртипкә китергәндә, аларны текст буларак карый. Шуңа күрә саннарны чын тәртиптә урнаштыру өчен махсус тәртипләү функциясе кирәк.

Мисал: Тәртипләү функциясе белән

```
let саннар = [4, 2, 5, 1, 3];
саннар.sort(function(a, b) {
    return a - b; // Кимү тәртибе өчен b - a
});
console.log(саннар); // [1, 2, 3, 4, 5]
```

Уклы функция белән:

```
let саннар = [4, 2, 5, 1, 3];
саннар.sort((a, b) => a - b);
console.log(саннар); // [1, 2, 3, 4, 5]
```

Төшенчә:

- **a - b** уңай сан кайтарса, а зуррак.

- **b - a** уңай сан кайтарса, b зуррак.

Тәртипләү функцияләренең катлаулы кулланылышы

Мисал: Кимү тәртибе

```
let саннар = [4, 2, 5, 1, 3];
саннар.sort((a, b) => b - a);
console.log(саннар); // [5, 4, 3, 2, 1]
```

Мисал: Объектларны тәртиплә

```
let студентлар = [
    { исем: "Гөлназ", баллар: 85 },
    { исем: "Айдар", баллар: 92 },
    { исем: "Рөстәм", баллар: 78 }
];

студентлар.sort((a, b) => b.баллар - a.баллар);
console.log(студентлар);
/*
[
    { исем: "Айдар", баллар: 92 },
    { исем: "Гөлназ", баллар: 85 },
    { исем: "Рөстәм", баллар: 78 }
]
*/
```

Тәртипләү функциясенең мөмкинлекләре

Мисал: Текстлар һәм саннарны бергә тәртиплә

```
let элементлар = ["10", "1", "2", "А", "Б"];
элементлар.sort((a, b) => {
    if (isNaN(a) && isNaN(b)) {
        return a.localeCompare(b); // Текстларны
алфавит буенча тәртипләү
    }
    return a - b; // Саннарны санча тәртипләү
});
console.log(элементлар); // ["1", "2", "10", "А", "Б"]
```

Locale-ка бәйле тәртипләү

localeCompare методы телгә бәйле тәртипләү өчен кулланыла.

Мисал: Татар телендәге сүзләр

```
let сүзләр = ["Яңа", "Айдар", "Гөлназ", "Әлмәт"];
сүзләр.sort((a, b) => a.localeCompare(b, "tt"));
```

```
console.log(сүзләр); // ["Айдар", "Гөлназ", "Әлмәт",
"Яңа"]
```

- **sort** методы массивны урыннарында тәртипкә китерә.
- Саннар өчен махсус функция куллану кирәк.
- Кастом функцияләр объектлар һәм катлаулы структураларны тәртипләү өчен файдалы.
- **localeCompare** телгә бәйле текстларны тәртипкә китерү өчен иң яхшы ысул.

Бу мөмкинлекләр программаны индивидуаль логика таләпләренә туры китереп эшләргә булыша.

3.13.JavaScriptның ES6 (ECMAScript 2015) яңалыклары

ES6 JavaScriptта программалаштыру мөмкинлекләрен киңәйтә һәм кодны гади, уку җиңелрәк итә торган күп яңа үзенчәлекләр кертте.

Template Literals (Шаблон Литераллары)

Template Literals — динамик текстлар һәм күп юллы юлламалар язуны җиңеләйтә.

Мисал: Динамик текст

```
let исем = "Айдар";
let яшь = 20;
console.log(`Исәнмесез, минем исемем ${исем}, минем
яшем ${яшь}.`);
// Исәнмесез, минем исемем Айдар, минем яшем 20.
Күп юллы текст
let текст = `Бу
күп
юллы
текст.`;

console.log(текст);
// Бу
// күп
// юллы
// текст.
```

Төшенчәләр:

- `` (гравис) — шаблон литераллары өчен кулланыла.
- **${}** — JavaScript выражениеләрен текстка кертү өчен.

Destructuring (Деструктуризация)

Destructuring — объектлардан яки массивлардан мәгълүматны аерып алырга мөмкинлек бирә.

Объектлар белән мисал:

```
let студент = {
    исем: "Гөлназ",
    төркем: "Казан",
    яше: 19
};

let { исем, төркем } = студент;
console.log(исем);   // "Гөлназ"
console.log(төркем); // "Казан"
Массивлар белән мисал:
let саннар = [1, 2, 3];

let [беренче, икенче, өченче] = саннар;
console.log(беренче); // 1
console.log(икенче);  // 2
console.log(өченче);  // 3
```

Төшенчәләр:

- Объектлар: {} структурасын куллану.
- Массивлар: [] структурасын куллану.

Spread Operator (Спред Оператор)

`...` операторы массивларны яки объектларны ачу һәм берләштерү өчен кулланыла.

Массивлар белән мисал:

```
let беренче = [1, 2, 3];
let икенче = [4, 5, 6];
let барысы = [...беренче, ...икенче];
console.log(барысы); // [1, 2, 3, 4, 5, 6]
Объектлар белән мисал:
let студент = {
    исем: "Айдар",
    төркем: "Казан"
};

let яңартылганСтудент = {
    ...студент,
```

```
    яше: 21
};

console.log(яңартылганСтудент);
// { исем: "Айдар", төркем: "Казан", яше: 21 }
```

Төшенчәләр:

- **...** — массивларны берләштерү һәм объектларны киңәйтү өчен уңайлы.

Classes (Класслыр)

ES6 класслары объектларга юнәлтелгән программалаштыруны җиңеләйтә.

Мисал: Гади класс

```
class Студент {
    constructor(исем, төркем) {
        this.исем = исем;
        this.төркем = төркем;
    }

    танышу() {
        console.log(`Исәнмесез, минем исемем
${this.исем}, мин ${this.төркем} төркемендә укыйм.`);
    }
}

let студент = new Студент("Рөстәм", "Казан");
студент.танышу();
// Исәнмесез, минем исемем Рөстәм, мин Казан төркемендә
укыйм.
```

Уклы функцияләр белән:

```
class Студент {
    constructor(исем, төркем) {
        this.исем = исем;
        this.төркем = төркем;
    }

    танышу = () => {
        console.log(`Исәнмесез, минем исемем
${this.исем}, мин ${this.төркем} төркемендә укыйм.`);
    }
}

let студент = new Студент("Гөлназ", "Чаллы");
```

```
студент.танышу();
// Исәнмесез, минем исемем Гөлназ, мин Чаллы төркемендә укыйм.
```

Төшенчәләр:

- **class** — класс билгеләү.
- **constructor** — объектның башлангыч хәлгә китерү функциясе.
- **this** — объектның үз үзенчәлекләренә мөрәҗәгать итү.
- Методлар: Класста билгеләнгән функцияләр.

Йомгак

ES6 мөмкинлекләре:

1. Template Literals — текстлар белән эшне җиңеләйтә.
2. Destructuring — объектлар һәм массивлардан мәгълүматны уңайлы чыгара.
3. Spread Operator — массивларны һәм объектларны берләштерү өчен көчле инструмент.
4. Classes — объектларга юнәлтелгән программалаштыруны структуралы итә.

ES6 яңалыклары JavaScriptны заманча һәм код язу өчен уңайлы итә, укыла торган һәм кыскартылган программалар төзүгә ярдәм итә.