КЫРГЫЗ РЕСПУБЛИКАСЫНЫН БИЛИМ БЕРҮҮ ЖАНА ИЛИМ МИНИСТИРЛИГИ
Б. ОСМОНОВ АТЫНДАГЫ ЖАЛАЛ-АБАД МАМЛЕКЕТТИК УНИВЕРСИТЕТИ

«Рассмотрено»
На заседании методсовета
протокол № 1
от «5» 09 2022-ж.
методист Аширкулова Г.Т

«Согласовано»
Директор КГТКЭиИ
Матосмонов Ж.Дж «___»
«___» _____ 2022-ж.

ОКУУ МАТЕРИАЛДАРЫ

Окуу модулу 1

"Коюлган тапшырманы формалдаштыруу"

Адистиги: 230109 "Эсептөө техникаларын жана автоматташтырылган системаларды программалык камсыздоо"

Жалпы саат – 2 кредит (60 с)

Алардын ичинен:

Аудиториялык:	**36 саат**
Өз алдынча иш:	**24 саат**

Кызыл-Кыя 2022-2023

Мазмуну

1. Глоссарий

Алгоритм – алдыга коюлган максатка жетүүдө аткарыла турган жумуштардын ирээти менен аткарылуусу

Блок схема – геометриялык фигуралардын жардамында алгоритмди схемалык көрүнүштө түзүү

Бутактануу – биринчи же экинчи операторду шартка карай тандоо

Модулдарды импорттоо - модулдун иштешине мүмкүнчүлүк алуу үчүн, аны программага импорттоо керек.

Операция –маалыматтар үсүндө аткарылган кандайдыр бир амалдар.

Өзгөрүлмө – программанын иштөө процессинде өз маанисин өзгөрткөн чоңдуктар

Программа – белгилүү бир аткаруучу үчүн инструкциялардын түрлөрү

Программалоо тили – бул программаларды жазуу үчүн (адатта ЭЭМ үчүн) арналган формалдуу тил

Программалоодогу функция - коддун өзүнчө бир бөлүгү, аны аталган аты менен атоого болот.

Сап бул- бирдик же экилик тырмакчага алынган символдордун катары.

Табигый тил – оозеки сөз түрүндөгү алгоритмдин түрү

Транслятор – программалык кодду тигил же бул программалоо тилинен машиналык кодко которгон атайын программа

Формалдаштыруу – формалдык тилдердин жардамы менен маалыматтык моделдерди куруу процесси.

Цикл – берилген шартка карап максатка жеткенге чейин кайталоо

Def – функцияларды аныктоочу оператор

Input() - маалыматты кийирүүчү функция

Python - бул чечмеленген программалоо тили

Print() – маалыматты экранга чыгаруучу функция

Range() – диапазон функциясы

While – циклдык программада колдонулуучу оператор

2. Кыскартылган сөздөрдүн тизмеси

ЭЭМ- электрондук эсептөөчү машина

ОС-операциондук система

3.Жумушчу программа (силлабус)

1. Окуу модулу жөнүндө маалымат

Окуу модулунун аталышы: ОМ 1 “Коюлган тапшырманы формалдаштыруу”.

Дисциплина: Алгоритмдештирүүнүн негиздери жана программалоо”

Жалпы саат: 2 кредит (60 саат)

Окутуу модулунун жадыбалы: 4 семестр – жумасына 2 аудиториялык саат

2. Окутуучу жөнүндө маалымат

Исманова Тынара Кубанычбековна – Жалал-Абад колледжинин “Табигый – техникалык” бөлүмүнүн БАС, ПОВТАС адистигинин кесиптик дисциплиналар окутуучусу.

3. Адабияттар

Негизги адабияттар

1. Гуриков, С.Р. Основы алгоритмизации и программирования на Python: Учебное пособие / С.Р. Гуриков. - М.: Форум, 2018. - 384 с.
2. Изучаем программирование на Python. 2-е издание. Пол Бэрри
3. Колдаев, В.Д. Основы алгоритмизации и программирования: Учебное пособие / В.Д. Колдаев. - М.: Форум, 2015. - 352 с.
4. Парфилова, Н.И. Программирование: Основы алгоритмизации и программирования: Учебник / Н.И. Парфилова; Под ред. Трусова Б.Г. - М.: Academia, 2018. - 32 с.
5. Семакин, И.Г. Основы алгоритмизации и программирования: Учебник / И.Г.
6. Семакин, И.Г. Основы алгоритмизации и программирования: Учебник / И.Г. Семакин. - М.: Academia, 2017. - 328 с.

Кошумча адабияттар

1. Серкова, Е.Г. Основы алгоритмизации и программирования: практикум / Е.Г.
2. Серкова, Е.Г. Основы алгоритмизации и программирования (ОП.04): практикум / Е.Г. Серкова. - Рн/Д: Феникс, 2017. - 159 с.
3. Фризен, И.Г. Основы алгоритмизации и программирования (среда PascalABC.Net): Учебное пособие / И.Г. Фризен. - М.: Форум, 2018. - 784 с.
4. Основы алгоритмизации и программирования: лабораторный практикум: практикум

4. Предметтин пререквизити

Бул предметти өздөштүрүүдө, негизги базалык курс, “Информатика”, “Кесиптик математика” дисциплиналары өздөштүрүлүшү керек. Студенттерди компьютер менен иштөөгө үйрөткөн, компьютерди колдонууну, программаларды жүктөөнү, иштетүүнү өздөштүрүшкөн. Ал эми математика предметинен формулаларды, математикалык чоңдуктарды эске салышкан, эсептөөлөрдү жүргүзүшкөн.

5. Предметтин постреквизити

УМ 2 “Алгоритмизация поставленной задачи”

УМ 3 “Написание программного кода с использованием языков объектноориентированного программирования, определения и манипулирования данными”

УМ 6 “Работа с системой контроля версий”

УМ 7 “Оформление программного кода в соответствии с установленными требованиями”

УМ 8 “Проверка и отладка программного кода” **Окуу модулун сүрөттөө**

Окутуу модулунун окутуунун максаты: берилген тапшырманы аткарууда алгач анын алгоритмин түзүүнү, формализациялоону жана программа түзүүнүн методикасын изилденип үйрөнүү.

Окутуу модулунун окутуунун натыйжасы: алгоритмдерди түрдүү ыкмада түзө алат; программалоо тилдерин билет; тиешелүү программалоо тутумунда иштей алат; жогорку деңгээлдеги программалоо тилдерин колдонуп программа түзө алат.

Колдонуу чөйрөсү: Бэккенд ишеп чыгуучулардын арасында колдонулат

Окуу модулун окутуу методу: лекциялык, лабораториялык сабактар

Окуу модулун үйрөнүү методдору: лабораториялык иштерди жана студенттердин өз алдынча иштерин аткаруу.

6. 1-Окуу модул боюнча тематикалык план:

Темалар	**Баары**	**Аудиториялык иш**	**Студенттин өз алдынча иштери**
	сааттар	сааттар	сааттар
Бөлүм 1: Формалдаштыруу жана алгоритм түшүнүгү	6	4	2
Бөлүм 2: Python программасы	54	32	22
Жалпы	**60 (2 кредит)**	**36**	**24**

7. Курстун политикасы

Коомдук жайларда, колледжде жүрүм-турумдун жалпы этикалык стандарттарын сактоо; сабакты жүйөлүү себепсиз калтырбоо; сабакка кечикпөө; сабакка активдүү катышуу, тапшырмаларды аткаруу жана өз убагында тапшыруу; сабак учурунда телефонду колдонбоо (эгерде сабакка керектүү болбосо);

8. Баалоо

Студенттердин билимин баалоо

"ЖАМУнун студенттерин учурдагы текшерүү жана орто аралык аттестацияны жүргүзүү жөнүндөгү жобосунун", "ЖАМУнун студенттеринин билимин компьютердик тестирлөө аркылуу тестирлөө аркылуу текшерүү жөнүндөгү жобосунун","Студенттердин билимин текшерүүнүн модульдук- рейтингдик системасы жөнүндө жобонун" негизинде жүргүзүлөт.

Баалоонун модулдук-рейтинг системасын уюштуруу

1). Студенттердин билимин текшерүү эки этаптан турат:

- модуль (учурдагы текшерүү, студенттердин өз алдынча иштери жана аралыктагы текшерүүнүн көрсөткүчтөрү); - жыйынтыктоочу текшерүү (экзамен);

Семестр ичиндеги модулдун саны окуу планындагы лекциялык сааттын жумалык жүктөмүнөн көз карандысыз бардык предметтер боюнча 2 модулду түзөт.

2). Модуль деканат тарабынан бекитилген модулдун жадыбалы боюнча лектордун жана ассистенттин катышуусу менен жүргүзүлөт. Модулдун ыргак боюнча алынышына жана билимдин ачык-айкын туура бааланышына лектор жана практикадан берген окутуучу бирдей жооптуу.Модуль - бул учурдагы текшерүү (УТ), студенттердин өз алдынча иштери (СӨАИ) жана аралыктагы текшерүүнүн (АТ) көрсөткүчтөрүнөн турат.

3). Учурдагы текшерүүдө (УТ) ар бир дисциплинанын жумушчу программасындагы критерийлер жана көнүмдөргө ылайык студенттердин тапшырмаларды аткаруусу, өтүлгөн материалдарды өздөштүрүүсү, сабактардагы катышуусун, жетишүүсүн эске алуу менен 010 го чейинки балл менен бааланып, окутуучу тарабынан модуль башталганга чейин коюлат. Учурдагы текшерүү (УТ) бааланбаса (же окутуучу тарабынан 0 балл коюлса) студент текшерүүгө (окутуучу тарабынан) киргизилбейт. Модулдагы учурдагы текшерүүдөгү "0"

баллы коюлат, эгер студент ошол предметтке бир жолу да катышпаса жана предметтен эч кандай баа албаса.

Учурдагы текшерүүнүн 10 баллынын 5 баллы студенттин сабактарга катышуусуна берилет. Сабактарга толук катышкан студентке 5 балл берилет, активдүүлүгүнө да жалпы 5 балл берилет.

Студент модул тапшыруу учурунда модул алуучу окутуучунун кабары жок модулдун графигинен башка убакытта, башка жайда тапшырып балл алган учурда, предмет боюнча модул алган окутуучу тарабынан бул жагдай жөнүндө факультеттин деканы (колледждин директору) тарабынан тастыкталып баллды жокко чыгаруу жөнүндө билдирүүсү менен окуу бөлүмүнө кайрылат.

Учурдагы текшерүүнүн (УТ) туура бааланышына, өз учурунда коюлушуна факультеттин деканына (колледждин директоруна) милдеттендирилет.

4). Студенттердин өз алдынча иштеринин мазмуну предметтер боюнча түзүлгөн жумушчу программада камтылат. Өз алдынча иш дисциплинага берилген жалпы сааттын 60% (окуу пландарына жараша) көлөмүн түзөт. Студенттин өз алдынча иштерин ***аткаруу*** (СӨАИ) ***сабак учурунда*** предметтин өзгөчөлүгүнө жараша ар кандай формада аныкталат:

- дисциплина боюнча практикалык тапшырмаларды аткаруу;
- дисциплина боюнча лабораториялык тажрыйбаларды жүргүзүү;
- сунуш кылынган текст боюнча конспект же доклад жазуу;
- жекече үй тапшырмаларды аткаруу;
- статья, эссе, тезис, илимий окуу басмалары, баяндама, буклет, презентация жасоо.

Студенттердин өз алдынча иштерин окуу процесси учурунда аткаруу үчүн окутуучулар тарабынан иштелме иштелип чыгат. Анда ӨАИ темасы, аткаруу формасы, тартиби, тапшырмалар, сааты, адабияттар берилет. Иштелме ар бир студентке берилет. 5)**.** *Модулдарды (экзамендерди) тапшыруу* компьютердик тестирлөө аркылуу жүргүзүлөт. *Бул учурда* аралыктагы текшерүү (АТ) жана студенттин өз алдынча иштери (СӨАИ) бириктирилип, окуу процессинин графигине ылайык жүргүзүлүп, 0-50 баллга чейин

бааланат.

6).Окуу планындагы ар бир дисциплина боюнча текшерүүнүн жыйынтыгы экзамен менен берилет. Сессиянын башталышына бир ай калганда экзамендик жадыбал деканат тарабынан түзүлүп, атайын доскага илинет. Экзамендик сессиянын алдындагы жумада курстук иш, долбоорлор тапшырылат. Курстук иштер жана долбоорлор, практикалар, өз алдынча дисциплина катары кабыл алынып, балл коюлуп бааланат жана AVN маалымат системасына катталат.

Академиялык карызды жоюу тартиби тиешелүү кабыл алынган жобо боюнча жүргүзүлөт.

7). Модуль компьютердик тестирлөө аркылуу жүргүзүлүп, ар бир модулдун максималдык баллы 60 ка барабар болот.

Сабактардын жыйынтык баллы (экзамен) сынак китепчеге жана экзамендик (зачеттук) ведомостко төмөндөгү шкала менен коюлат:

61-73 балл – «канаттандыраарлык»	«3»;
74-86 балл – «жакшы»	«4»;
87-100 балл - «эң жакшы»	«5»

Студенттин модулдагы арифметикалык орточо баллы **31-60** болсо, ал жыйынтыктоочу текшерүүгө киргизилип, билимине жараша 0-40 балл ала алат.

Студенттин жалпы блоктогу арифметикалык орточо баллы **0-30** болсо, ал жыйынтыктоочу текшерүүгө киргизилбейт, жайкы семестрге калтырылат**.**

Дифференциалдык зачет жана зачеттун баалоо шкаласы бирдей болуп, дифференциалдык зачет “баа” түрүндө, ал эми зачет “өттү” деп коюлат. Шкала төмөндөгүдөй болот:

Дифференциалдык зачет:

0-40 – «канаттаандыраарлык эмес» - “2”
41-48 – «канаттаандыраарлык» - “3”
49-55 – «жакшы» - “4”
55-60 – «эң жакшы» - “5”

Учурдагы текшерүүнүн (модулдун) баллын жогорулатуу үчүн кайрадан тапшырууга уруксат берилбейт. Учурдагы текшерүүгө келбеген (киргизилбеген) студентке «кж» белгиси коюлат. Студент биринчи модулга кандайдыр бир жүйөлүү себептер менен келбей калса (киргизилбесе) тастыктоочу документтердин негизинде кийинки модулга эки жума калганда факультеттин деканы тарабынан уруксат берилет жана кафедра тарабынан модулду тапшырууга жекече график түзүлөт, ал эми модулга себепсиз келбей калса: 1) деканат тарабынан сабакка катышуусу анализденип, окуудан чыгарууга сунушталат; 2) ушул Жобонун 4.9 пунктунун негизинде жыйынтыктоочу текшерүүгө киргизилбейт. Экинчи модулга же жыйынтыктоочу текшерүүгө кандайдыр бир жүйөлүү себептер менен келбей калган студенттерге модулдарды тапшырууга уруксат тастыктоочу документтеринин негизинде “Окуу процессин окутуунун кредиттик технологияларынын (ECTS) негизинде уюштуруу” жөнүндө жобого ылайык сессиядан кийин академиялык карыздарды жоюу мезгилинде берилет.

Компьютердик тестирлөө учурунда (модуль, экзамен) сабак берген окутуучу сөзсүз катышып, аудиториядагы тартипке жана экзамендерди кабыл алуу эрежелерин сактоого милдеттүү.

5. Окуу модулу боюнча лекциялык курс (окуу материалдары)

№	Бөлүмдөрдүн жана темалардын аталышы	Тема боюнча сааттардын көлөмү				Сабак №	Сабактар боюнча СӨАИ нин тапшырмалары
		Жалпы	Аудиториялык		СӨАИ		
			Лек.	Лаб.			
	Бөлүм 1. Формалдаштыруу жана алгоритм түшүнүгү						
1	ЭЭМде мисалды аткаруу этаптарын өздөштүрүү	2	2			1 (лек.)	
2	Алгоритм түзүү жөнүндө жалпы маалымат	4	2		2	2 (лек.)	**2-сабак. Үй тапшырмасы:** 1. Алгоритмдин түрлөрүнө мисалдар келтирүү. 2. Программалоо тилдерине хронологиялык таблица түзүү
	ЖЫЙЫНТЫК:	**6**	**4**		**2**		
	Бөлүм 2. Python программалоо тили						
3	Python программалоо тили менен таанышуу			2	2	3 (лек)	**3-сабак. Үй тапшырмасы:** Python интерпретаторунда интерактивдүү режимде бир нече жөнөкөй математикалык амалдарды аткаруу. Жыйынтыгын түшүндүрүп берүү.
4	Маалыматтар жана алардын түрлөрү		2			4 (лек.)	
5	Маалыматтарды киргизүүчү жана чыгаруучу функциялар			2	2	5 (лаб.)	**5-сабак. Үй тапшырмасы:** Колдонуучудан төрт номерди сураңыз. Биринчи экөөнү өзүнчө, экинчисин өзүнчө эсептеңиз. Биринчи сумманы экинчисине бөлүңүз. Жыйынтыгын ондуктан кийин эки цифра камтылгандай кылып экранда көрсөтүңүз.
6	Логикалык туюнтмалар жана операторлор			4	4	6 (лаб.), 7 (лаб.)	**6-7-сабактар. Үй тапшырмасы:**

							Колдонуучудан эки сан сураган жана биринчи сан экинчисинен чоң же чоң эместигине жараша True же False чыгарган программа түзүү.
7	Программалоонун бутактануу түрү. Шарттуу оператор		2			8 (лек.)	

8	Программалоонун бутактануу түрү. Бир нече бутактануу ifelif-else			2	2	9 (лаб.)	**9-сабак. Үй тапшырмасы:** Кандайдыр бир сан кийиргенде ал нөлдөн чоң болсо, анда экранга жооп иретинде 1 саны көрсөтүлгөн, эгерде киргизилген сан оң болбосо, анда экранда -1 көрсөтүлгөн программаны түзүү.
9	Циклдык For оператору		4		2	10 (лек.), 11 (лек)	**11-сабак. Үй тапшырмасы:** У=х>10 мисалга For операторун колдонуп программа түзүү.
10	Программалоодогу циклдар. While циклы			2	2	12 (лаб.)	**12-сабак. Үй тапшырмасы:** While циклин колдонуп, 2 саны үчүн анын кубаттуулугун 0дон 20га чейин көрсөтүп бер.
11	Каталар жана өзгөчө учурлар		2		2	13 (лек.)	**13-сабак. Үй тапшырмасы:** Эки маанини кийирүүнү сураган программа . Эгерде алардын бирөө сан болбосо, анда, бириктирүү жүргүзүлүшү керек. Экөө тен сан болгон учурда кошулушу керек болгон программа түзүү.
12	Программалоодогу функциялар			4	4	14 (лаб.), 15 (лаб.)	**14-15-сабактар. Үй тапшырмасы:** Үч бурчтуктун жана беш бурчтуктун периметрин табуунун программасын түзүү.
13	Функциядан маанилерди кайтаруу. Return оператору			2	2	16 (лаб.)	**16-сабак. Үй тапшырмасы:** Киргизилген саптын узундугун өлчөгөн программа жазыңыз. Эгер он белгиден узун сап болсо, анда эскертүү берилет.
14	Модулдар		2			17 (лек.)	**17-сабак. Үй тапшырмасы:** төрт бурчтуктун, үч бурчтуктун жана эллипстин аянттарын эсептөө үчүн функциялары менен модуль түз.
15	Саптар		2			18 (лек.)	**18-сабак. Үй тапшырмасы:** Split() жана join() методдоруна мисал келти рип программа түз.
ЖЫЙЫНТЫК:			**14**	**18**	**22**		
ЖАЛПЫ:		**60**	**18**	**18**	**24**		

БААЛОО КАРАЖАТТАРЫ

№	Баалоонун түрлөрү	Баалоо критерийлери	Балл (бир темага)
1.	Текшерүү иши	Ишти анализдөө жана чыгаруу ыкмаларын аныктоо	0,5-1
		Туура ыкманы колдонуу	0,5-1
		Жыйынтык балл	**1-2 балл**
2.	Жумушчу тетрадь	Лекцияларды туура жана түшүнүктүү жазуу	0,1
		Лабораториялык иштердин туура жана толуктугу	0,1
		Өз алдынча иштин жазылыштары	0,1
		Үй тапшырмаларынын аткарылуусу	0,2
		Жыйынтык балл	**0,5 балл**
3.	Кырдаалдык мисал маселелер	Кырдаалды анализдөө	0,1
		Берилгендерди талдоо үчүн ыкмалары тандоо	0,1
		Программа түзүү жолдорун аныктоо	0,1
		Туура түзүүнүн эң ыңгайлуу жолун тандоо	0,2
		Жыйынтык балл	**0,5 балл**
4.	Аңгемелешүү	Аңгемелешүүнүн максатын түшүнүүсү	0,1
		Суроолорго жооп берүүсү жана суроо бере билүүсү	0,1
		Аңгемелешүү темасындагы практикалык жана теориялык билимдери	0,3
		Жыйынтык балл	**0,5 балл**
5.	Чыгармачылык тапшырма	Тапшырманы аткаруу планы	0,1
		Берилген темадагы түшүнүктөрдүн аныктоолорун билүүсү	0,1
		Тема боюнча программа түзүү	0,1
		Тема боюнча алгоритм тузүү	0,1
		Түшүнүктөрдүн арасындагы байланыштар	0,1
		Жыйынтык балл	**0,5 балл**
6.	Тест	Ар бир туура жоопко	0,1-0,2
		Жыйынтык балл	**1-2 балл**
		Бардыгы:	**5-6 балл**

Бөлүм 1. Лекция №1. Тема: ЭЭМде мисалды аткаруу этаптарын өздөштүрүү Тема 1. Киришүү 1-сабак (лекция)

Негизги суроолор: формализация түшүнүгү. Сыпаттамалык моделди математикалык түргө өткөрүү. ЭЭМде берилген маселени аткарууда этаптарга бөлүп кароо

Студенттердин мурунку алган билимдери: “Информатика”, “Кесиптик математика”

Окуунун натыйжасы:

Студенттер формалдаштыруу түшүнүгүн, тапшырманы формалдаштырууну, берилген маселени этаптарга бөлүп аткарууну билишет. **Тема боюнча маалымат:** Формалдаштыруу түшүнүгү

Формалдаштыруу - бул илимий теориянын логикалык өзгөчөлүктөрүн, дедуктивдик жана көркөмдүк мүмкүнчүлүктөрүн изилдөө максатында түшүнүктөрдүн маанисинен жана сөз айкаштарынан ажыратууну камсыз кылган таанып-билүү иш-аракеттеринин жыйындысы.

1-сүрөт

Сөздүн кеңири маанисин алып караганда формалдаштыруу бул тааныш формада каалагандай программалоо тилдеринин жардамында алардын мазмунун жана түзүлүшүн чагылдыруу жолу менен ар түрдүү объекттерди изилдөө методу. Анын ичине, мисалы, математика тили, математикалык логика, химия, радио жана башка кээ бир илимдер кирет. Бул илимдердеги атайын белгилерди изилдөө - бул чындыкты чагылдыруунун зарыл жана барган сайын прогрессивдүү ыкмаларынын бири.

Биз формалдаштыруу элементтери менен мектепте эле кездешип жатабыз, кандайдыр бир маселени чечүүдө белгисиз нерсенин конкреттүү мазмунунан алагды болуп, аны теңдемедеги "X" түрү катары эсептейбиз. Формалдаштыруу ыкмасынын күчүн сезүү үчүн бул жетиштүү.

2-сүрөт

Бул методдун артыкчылыктары төмөнкүлөр:

- формалдаштыруу маселенин белгилүү бир бөлүгүнүн толук обзорун, аларды чечүүгө жалпыланган мамилени камсыз кылат. Формалдаштыруу маселелердин бүт класстарын чечүүнүн жалпы алгоритмин табууга мүмкүндүк берет;

- формалдаштыруу методу атайын белгилерди колдонууга негизделет, аларды киргизүү билимдин белгиленишинин кыскалыгын жана тунуктугун камсыз кылат (математикалык жана физикалык теориялардын элеганттуулугу, алардын компакттуулугу);

Эгерде формалдаштыруу туура жүргүзүлсө, объектинин белгиленген моделинде эң негизгиси чагылдырылса, анда бул моделди изилдөө объект жөнүндө баалуу маалыматтарды берип, атүгүл көрүнүктүү жаңы ачылыштарга алып келиши мүмкүн.

Формалдаштыруу методу жөнүндө айтып жатып, анын көптөгөн башка методдор менен тыгыз байланышта экендигин баса белгилөө керек: моделдөө, абстракциялоо, идеалдаштыруу ж.б.

Формалдаштыруу методу объектинин мазмунундагы негизги нерсе туура аныкталганда жана анын маңызы ийгиликтүү колго алынганда натыйжалуу болот. Мунсуз, символиканын эң чебер формалдуу манипуляциясы дагы натыйжасыз болот же жалган жыйынтыкка алып келет.

Сыпаттамалык моделди математикалык түргө өткөрүү

Оозеки же тексттик маалымат моделин түзүү үчүн табигый тилдер колдонулат. Илим тарыхында көптөгөн сүрөттөөчү маалыматтык моделдер белгилүү. Мисалы, Коперник тарабынан сунуш кылынган дүйнөнүн гелиоцентрдик модели төмөнкүдөй формулировка кылынган:

- Жер өз огунун айланасында жана Күндүн айланасында айланат; -

бардык планеталардын орбиталары күндү айланып өтөт.

Формалдуу маалымат моделдери (математикалык, логикалык ж.б.) формалдуу тилдердин жардамы менен курулат. Формалдуу тилдерди колдонуп маалыматтык моделдерди түзүү процесси формалдаштыруу деп аталат.

Кеңири колдонулган формалдык тилдердин бири - математикалык. Математикалык түшүнүктөрдү жана формулаларды колдонуп түзүлгөн моделдер математикалык моделдер деп аталат. Математика тили - формалдык тилдердин жыйындысы; алардын айрымдары жөнүндө (алгебралык, геометриялык) мектеп программасынан белгилүү болсо, билим алуу учурунда башкалары менен да кеңири таанышабыз.

Алгебра тили чоңдуктардын ортосундагы функционалдык байланыштарды формалдаштырууга мүмкүндүк берет. Ньютон, Коперник дүйнөсүнүн гелиоцентрикалык тутумун механика мыйзамдарын жана бүткүл дүйнөлүк тартылуу мыйзамын ачып, аларды алгебралык функционалдык көзкарандылык түрүндө жазуу менен формалдаштырды. Мектептин физика курсунда изилденүүчү кубулуштардын же процесстердин математикалык моделдери болгон алгебра тили менен чагылдырылган көптөгөн ар кандай функционалдык көзкарандылыктар каралат.

Логикалык алгебранын тили формалдуу логикалык моделдерди курууга мүмкүндүк берет. Проекциялык алгебранын жардамы менен табигый тилде айтылган жөнөкөй жана татаал сунуштар формалдаштырылат (логикалык туюнтмалар түрүндө жазылган). Логикалык моделдерди куруу менен, логикалык маселелерди чечүү, компьютердик шаймандардын логикалык моделдерин түзүү (сумматор, триггер) ж.б.у.с. Курчап турган дүйнөнү таанып-билүү процессинде адамзат ар дайым моделдөөгө жана формалдаштырууга муктаж.

ЭЭМде берилген тапшырманы аткаруудагы 4 этап

Компьютерде берилген тапшырманы аткаруу төмөнкү төрт этапка бөлүнөт:

Тапшырманын берилиши

Формалдаштыруу

Алгоритмин түзүү

Программасын тузүү
Алынган натыйжаларды эсептөө жана талдоо жүргүзүү
Бул ырааттуулукту ЭЭМде тапшырманы чечүүдөгү технологиялык чынжырча деп аташат. 1, 2- пункттары негизгилери болуп саналат. Берилген тапшырманы формалдаштырбай туруп программалоо ааламына жаңы киришкен адистер үчүн гана эмес бул тармакты мыкты өздөштүргөн программисттер үчүн да так жана туура жоопту алуу өтө татаал. 3, 4, 5пункттар программалоо этабына түздөн түз байланыштуу. 1-этапта эмне берилди жана эмнени табуу керек экендиги так формулировкалануу тийиш. Бул жерде чечим алуу үчүн талап кылынган баштапкы маалыматтардын толук топтомун аныктоо абдан маанилүү. 2этап тапшырманы формалдаштыруу - бул жерде көбүнчө маселе математикалык формулалардын, теңдемелердин тилине которулат. Эгер чечим кандайдыр бир чыныгы объектинин, кубулуштун же процесстин математикалык чечимин талап кылса, анда формалдаштырууда тиешелүү математикалык моделди алуу керек. 3-этап алгоритимди түзүү. Тажрыйбалуу программисттер алгоритм түзбөй туруп эле программалоого өтүшөт. Бирок, билим берүү максатында алгоритмди түзүп, андан кийин пайда болгон алгоритмди программалоо тилине которуу пайдалуу. Биринчи үч кадам компьютерсиз иштөөнү камтыйт. Андан кийин белгилүү бир тилде, белгилүү бир программалоо тутумунда иш жүзүндө программалоо жүрөт. Акыркы (алтынчы) этап - буга чейин иштелип чыккан программаны практикалык максаттарда колдонуу. Программист сабаттуулугунун негизин алгоритмдик ой жүгүртүү түзөт.
Компьютерлер үчүн программа түзүүнүн методдорун үйрөтүүчү информатиканын бөлүгү «программалоо» деп аталат. Информатика предметин окууда программалоону өздөштүрүү өтө маанилүү. Конкреттүү бир маселени чечүүдө программа өтө чоң ролду ойнойт

Бышыктоочу суроолор
ЭЭМде тапшырманы аткаруу этаптарын атап бер жана аларга аныктоо киргиз.
Формалдаштыруу түшүнүгү жана ага мисалдар

Колдонулган адабияттар
Семакин, И.Г. Основы алгоритмизации и программирования: Учебник / И.Г. Семакин. - М.: Academia, 2017

Тема 2. Алгоритм түзүү жөнүндө жалпы маалымат 2-сабак (лекция)

Алгоритмди жазуу түрлөрү
Негизги суроолор: алгоритмди түзүүнүн тексттик, графикалык (блок схема) жана программалоо тилдеринин түрлөрүн.
Студенттердин мурунку алган билимдери: “Информатика”, “Кесиптик математика”
Окуунун натыйжасы:
Студенттер берилген тапшырманын алгоритмин оозеки же тексттик түрдө түзүп блок схемасын чие алышат. Программалоо тилдеринин түрлөрүн, пайтон программалоо тилинин тарыхын билишет.
Тема боюнча маалымат:

3-сүрөт

Табигый тилде. Сөз түрүндө

Сөз түрүндөгү алгоритм – бул берилген тапшырманы алгоритмин оозеки айтуу же жазма түрүндө жазуу

Мисалы: эртенки аткарыла турган жумушту пландоо, оозеки айтып аткаруучуга дайындоо. Жаңы маалымат технологиясы жана эсептөөчү түзүлүштүн формалдуу сандык методдорун жана программалануучу каражаттарын колдонуунун негизинде тексттик маалыматты талдоо ыкмасы сунушталды. Тексттик маалыматтын авторлугун аныктоо үчүн аны талдоонун инновациялык механизми талап кылынары көрсөтүлгөн. Ыктымалдуулук мүнөздөмөлөрү боюнча тексттин авторлугун аныктоонун ишенимдүүлүк деңгээлин жогорулатуу үчүн программалык камсыздоону кеңейтүүчү мамиле сунушталат. Сунушталган эсептөөчү шаймандын иштөө алгоритми жана принциби сүрөттөлгөн, аны тексттик маалыматты идентификациялоого байланыштуу эсептөөлөрдө, ошондой эле анын авторлугун аныктоодо колдонсо болот. Тексттерди мазмундуу талдоонун негизинде чечим кабыл алуунун сапатын баалоо ыкмасы келтирилген. Сунуш кылынган шаймандын иштешине деталдуу анализ берилген.

Графикалык блок схемасын түзүү

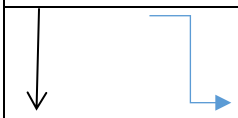
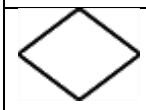
Блок-схема – алгоритмдин аткарылуу процессин, схемалык түрдө көрүнүшү. Блоксхемалар көбүнчө жөнөкөй, логикалык диаграммалардын жардамында документтештирүүдө, изилдөөдө, пландаштырууда, татаал процесстерди түшүндүрүүдө колдонулат. Блок-схемаларды курууда белгилүү бир операцияларды көрсөтүү үчүн тик бурчтуктар, овалдар, ромбдор жана башка геометриялык формалар ошондой эле кадамдардын ырааттуулугун же процесстин багытын көрсөтүүчү туташтыруучу жебелер колдонулат.

Блок-схема белгилеринин жалпы белгилери жана аныктамалары

Блок-схеманын ар кандай түрлөрү бар жана ар биринин функциясы ар түрдүү. Ар бир символдун өзүнүн мааниси жана колдонуунун контексти бар. Эгер блок-схема түзүү учурунда күтүлбөгөн жерден символдорго аралашып кетсе, анда көпчүлүк учурларда,

төмөндө келтирилген жалпы кабыл алынган символдордун минималдуу топтому менен иштелет.

Блок-схемаларда көбүнчө төмөнкү формалар жана белгилер кездешет.

Символ	Аты	Аныктама
	Процесс	Бул белги Аракет деп дагы белгилүү, ишаракеттин же функциянын процессин белгилөө үчүн колдонулат. Бул блоксхемалардагы эң кеңири колдонгон символ.
	Башы жана аягы	Кээде "Терминатор" деп да аталган бул белги схеманын башталыш же аяктоочу чекитин же процессти иштеп чыгуунун белгилүү бир жолунун мүмкүн болгон натыйжасын белгилөө үчүн колдонулат. Блоктун ичинде, эреже катары, "Башы" же "Аягы" деген сөздөр жазылат.
	Багыттоо	Эки блокту туташтырган багыттагычтар
	Чечим	Жоопту талап кылган суроону символдоштурат (көбүнчө ооба / жок же чын / жалган). Бул учурда, блок-схема тандалган жоопко жана кийинки блокторго жараша эки багытта бутактайт.
	Туташтыргыч	Адатта, бир баракты ичиндеги блокторду бириктирүү үчүн татаал схемаларда колдонулат.
	Баракты тышкары туташтыргыч	Көбүнчө ар кандай беттерде жайгашкан өзүнчө блокторду туташтыруу үчүн татаал схемаларда колдонулат. Түшүндүрүүнү жеңилдетүү үчүн, баракты номери, адатта, сүрөттүн ичинде көрсөтүлөт.
	Input Output	Маалыматтар деп да аталган бул форма киргизүү же чыгаруу үчүн жеткиликтүү маалыматтарды жана сарпталган же алынган ресурстарды билдирет. "Кагаз тасма" маалыматтарды киргизүү / чыгаруу дегенди билдирсе дагы , бүгүнкү күндө бул символ эскирген деп эсептелет, ошондуктан блок-схемаларда сейрек колдонулат.

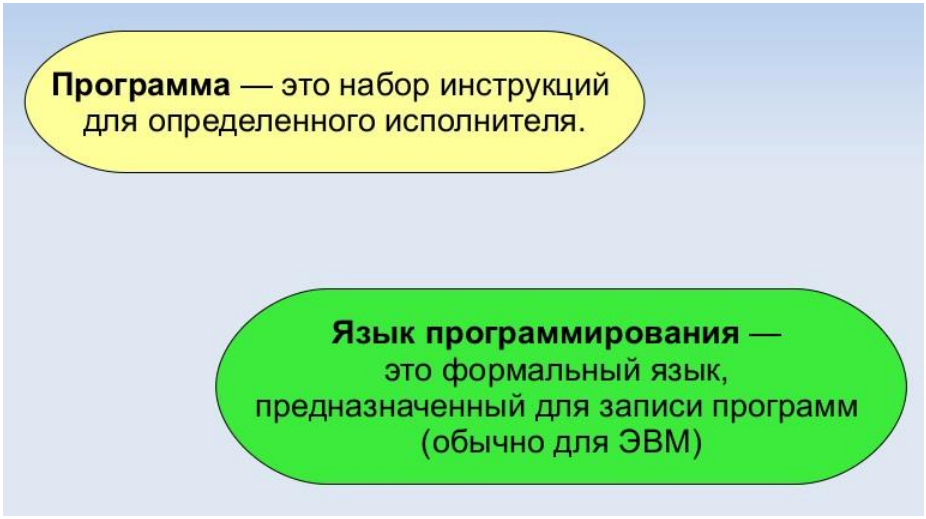
Алгоритмдин программалоо түрү

4-сүрөт

Программалоо тилдеринин түрлөрү аябай көп. Программист каалаган тилде ийгиликтүү программаны түзүү үчүн, берилген тапшырманын алгоритмин туура түзүү жетиштүү.

Программаны объект (аткаруучу) үчүн ырааттуу буйруктардын (алгоритмдин) жыйындысы катары кароого болот, алар белгилүү бир максатка жетүү үчүн аларды аткарышы керек. Демек, сиз "куймакты кантип бышыруу керектиги" жөнүндө көрсөтмө (алгоритмин) түзсөңүз, "программалай" аласыз, ошондо ал аны так аткара баштайт. Бул учурда, адам үчүн көрсөтмө (программа) табигый тил деп аталган, мисалы, орус же англис тилинде жазылат.

Адатта, адамдарды эмес, компьютерлерди атайын тилдерди колдонуу менен программалоо болот. Атайын тилдердин колдонулушуна машиналардын биздин, башкача айтканда, адам тилдерин "түшүнө албагандыгы" себеп болот. Машиналар үчүн нускамалар программалоо тилдеринде жазылган, алар синтаксистик бир мааниликсиздик менен мүнөздөлөт (мисалы, айрым сөздөрдү алмаштырууга болбойт) жана чектелген (алардын сөзсүз түрдө жана символдор топтому бар).



5-сүрөт

Программалоо тилдеринин тарыхый өнүгүшүнүн негизги этаптары

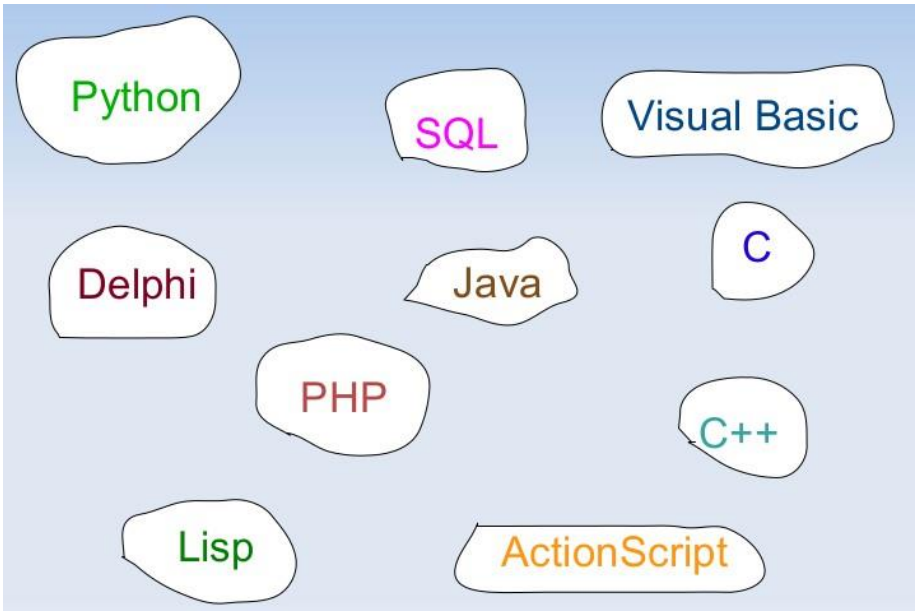
Биринчи программалар машина тилинде жазылган; ал учурда компьютер үчүн азыркыдай иштелип чыккан программалык камсыздоо болгон эмес, ал эми машина тили компьютердик жабдыктар менен иштөөнүн бирден-бир жолу болгон. Машина тилинин ар бир буйругу тигил же бул электрондук шайман тарабынан түздөн-түз аткарылган. Маалыматтар жана буйруктар санарип түрүндө жазылган (мисалы, он алтылык же экилик эсептөө системаларында). Мындай тилдеги программаны түшүнүү өтө кыйын болгон; Мындан тышкары, кичинекей бир программа да көптөгөн саптар менен аяктаган. Ар бир компьютер өзүнүн машиналык тилин гана түшүнгөндүктөн, кырдаал мындан да татаалдашып барган.

Машиналардан айырмаланып адамдарга сандар топтомунан сөздөр көбүрөөк түшүнүктүү. Адамдын сандарга караганда сөздөр менен иштөөгө болгон каалоосу ассемблердин пайда болушуна түрткү берди. Булар буйруктарды жана эс тутумун сандык белгилөөнүн ордуна оозеки жана алфавиттик колдонуучу тилдер.

Машина сөздөрдү түшүнө албай калганы көйгөйдү жаратат. Машина тилине которууга котормочу керек.Ассемблерди колдонулган күндөрүнөн баштап ар бир программалоо тили үчүн котормочулар түзүлүп келген - программанын кодун программалоо тилинен машиналык кодго айландыруучу атайын программалар. Монтаждоочулар. Системалык программалоодо алар иштөө тутумдарынын төмөнкү деңгээлдеги интерфейстерин, драйвер компоненттерин түзүү үчүн колдонулат

Ассемблерлерден кийин жогорку деңгээлдеги тилдердин таңы атат. Бул тилдер үчүн татаал котормочуларды иштеп чыгуу керек болчу, анткени компьютерлерге караганда жогорку деңгээлдеги тилдер адамдар үчүн кыйла ыңгайлуу. Машиналардын түрлөрүнө байланган монтажчылардан айырмаланып, жогорку деңгээлдеги тилдер көчмө. Демек, программаны жогорку деңгээлдеги программалоо тилинде бир жолу жазып, ага ылайыктуу котормочу орнотулган болсо, программист аны каалаган компьютерде аткара алат. Кийинки олуттуу кадам, обьектке багытталган программалоо тилдеринин пайда болушу болду, ал баарынан мурда, иштелип чыккан программалардын татаалдашуусу менен байланыштуу болду. Ушундай тилдердин жардамы менен программист виртуалдык объекттерди башкарат, бул белгилүү бир мааниде программаны чындыкка жакындатат. Бүгүнкү күндө көпчүлүк учурларда ири жана татаал долбоорлорду ишке ашыруу тилдердин объективдүү мүмкүнчүлүктөрүн пайдалануу менен жүзөгө ашырылууда. Башка же ошол эле тилдерде колдоого алынган башка заманбап программалоо парадигмалары бар.

6-сүрөт **Программалоо тилдеринин түрлөрү**



7-сүрөт **Трансляциялоо. Которуу**

Учурда көптөгөн ар кандай жана окшош программалоо тилдери бар. Азыркы учурда компьютердик технологиянын жардамы менен чечилип жаткан маселелердин санын жана ар түрдүүлүгүн элестетсек, мындай көрүнүштүн себеби айдан ачык болуп калат. Ар кандай тапшырмалар ар кандай шаймандарды, башкача айтканда, ар кандай тилдерди жана программалоо парадигмаларын талап кылат.

Көптөгөн программисттер айрым артыкчылыктары бар өзүнүн программалоо тилин ойлоп табууга аракет кылып келишет. Учурда басымдуу көпчүлүгү учурдагы шаймандар турун үйрөнүүгө, программалоо тилдерди сактап калууга жана өнүктүрүүгө көп убакыт сарпташат.

Бардык турдуу тилдерди ар кандай критерийлер боюнча классификациялоого болот. Мисалы, маселелерди чечүү тиби боюнча (системдик же прикладдык багытталган тилдери, вебди иштеп чыгуу тилдери, маалымат базасын уюштуруу, мобилдик тиркеме иштеп чыгуу ж.б.). Бүгүнкү күндө эң популярдуу болуп Java, C ++, PHP, жана Python эсептелет, анын негизин изилдөөгө ушул сабак арналган.

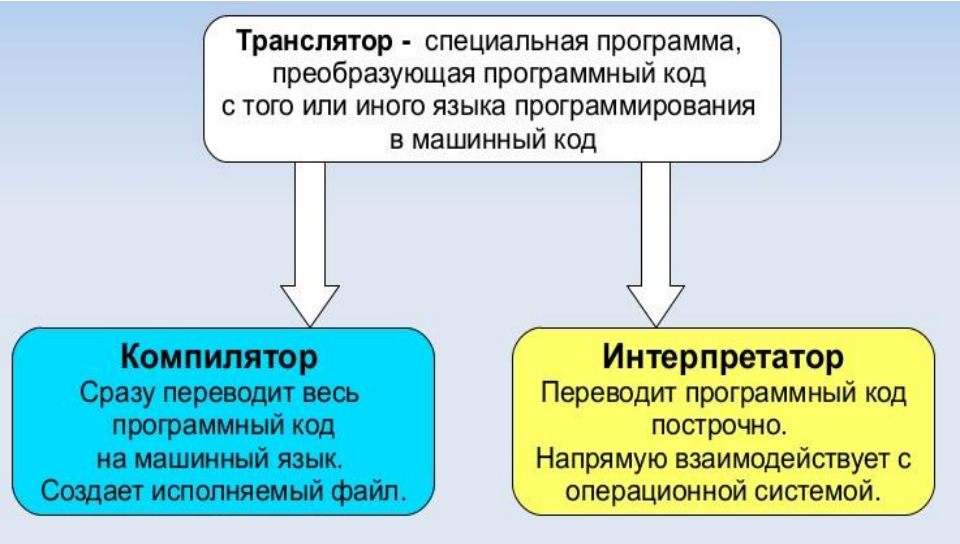
Буга чейин кодду жогорку деңгээлдеги программалоо тилинен машиналык тилге которуу үчүн атайын программа - котормочу керек деп айтылган.

Котормочуга кирген котормо алгоритми татаал. Которуунун эки негизги жолу бар экендигин билүү жетиштүү - программаны түзүү же аны чечмелөө.

Компиляциялоодо бардык баштапкы код (программист жазган) дароо машиналык кодго которулат. Баштапкы код менен эч кандай байланышы жок өзүнчө аткарылуучу файл тузулот. Аткарылуучу файлдын аткарылышы операциондук система (ОС) тарабынан камсыздалат. Аткарылуучу файл алынгандан кийин, котормочу аны окууга муктаж болбой калат.

Түшүндүрүүдө код ырааттуу түрдө аткарылат (биринин артынан бири деп айтсак болот). Болжол менен айтканда, операциондук система программанын кодун камтыган файл менен эмес, котормочу менен иштешет. Тилмеч башка булак кодун окуп, аны машиналык кодго которот (же компьютердик код эмес, бирок ОС үчүн "түшүнүктүү") жана ОСге берет. ОС ушул кодду аткарат жана котормочудан кийинки "берууну" күтөт. Python - дал ушундай тил. Бул чечмеленген программалоо тили.

Түзүлгөн программанын аткарылышы тезирээк, анткени бул даяр машина коду. Бирок, азыркы компьютерлерде чечмелөө учурунда аткаруу ылдамдыгынын төмөндөшү байкалбайт. Мындан тышкары, чечмеленген тилдердин бир катар артыкчылыктары бар, анын ичинде программаны ишке ашыруу боюнча даярдык иш-аракеттеринин жоктугу, бул жаңы башталгычтар үчүн биринчи жолу программа түзүшү мүмкүн.



8-сүрөт

Транслятор – программалык кодду тигил же бул программалоо тилинен машиналык кодко которгон атайын программа

Бышыктоочу суроолор

Транслятор, компилятор, интерпретатор деген эмне

Программалоо тилдеринин түрлөрүн ата

Программа жана программалоо тили дегенге аныктама бер

Программалоо тилинин тарыхый өнүгүшү Алгоритмди жазуунун канча түрү бар?

Блок схема деген эмне жана аны түзүүдө эмнелер колдонулат?

Программалоо тилдерин атап бер

Үйгө тапшырма

Алгоритмдин түрлөрүнө мисалдар келтирүү

Программалоо тилдерине хронологиялык таблица түзүү

Колдонулган адабияттар

Гуриков, С.Р. Основы алгоритмизации и программирования на Python: Учебное пособие / С.Р. Гуриков. - М.: Форум, 2018

Семакин, И.Г. Основы алгоритмизации и программирования: Учебник / И.Г. Семакин. - М.: Academia, 2017

Бөлүм 2 Тема 3. Python программалоо тили. Python программалоо тили менен таанышуу

3-сабак (лабораториялык)

Максаты: Python программасы менен таанышуу. Python программалоо тилинин кыскача тарыхын жана өзгөчөлүктөрүн билүү.

Негизги суроолор: Кыскача тарыхый маалымат. Тилдин негизги өзгөчөлүктөрү. Python программасында программа түзүү. Zen Python. Интерактивдүү режим. Скриптерди жаратуу

Студенттердин мурунку алган билимдери: “Информатика”, “Кесиптик математика”

Окуунун натыйжасы: Python программалоо тилинин тарыхын, башка программалардан айырмачылыгын, скриптер түшүнүгүн билишет.

Тема боюнча маалымат:

Кыскача тарыхый маалымат

Python программалоо тили 1991-жылы голландиялык Гидо ван Россум тарабынан түзүлгөн.

Анын аты - Python (же Python) - сойлоп жүрүүчү эмес, сериалдын аталышынан келип чыккан.

Россум тилди иштеп чыккандан кийин, аны Интернетте жайгаштырган, программистер уюму аны жакшыртып иштеп чыгууга жана өркүндөтүүгө киришкен.

Учурда Python активдүү өркүндөп жатат. Жаңы версиялары тез-тез чыгып турат. Колдоого алынган эки бутак бар: Python 2.x жана Python 3.x. Бул жерде англис тилиндеги "x" тамгасы белгилүү бир релизди билдирет. Экинчи жана үчүнчү Pythondун ортосунда бир аз айырма бар. Бул сабак Python 3.x негизделген.

Расмий тилди колдоо сайты - https://www.python.org.

Тилдин негизги өзгөчөлүктөрү

Python - бул чечмеленген программалоо тили. Бул баштапкы код атайын программа - котормочу тарабынан окуу процессинде машиналык кодго бөлүктөрү менен бөлүнүп которулат дегенди билдирет.

Python ачык синтаксис менен мүнөздөлөт. Андагы кодду окуу башка программалоо тилдерине караганда жеңилирээк, анткени Pythondo кашаа, үтүрлүү чекит сыяктуу көмөкчү синтаксистик элементтер аз колдонулат. Экинчи жагынан, тилдин эрежелери программисттерди уюлдук конструкцияларын белгилоодо отступ жасоого мажбурлайт, уюлдук структураларды көрсөтүүгө мажбурлайт. Ооба, жакшы иштелип чыккан конулду алаксыткан элементтердин аздыгы, окууга жана түшүнүүгө оңой.

Python - бул ар тараптуу колдонулган, толук кандуу универсалдуу программалоо тили. Негизги, бирок жалгыз колдогон парадигма эмес, объектке багытталган программалоо.

Бирок, бул сабакта биз объектилер жөнүндө гана сөз кылабыз жана структураланган программалоону изилдейбиз, анткени ал база болуп саналат. Берилген маалыматтардын негизги түрлөрү, бутактары, циклдары, функциялары жөнүндө билбестен, татаал парадигмаларды изилдөөнүн эч кандай мааниси жок, анткени аларда ушулардын бардыгын колдонушат.

Python котормочулары GNU General Public License сыяктуу лицензия боюнча эркин таратылат.

Python программасында программа түзүү. Zen Python

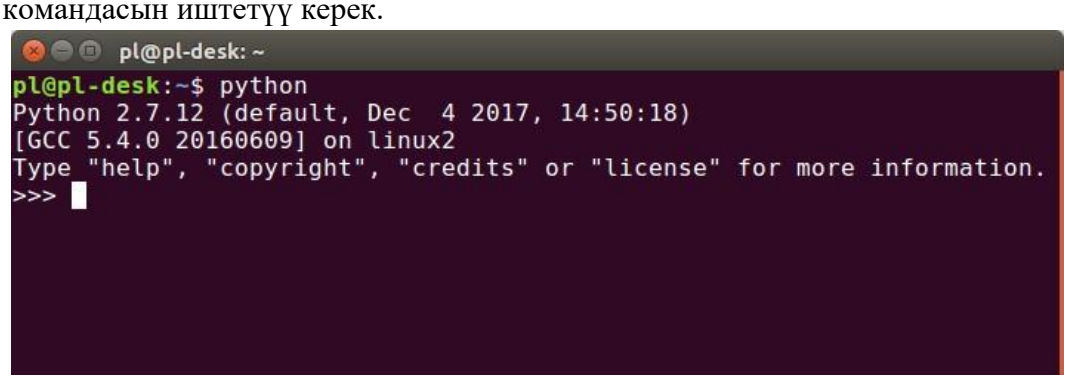
Эгерде Python котормочусуна импорттоо буйругу берилсе import this (бул жакка "муну импортто", "өзүңөрдү импорттогула" деп түшүнүш керек), анда "Python Zen" деп аталган нерсе ушул тилдин идеологиясын жана өзгөчөлүктөрүн чагылдырган көрсөтүлөт. Бул постулаттардын маанисин программалоого тиркемеде түшүнүү тилди толук өздөштүрүп, Лабораториялык программалоодо тажрыйба топтогондо пайда болот.

Интерактивдүү режим

Атап айтканда, интерпретатор буйруктарды сап менен аткарат. Бир сап жазып, Enter баскычы басылат, интерпретатор аны аткарат, натыйжасын чыгарат.

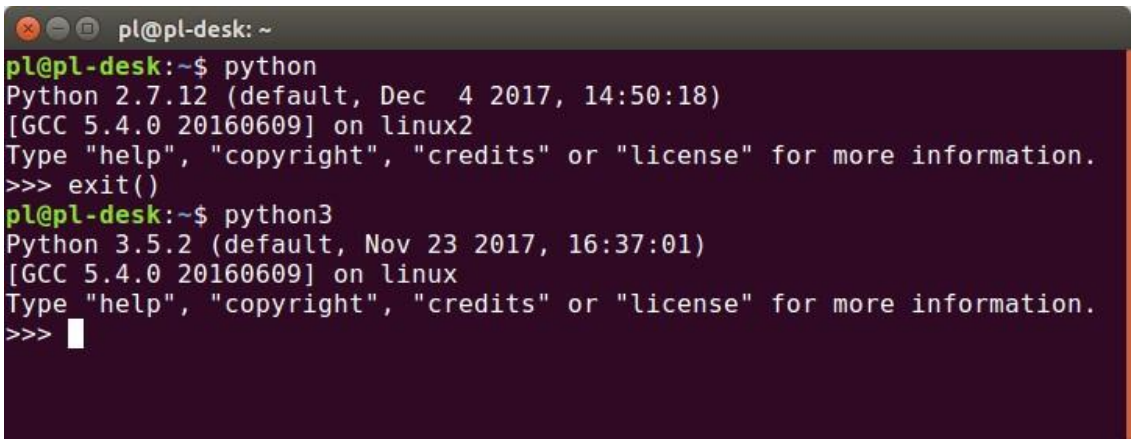
Бул тилдин өзгөчөлүктөрүн үйрөнүп жатканда же коддун айрым кичинекей бөлүктөрүн текшерип жатканда ыңгайлуу. Эгерде компиляцияланган тилде иштесе, анда биринчи файлдын кодун баштапкы программалоо тилинде түзүп, андан кийин аны компиляторго өткөрүп, андан аткарылуучу файлды алып, андан кийин гана программаны аткарып, натыйжасын баалоо керек болот. Бактыга жараша, компиляцияланган тилдерде дагы, бул иш-аракеттердин бардыгы программалоочунун жашоосун жеңилдеткен өнүгүү чөйрөсү тарабынан жүзөгө ашырылат.

Linux ядросунун базасында, операциондук системаларда интерактивдүү режимде, командалык кабык Bash иштеген “Терминал” тиркемесинин жардамында Python программасында программалоо мүмкүн. Бул жерде, интерпретаторду баштоо үчүн, python командасын иштетүү керек.



9-сүрөт

Pythondун экинчи бутагынын интерпреттору башталат, аны биринчи маалымат тилкесинен көрүүгө болот. Азыркы учурда, 2.7.12 версиясы ишке киргизилген. Биринчи "2" саны анын Python 2 программалоо тили үчүн интерпретатор экендигин билдирет.Үч кырлуу кашаа (>>>) бар акыркы катар бул командаларды киргизүүгө чакыруу. Бул курста Python 3 колдонула тургандыктан, exit () командасын колдонуп, котормочудан чыгабыз (exit-чыгуу). Андан кийин python3 командасын терминалда аткарабыз.

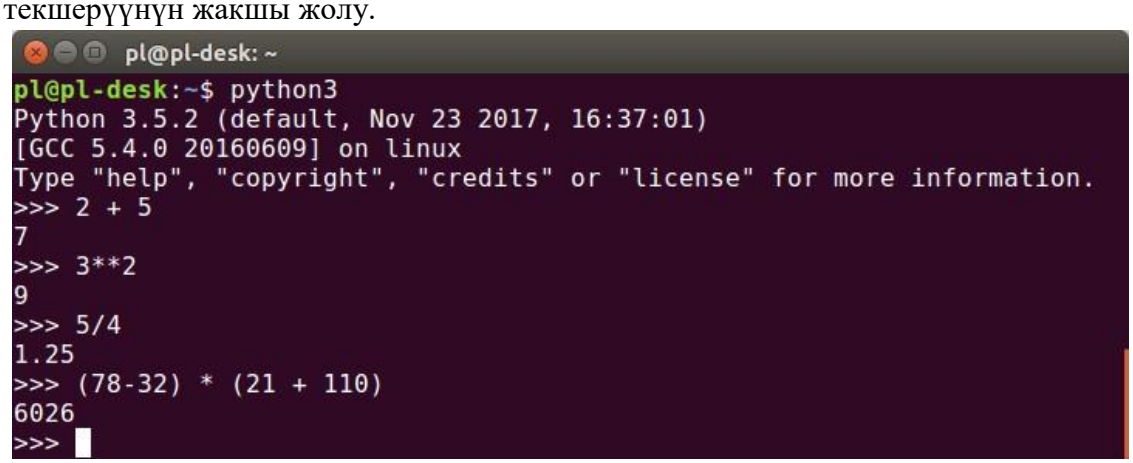


10-сүрөт

Эгерде python3 пакети орнотулбаган болсо. Аны өзүңүз орнотууга керек болот.

Windows иштөө тутумдары үчүн интерпретаторду расмий сайттан жүктөп алуу керек (https://www.python.org/downloads/windows/). Орнотуудан кийин, ал ярлык аркылуу ишке киргизилет. Бул жерде буйрук кабыгын колдонуунун кажети жок.

Pythondун мүмкүнчүлүктөрү аны калькулятор катары колдонууга мүмкүндүк берет. Биз тил буйруктарын изилдеп көрбөгөндүктөн, бул интерактивдүү буйрук киргизүүнү текшерүүнүн жакшы жолу.



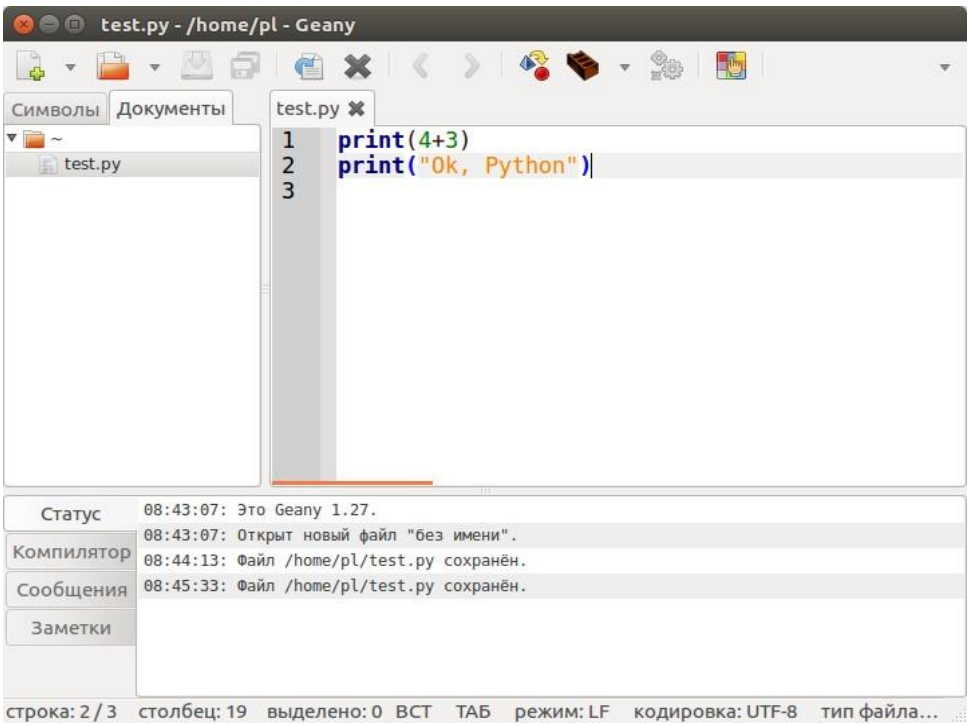
11-сүрөт

Киргизүү процессинде ката кетирилген же мурун колдонулган буйрукту кайталоо керек болгон учурлар болот. Сапка кайра кайра кийирбөө үчүн, консолдо буйруктар таржымалын, тергичтин өйдө жана ылдый жебелери аркылуу жылдырып колдонсо болот. IDLE (Windows) чөйрөсүндө клавиатуранын биргеликтери колдонулат (Alt + N жана Alt + P).

Скриптерди жаратуу

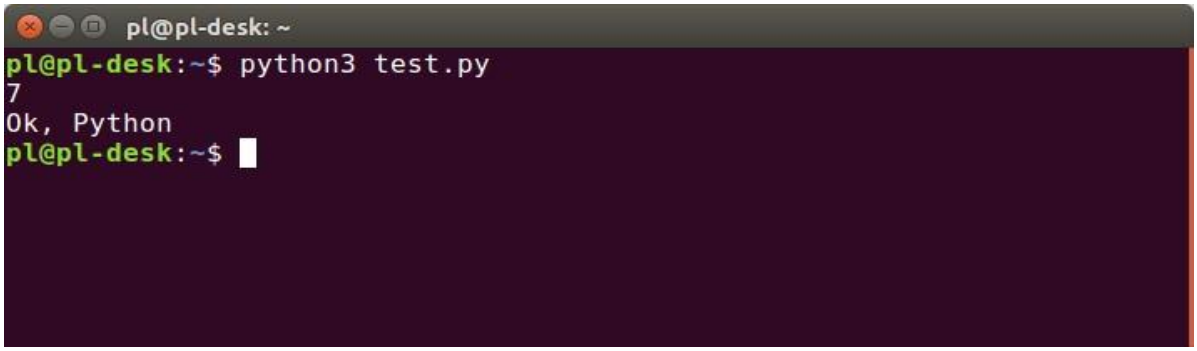
Интерактивдүү режимдин ыңгайлуулугуна карабастан, көбүнчө программанын баштапкы кодун кийинчерээк аткаруу жана колдонуу үчүн сактоо керек. Бул учурда, интерпретаторго аткарууга өткөрүлүп берилүү үчүн файлдар даярдалат. Python го файлдар адатта ру кеңейтүүсүнө ээ.

PyCharm сыяктуу бир катар Python иштеп чыгуу үчүн чөйрөлөр бар. Бирок, баштапкы учурда Geany, синтаксисти бөлүп көрсөтүү менен тексттик редактор ылайыктуу



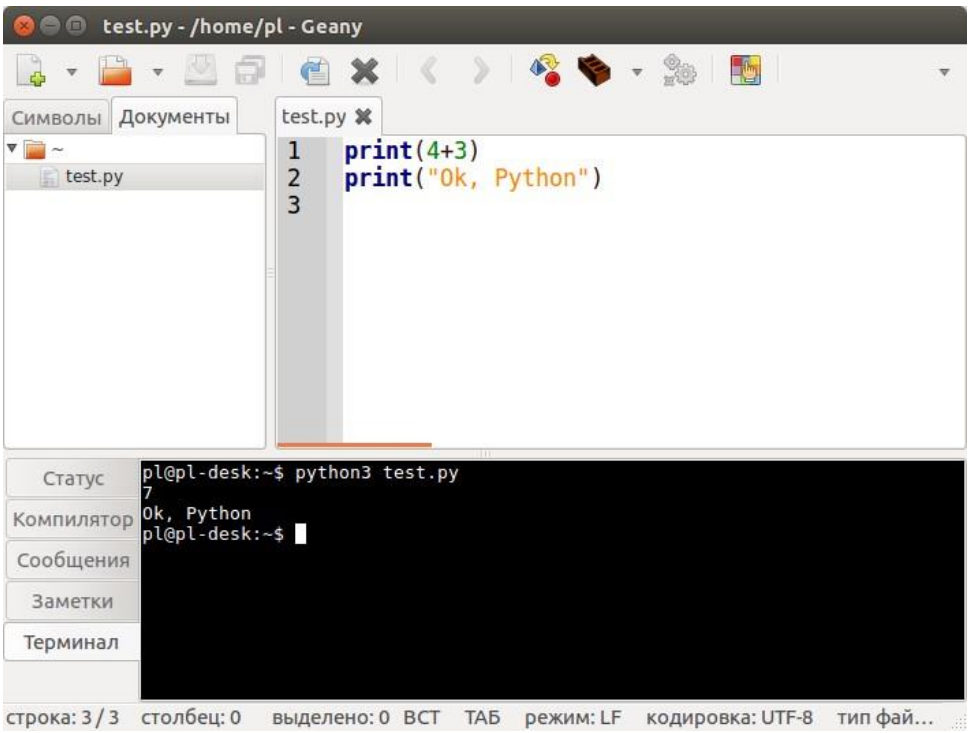
12-сүрөт

Бул жерде файл кодду менен түзүлүп, сакталат. Андан кийин аны терминал аркылуу иштетүүгө болот. Бул учурда алгач интерпретатор көрсөтүлөт (биздин учурда python3), андан кийин файлдын аталышы (эгер файл башка каталогдо болсо, анда ал дареги менен көрсөтүлөт же Bash кабыгындагы cd командасынын жардамында, ушул каталогго өтүү керек).



13-сүрөт

Бирок, Geany ге кошумча терминал орнотууга болот, (sudo apt-get install libvte9), бул ишти жөнөкөйлөтөт.



14-сүрөт

Акырында, редактордо F5 басылат, ал файлды аткарууга жөнөтөт (терминал өзү ачылат, программаны аткаргандан кийин жана Enter баскычын басканда жабылат). Windows'то файлдарды даярдоо ошол эле IDLE чөйрөсүндө да мүмкүн. Ал үчүн менюдан Файл → Жаңы терезе (Crtl + N) командасын тандаңыз, таза (чакыруусуз >>>) жаңы терезе ачылат., Синтаксистин белгилеп көрсөтүүсү пайда болушу үчүн, мүмкүнчүлүктүн болушунча файлды ру кеңейтүүсү менен сактоо керек. Код даяр болгондон кийин, файлды дагы бир жолу сактаңыз. Скрипти ишке киргизүү Run → Run Module (F5) командасынын жардамы менен ишке ашырылат. Андан кийин, коддун аткарылышынын натыйжасы интерактивдүү режим терезесинде пайда болот.

Үй тапшырма

Python интерпретаторунда интерактивдүү режимде бир нече жөнөкөй математикалык амалдарды аткаруу. Жыйынтыгын түшүндүрүп берүү.

Колдонулган адабият

Гуриков, С.Р. Основы алгоритмизации и программирования на Python: Учебное пособие / С.Р. Гуриков. - М.: Форум, 2018

Тема 4. Маалыматтар жана алардын түрлөрү 4-сабак (лекция)

Негизги суроолор: Берилиштердин түрлөрү. Программалоодогу операциялар. Маалымат түрлөрүн өзгөртүү. Өзгөрүлмөлөр

Студенттердин мурунку алган билимдери: “Информатика”, “Кесиптик математика”

Окуунун натыйжасы: Python программасындагы берилиштердин түрлөрү, операнда жана операция түшүнүгүн билишет. Маалымат жана өзгөрүлмөлөр менен иштей алат.

Тема боюнча маалымат:

Жашоодо бизди курчап турган нерселерге, же буюмдарга ар кандай амалдарды жасайбыз. Биз алардын касиеттерин өзгөртөбүз, аларды жаңы функциялар менен камсыз кылабыз. Түшүнүк боюнча бул менен, компьютердик программалар объекттерди бир гана виртуалдык, санариптик башкарат. Биз объект-багытталган программалоо деңгээлине жеткенге чейин, мындай объектилерди маалыматтар деп атайбыз.

Албетте, маалыматтар түрдүү болот. Көбүнчө, компьютердик программа сандар жана саптар менен иштөөгө туура келет. Ошентип өткөн сабакта арифметикалык амалдарды аткарып сандар менен иштедик. Кошуу операциясы биринчи сандын экинчи сандын чоңдугуна өзгөрүүсүн аткарган, ал эми көбөйтүү бир санды экинчисине туура келген көлөм жолу көбөйттү.

Өз кезегинде, сандар дагы ар башка болот: бүтүн, чыныгы, чоң мааниге же өтө узун бөлчөккө ээ болушу мүмкүн.

Python программалоо тили менен таанышканда, биз маалыматтардын үч түрүн кездештиребиз:

бүтүн сандар (int тиби) - оң жана терс бүтүн сандар, ошондой эле 0 (мисалы, 4, 687, -45, 0).

• калкып чыккан чекит сандары (float тиби) - бөлчөк, ошол эле учурда чыныгы, сандар (мисалы, 1.45, -3.789654, 0,00453). Эскертүү: Бүтүн жана бөлчөк бөлүктөрдү бөлүү үчүн үтүр эмес, чекит колдонулат.

• саптар (str тиби) - тырмакчага алынган символдордун жыйындысы (мисалы, "ball", "What is your name?", 'dkfjUUv', '6589').

Эскертүү: Pythondогу тырмакчалар бирдик же экилик болушу мүмкүн; тырмакчадагы жалгыз символ да катар болуп эсептелинет, өзүнчө бөлөк символ тип Pythondо жок.

Программалоодогу операциялар

Операция –маалыматтар үсүндө аткарылган кандайдыр бир амалдар. Аракеттин өзүн оператор аткарат - атайын курал. Эгерде сиз стол жасоо операциясын аткарып жаткан болсоңуз, анда сиздин операндлар тактай жана мык, ал эми операторуңуз балка болмок.

Математикада жана программалоодо плюс белгиси сандарга карата кошуу операциясынын оператору болуп саналат. Саптар болсо, ошол эле оператор конкатенация операциясын аткарат, башкача айтканда, кошулат. >>> 10.25 + 98.36 108.61

>>> 'Hello' + 'World'

'HelloWorld'

Бул жерде белгилей кетүүчү нерсе, операторду операцияда жасай турган нерсеси ага гана эмес, ал иштеген маалыматтардын түрлөрүнө да байланыштуу. Крокодил кол салган учурда, балка курулуш куралынын ролун аткарбай калат. Бирок, көпчүлүк учурларда операторлор универсалдуу болбой калат. Мисалы, операндалар бир жагынан сан болуп, экинчи жагынан сап болсо, кошуу белгиси колдонулбайт.

>>> 1 + 'a'

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: unsupported operand type(s) for +: 'int' and 'str'

Бул жерде сапта TypeError: unsupported operand type(s) for +: 'int' and 'str' интерпретатор ката кеткендиги жөнүндө кабар берет, мисалы int и str типтерин колдобогон операнд ушул сыяктуу.

Маалымат түрлөрүн өзгөртүү

Эгерде 1 санын "1" сапка айландырса жогорудагы операцияны аткарса болот. Айрым маалыматтардын түрлөрүн башкаларга өзгөртүү үчүн, Python бир катар камтылган функцияларды камсыз кылат (функция деген эмне, негизинен, сиз башка сабактардан үйрөнөсүз). Азырынча үч түр менен гана иштегендиктен (int, float жана str), тиешелүү функцияларды гана карайбыз: int
(), float (), str ().

```
>>> str(1) + 'a'
'1a'
>>> int('3') + 4
7
>>> float('3.2') + int('2')
5.2
>>> str(4) + str(1.2)
'41.2'
```

Бул функциялар кашаанын ичине кандай гана нерсе салынбасын, аны толугу менен, чыныгы санга же сапка айландырат. Бирок, баардыгын өзгөртүү мүмкүн эместигин түшүнүү керек:

```
>>> int('hi')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'hi'
```

Бул жерде маанинин катасы (ValueError) чыгарылган, анткени базалык 10 санына которулбай турган сөзмө-сөз (бул учурда алфавиттик белгилер менен сап) өткөрүлүп берилген, бирок int () функциясы анчалык деле жөнөкөй эмес: >>> int('101', 2)
5
>>> int('F', 16)
15

Эгерде сиз ар кандай эсептөө тутумдары жөнүндө билсеңиз, анда бул жерде эмне болгонун түшүнөсүз.

Дагы бир нерсеге көңүл буруңуз. Берилген маалыматтарды баалуулуктар жана литералдар деп атоого болот. Ушул үч түшүнүк ("маалымат", "баалуулук", "сөзмө-сөз") бир эле нерсени билдирбейт, бирок жакын жана көп учурда синоним катары колдонулат. Алардын ортосундагы айырмачылыкты, колдонулган жерлерин түшүнүү үчүн программалоону тереңирээк изилдөө керек.

Өзгөрүлмөлөр

Берилген маалыматтар компьютердин эс тутумдарында сакталат. Санды киргизгенибизде, ал кандайдыр бир эс тутумуна жайгаштырылат. Бирок кайда жайгашкандыгын так кайдан билүүгө болот? Кийин бул маалыматтарга кандайча жетүүгө болот? Кандайдыр бир жол менен эстеп калуу керек, ылайыктуу уячаны белгилеп коюу керек.

Буга чейин, программаларды машинанын тилинде жазууда, эс тутум уячаларына кирүү алардын регистрлерин көрсөтүү менен жүргүзүлүп келген, башкача айтканда, алар маалыматтарды кайда коюп, кайдан алса болорун айтышкан. Бирок, ассемблерлердин пайда

болушу менен, алар адамдарга бир топ ыңгайлуу болгон маалыматтарга жетүүдө оозеки өзгөрмөлөрдү колдоно башташты.

Өзгөрмөлөр менен берилиштердин ортосундагы байланыш механизми программалоо тилине жана маалыматтардын түрлөрүнө жараша айырмаланышы мүмкүн. Азырынча, программа маалыматтарды ысым менен байланыштыраарын жана келечекте аларга ушул өзгөрмө ат менен жеткиликтүүлүгүн унутпоо жетиштүү.

"Өзгөрмө" сөзү бир нерсени өзгөртө алат дегенди билдирет, ал туруктуу эмес. Чындыгында, сиз муну кийинчерээк көрө аласыз, ошол эле өзгөрмө алгач айрым маалыматтар менен, андан кийин башкалар менен байланыштырылышы мүмкүн. Башкача айтканда, анын мааниси өзгөрүшү мүмкүн, ал өзгөрүүчөн.

Python программасында, көпчүлүк башка тилдердегидей эле, = белгисин колдонуп, берилмелер менен өзгөрмөлөрдүн ортосундагы байланыш түзүлөт. Бул операция ыйгаруу деп аталат. Мисалы, sq = 4 сөз айкашы эс тутумдун белгилүү бир аймагында жайгашкан 4 санын чагылдырган объектке sq өзгөрмөсү шилтеме берип жаткандыгын билдирет жана бул объектти sq аты менен атоо керек.

Ар кандай аталыштар болушу мүмкүн. Бирок, аларды жазуунун жалпы эрежелери бар:

1. Өзгөрмөлөргө алар шилтеме кылган маалыматтардын максатын көрсөткөн, алардын маанисин берген ысымдарды берген оң.
2. Өзгөрмөнүн аталышы тил буйруктары менен дал келбеши керек (резервдик ачкыч сөздөр).
3. Өзгөрмөнүн аталышы тамга же асты сызык менен башталууга тийиш, бирок сан менен эмес.
4. Өзгөрмө аталышында боштук болбошу керек.

Котормочунун режиминде турганда, өзгөрмө тарабынан айтылган маанини билүү үчүн, аны жөн эле чакырыңыз, башкача айтканда атын жазып, Enter баскычын басыңыз. >>> sq = 4
>>> sq 4

Бул жерде өзгөрмө менен иштөөнүн татаал мисалы келтирилген:

```
>>> apples = 100
>>> eat_day = 5
>>> day = 7
>>> apples = apples - eat_day * day
>>> apples 65
```

Үч өзгөрмө бар: apples, eat_day и day. Алардын ар бирине өзүнчө маани берилген. apples = apples - eat_day * day татаал жазуу. Адегенде барабар белгинин оң жагындагы жазуу аткарылат. Андан кийин, анын натыйжасы өзгөрүлмө apples ке ыйгарылат, натыйжада анын эски мааниси (100) жоголот. apples - eat_day * day өзгөрүлмө аталыштардын ордуна, башкача айтканда, 100, 5 жана 7 сандарынын маанисин колдонот.

Бышыктоочу суроолор

1. Операция менен операнданы программада көрсөтүп бер.
2. Өзгөрүлмөлөргө мисалдар келтир.
3. маалымат түрлөрүн өзгөртүү жолдорун айтып бер

Колдонулган адабият

Гуриков, С.Р. Основы алгоритмизации и программирования на Python: Учебное пособие / С.Р. Гуриков. - М.: Форум, 2018

Тема 5. Маалыматтарды киргизүүчү жана чыгаруучу функциялар 5-сабак (лабораториялык)

Максаты: маалыматтарды чыгаруучу print() жана кийирүүчү input() функцияларын программада иштетүү ыкмасын билүү

Негизги суроолор: Программа түзүүдө маалыматтардын ролу. Маалыматтарды чыгаруучу print() функциясы. Маалыматтарды кийирүүчү input() функциясы.

Студенттердин мурунку алган билимдери: “Информатика”, “Кесиптик математика”

Окуунун натыйжасы: программа түшүнүгүн билет. Python программасында чыгаруучу print() жана кийирүүчү input() операторлору менен иштей алат.

Тема боюнча маалымат:

Print () функциясы менен таанышканбыз. Ал, дайындарды экранда көрсөтүү үчүн жооп берет. Эгер код файлда камтылса, ансыз жасай албайсыз. Интерактивдүү режимде, айрым учурларда, ансыз деле жасай аласыз.

Программага маалыматтарды киргизүү жана чыгаруу программалоодо маанилүү. Киргизүүсүз, программалар өзүлөрү кокустук маанилерди жараткан учурларды кошпогондо, ошол эле нерсени жасашмак. Чыгуу программанын натыйжаларын көрүүгө, пайдаланууга, андан ары өткөрүүгө мүмкүнчүлүк берет.

Адатта, программа тышкы булактардан келген ар кандай маалыматтарды иштеп чыгуу үчүн талап кылынат. Экинчиси файлдар, клавиатура, тармак, башка программанын чыгышы болушу мүмкүн. Маалыматтарды чыгаруу файлдарга ж.б.у.с. мүмкүн, бирок, көпчүлүк учурда монитордун экранында болот.

Программа - бул сырткы чөйрө менен бир нерсе алмаштыруучу ачык система деп айта алабыз. Эгерде тирүү организм негизинен зат жана энергия менен алмашса, анда программа – маалыматтар менен алмашат.

Маалыматтарды чыгаруу. print() функциясы

Программалоодо кандай функция бар экендигин кийинчерээк билебиз. Азырынча print () - кашаанын ичиндеги нерсени экранга чыгарып турган Python буйругу деп ойлойбуз. >>> print(1032)

1032

>>> print(2.34)

2.34

>>> print("Hello") Hello

Бардык маалыматтардын түрлөрү кашаанын ичинде болушу мүмкүн. Мындан тышкары, маалыматтардын көлөмү ар кандай болушу мүмкүн.

>>> print("a:", 1)

a: 1

```
>>> one = 1
>>> two = 2
>>> three = 3
>>> print(one, two, three)
1 2 3
```
Басып чыгаруу () функцияларына, ошондой эле алардын маниси басылып чыга турган өзгөрмөлөргө түздөн-түз литералдарды (бул учурда "a:" жана 1) өткөрсө болот.
Функциянын аргументтери (кашаанын ичинде эмне бар) үтүр менен ажыратылат. Чыгууда маанилер үтүрдүн ордуна боштук менен бөлүнөт.
Эгерде кашаанын ичинде бир сөз айкашы бар болсо, анда алгач ал аткарылат, андан кийин print () ушул сөз айкашынын натыйжасын көрсөтөт:
```
>>> print("hello" + " " + "world") hello
world
>>> print(10 - 2.5/2)
8.75
```
Print () кошумча параметрлерди камсыз кылат. Мисалы, sep параметрин колдонуп, боштуктан башка сызык бөлгүчтү көрсөтсөңүз болот:
```
>>> print("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun", sep="-")
Mon-Tue-Wed-Thu-Fri-Sat-Sun
>>> print(1, 2, 3, sep="//")
1//2//3
```
:
Аяктоочу параметр сап басылып чыккандан кийин эмне кылуу керектигин көрсөтүүгө мүмкүндүк берет. Демейки боюнча, сап үзүлүшү пайда болот. Бирок, бул иш-аракетти башка символдорду же саптарды көрсөтүү менен өзгөртүүгө болот:
```
>>> print(10, end="")
10>>>
```
Адатта, эгерде аягы колдонулса, анда интерактивдүү режимде эмес, сценарийлерде, катардагы бир нече чыгышты жаңы сап менен эмес, айталы, үтүр менен бөлүү керек. Жаңы сызык өзү "\ n" белгилеринин айкалышы менен көрсөтүлөт. Эгерде сиз бул маанини акыркы параметрге ыйгарсаңыз, анда print () функциясынын иштешинде эч кандай өзгөрүүлөрдү көрө албайсыз, анткени бул маани демейки боюнча дайындалган:
```
>>> print(10, end='\n')
10
>>>
```
Бирок, эгер чыккандан кийин дагы бир кошумча сызыкты чегиндириш керек болсо, анда мындай кылсаңыз болот:
```
>>> print(10, end='\n\n')
10
 >>>
```
Print () функциясы жөнүндө сөз кыла турган кийинки нерсе - бул сапты форматтоону колдонуу. Чындыгында, бул print () менен эч кандай байланышы жок, бирок саптарга тиешелүү. Бирок ал көбүнчө print () функциясы менен бирге колдонулат.
Форматтоо эски деп аталган стилде же форматтын сап ыкмасын колдонуу менен жүргүзүлөт. Эски стилди Си экранда көрсөтүү ыкмасына окшош болгондуктан C-стили деп да аташат, бир мисалды карап көрөлү:

```
>>> pupil = "Ben"
>>> old = 16
>>> grade = 9.2
>>> print("It's %s, %d. Level: %f" % (pupil, old, grade))
It's Ben, 16. Level: 9.200000
```

Бул жерде% s,% d,% f белгилеринин үч айкалышынын ордуна, pupil, old, grade өзгөрүлмө маанiси алмаштырылды. S, d, f тамгалары маалыматтардын түрлөрүн билдирет - сап, бүтүн сан, чыныгы сан. Эгер үч сап керек болсо,% s бардык учурларда колдонулат. grade өзгөрүлмөсүнүн маанiси ордунда 9.2 көрсөтүлгөнү менен, экранга ал кошумча нөлдөр менен басылып чыккан. Ошентсе да үтүрдөн кийин канча белги керектигин көрсөтө алабыз: f тамгасынын алдынан каалаган числону жазып

```
>>> print("It's %s, %d. Level: %.1f" % (pupil, old, grade)) It's
Ben, 16. Level: 9.2
```

Эми format(): медодуна крап көрөбүз

```
>>> print("This is a {0}. It's {1}.".format("ball", "red")) This
is a ball. It's red.
>>> print("This is a {0}. It's {1}.".format("cat", "white")) This
is a cat. It's white.
>>> print("This is a {0}. It's {1} {2}.".format(1, "a", "number")) This
is a 1. It's a number.
```

Тармал кашаадагы сызык бул жерде алмаштырыла турган маалыматтардын сандарын көрсөтөт. Андан кийин, сапка format () ыкмасы колдонулат. Маалыматтардын өзү анын кашаанын ичинде көрсөтүлгөн (сиз өзгөрмө колдонсоңуз болот). Format () ыкмасынын биринчи аргументи нөлдүк орунга, экинчиси 1 саны бар орунга ж.б.у.с. Чындыгында format () ыкмасынын мүмкүнчүлүктөрү бир кыйла кеңирээк жана аларды үйрөнүү үчүн өзүнчө сабак керек болот. Бул бизге азырынча жетиштүү болот.

Маалыматтарды кийирүү. input() функциясы

Input () функциясы программага Pythondогу клавиатурадан маалыматтарды киргизүүгө жооп берет. Бул функция чакырылганда, программа аткарууну токтотуп, колдонуучунун текст киргизүүсүн күтөт. Андан кийин, ал Enter басканда, input () функциясы киргизилген текстти алып, алгоритмге ылайык иштеп чыккан программага өткөрүп берет.

Эгер сиз input() командасын интерактивдүү режимде киргизсеңиз, анда сиз кызыктуу эч нерсе көрө албайсыз. Компьютер сизден бир нерсе терип, Enter басканды же жөн гана Enter басканды күтөт. Эгер бир нерсе киргизсеңиз, ал дароо экранда пайда болот:

```
>>> input() Yes!
'Yes!'
```

Input () функциясы киргизүүнү программага өткөрүп берет. Алар өзгөрүлмөгө берилиши мүмкүн. Бул учурда интерпретатор дароо катарды чыгарбайт: >>> answer = input() No, it is not.

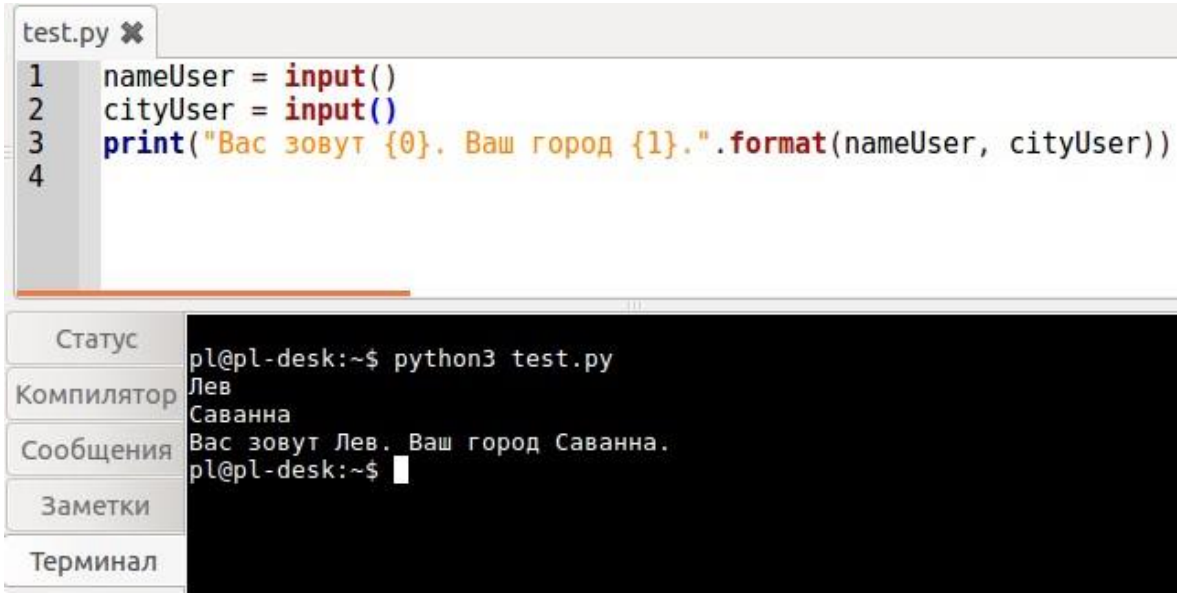
Бул учурда, answer катар өзгөрүлмөсүндө сакталат, эгер кааласак, анын маанисин экранда көрсөтө алабыз: >>> answer

```
'No, it is not.'
```

Print () функциясын колдонгондо, натыйжада тырмакчалар алынып салынат:

```
>>> print(answer) No,
it is not.
```

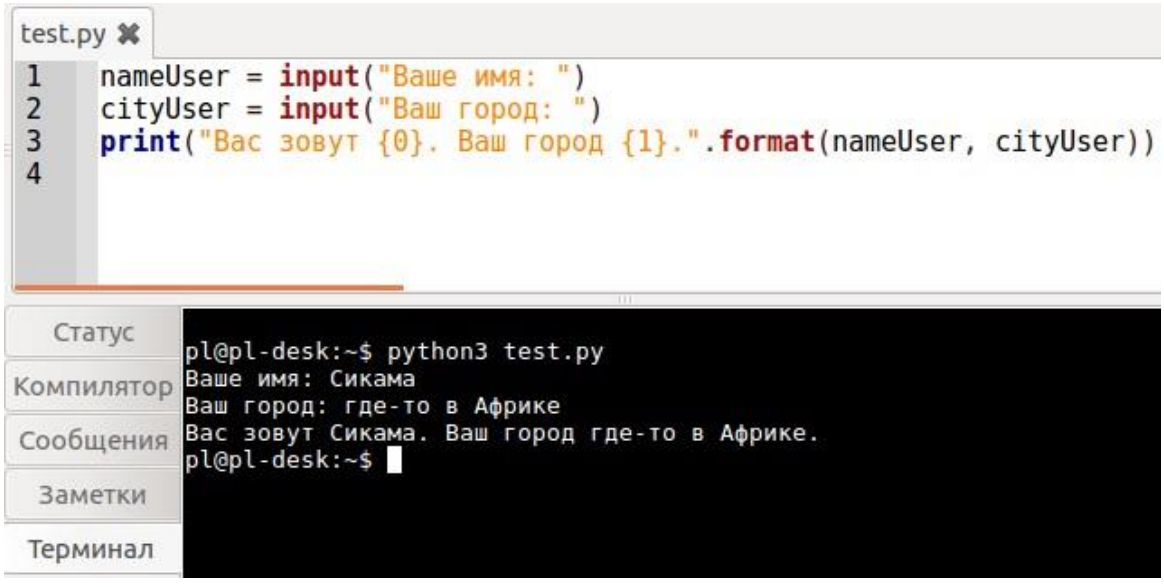
Input () функциясын скрипттерде - коду бар файлдарда колдонуу алда канча кызыктуу. Ушул сыяктуу программаны карап көрөлү:



15-сүрөт

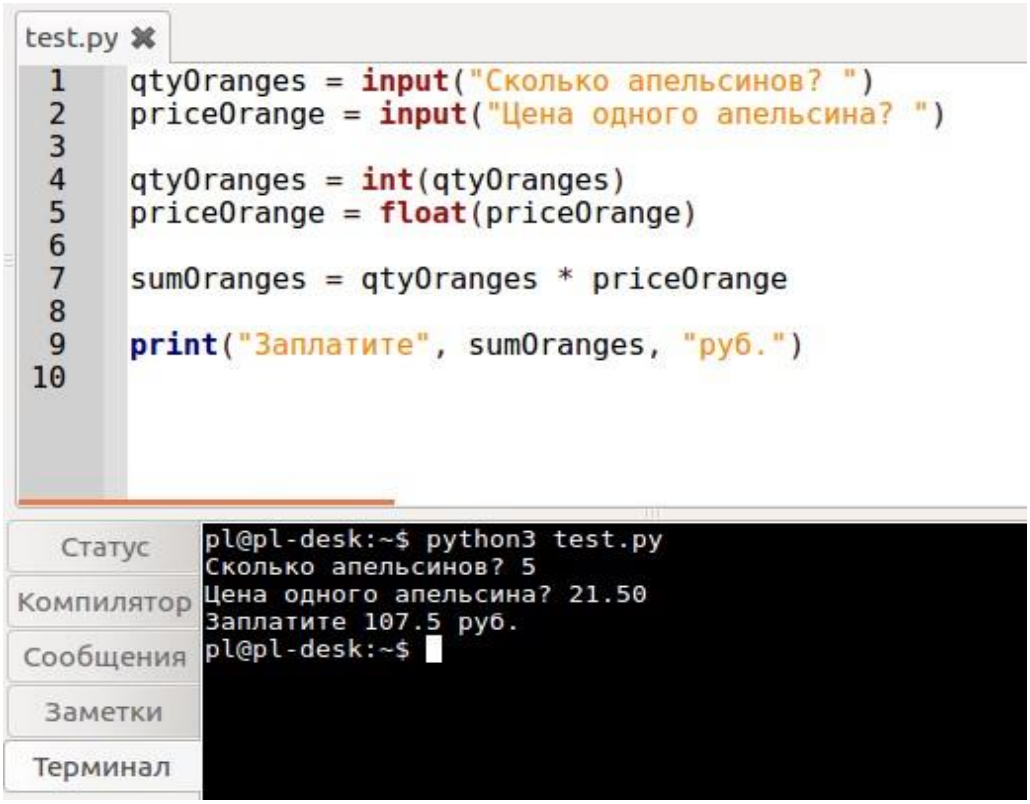
Программаны ишке киргизгенден соң компьютер биринчи катар, андан кийинкиси экинчи катарга жазылышын күтөт. Алар nameUser жана cityUser өзгөрмөлөрүнө ыйгарылат. Андан кийин бул өзгөрмөлөрдүн маанисин форматталган жыйынтыктын жардамы менен көрсөтүлөт.

Жогорудагы сценарий мыкты деп айтууга болбойт. Колдонуучу программанын андан эмнени каалаарын кайдан билет? Адамды чаташтырбоо үчүн input () функциясы үчүн атайын ыкчам параметр каралган. Бул чакыруу input () чакырылганда көрсөтүлөт. Өркүндөтүлгөн программа төмөнкүдөй болушу мүмкүн:



16-сүрөт

Программа сап менен камсыздалгандыгын эске алыңыз. Эгер сиз сан киргизсеңиз дагы, input () функциясы анын сап көрүнүшүн кайтарып берет. Бирок сан алыш керек болсочу? типин түрүн өзгөртүү функцияларын колдонуу.



17-сүрөт

Бул учурда int () жана float () функцияларын колдонуп, qtyOranges жана priceOrange өзгөрмөлөрүнүн сап маанилери, тиешелүүлүгүнө жараша бүтүн жана чыныгы санга айландырылат. Андан кийин, жаңы эле сандык маанилер ошол эле өзгөрмөлөргө ыйгарылат.

Программанын кодун кыскартууга болот, эгер типтин өзгөрүүсүн ошол эле катардын коддун input():функциясы чакырылган ошол эле саптарында аткарылса:

qtyOranges = int(input("Сколько апельсинов? "))
priceOrange = float(input("Цена одного апельсина? "))
sumOranges = qtyOranges * priceOrange print("Заплатите",
sumOranges, "руб.")

Алгач input () функциясы аткарылат. Ал int () же float () функциясы дароо санга айландырган сапты кайтарат. Андан кийин гана өзгөрмө дайындалат, башкача айтканда, ал дароо эле сандык маанини алат.

Колдонулган адабият Пол Бэрри. Изучаем программирование на Python. 2-е издание.

Үй тапшырмасы

1. Колдонуучудан төрт номерди сураңыз. Биринчи экөөнү өзүнчө, экинчисин өзүнчө эсептеңиз. Биринчи сумманы экинчисине бөлүңүз. Жыйынтыгын ондуктан кийин эки цифра камтылгандай кылып экранда көрсөтүңүз.

Тема 6. Логикалык туюнтмалар жана операторлор 6, 7-сабак (лабораториялык)

Максаты: логикалык операторлорду жана татаал логикалык туюнтмаларды өздөштүрүү, программада колдоно билүү

Негизги суроолор: Жалпы түшүнүк. Логикалык операторлор. Татаал логикалык туюнтмалар.

Студенттердин мурунку алган билимдери: “Информатика”, “Кесиптик математика”

Окуунун натыйжасы: логикалык туюнтма түшүнүгүн билет. Python программасында логикалык операторлорду, татаал логикалык уюнтмаларды колдоно алат.

Тема боюнча маалымат:

Көпчүлүк учурда чыныгы жашоодо биз айтылган сөздөргө макулбуз же четке кагабыз. Мисалы, сизге 3 жана 5 сандарынын суммасы 7ден чоң деп айтылса, макул болосуз: "Ооба, бул туура". Эгерде кимдир бирөө үч менен бештин суммасы жетиден кем деп айтса, анда сиз мындай билдирүүнү жалган деп эсептейсиз.

Мындай сөз айкаштары эки гана мүмкүн болгон жоопту сунуш кылат - же туюнтма “ооба” туюнтма чын деп бааланганда "чындык", же “жок” аныктоо жалган деп бааланганда "жалган". Программалоодо жана математикада, эгерде туюнтманы натыйжасы чын же жалган, ооба же жок болушу мүмкүн болсо, анда мындай туюнтма логикалык деп аталат. Мисалы, 4> 5 логикалык туюнтма, анткени анын жыйынтыгы чыныгы же жалган деп бааланат. 4 + 5 сөз айкашы логикалуу эмес, анткени анын аткарылышынын жыйынтыгында сан келип чыгат.

Акыркы сабакта биз маалыматтардын үч түрү - бүтүн жана чыныгы сандар, ошондой эле саптар менен таанышканбыз. Бүгүн биз төртүнчүсүн - логикалык маалыматтын (тип bool) тибин киргизебиз. Бул типтин эки гана маниси бар: True (чын) и False (жалган). >>> a = True

```
>>> type(a)
<class 'bool'>
>>> b = False
>>> type(b)
<class 'bool'>
```

Бул жерде (a) өзгөрмөсүнө True “чын” маниси берилген, андан кийин анын түрү Pythondогу type () функциясы аркылуу текшерилген. Интерпретатор бул bool классындагы өзгөрмө деп билдирди. Бул учурда "класс" жана "маалыматтардын түрү" түшүнүктөрү бирдей. (B) өзгөрмөсү да логикалык маани менен байланыштуу.

Программалоодо адатта False=0, ал эми True=1 барабарлайт. Буга ынануу үчүн логикалык маанини бүтүн типке айландырууга болот

```
>>> int(True)
1
>>> int(False)
0
```

Тескерисинче кандайдыр бир маанини лоикалык типке да айландырса болот >>>

```
bool(3.4)
True
>>> bool(-150)
True
>>> bool(0)
False
>>> bool(' ')
True
>>> bool('') False
```

Бул жерде мындай эреже иштейт: баардык ноль жана боштук эместер “чын” болуп саналат.
Логикалык операторлор
Табигый тилде (мисалы, кыргыз тилинде) сүйлөө менен биз "барабар", "чоң", "кичине" деген сөздөр менен салыштырууларды белгилейбиз. Программалоо тилдеринде математикада колдонулган белгилерге окшош атайын белгилер колдонулат:> (чоң), <(кичине),> = (чоң же барабар), <= (кичине же барабар), == (барабар) ,! = (барабар эмес).
Pythondo =(барабар) белгиси менен өзгөрүлмөлөрдү ыйгарууда колдонобуз ал эми == (эки жолу барабар) белгисин салыштырууда колдонобуз. Муну алмаштрп албагыла. Ыйгаруу жана салыштыруу ар кандай операциялар. >>> a = 10
>>> b = 5
>>> a + b > 14
True
>>> a < 14 - b
False
>>> a <= b + 5
True
>>> a != b
True
>>> a == b
False
>>> c = a == b
>>> a, b, c
(10, 5, False)
Бул мисалда c = a == b туюнтмасы эки туюнтмадан турат. Биринчиден, a жана b өзгөрмөлөрүн салыштыруу (==) бар. Андан кийин, логикалык иштин натыйжасы c өзгөрмөсүнө ыйгарылат. A, b, c туюнтмасы экранга өзгөрмөлөрдүн маанилерин жөн гана басып чыгарат.
Татаал логикалык туюнтмалар

KByte> = 1023 сыяктуу логикалык туюнтмалар жөнөкөй, анткени алар бир гана логикалык операциясын аткарышат. Бирок, иш жүзүндө татаал сөз айкаштарына муктаждык көп болот. Эки жөнөкөй сөз айкашынын натыйжасына жараша Ооба же Жок деп жооп алышыңыз керек. Мисалы, "эшикте кар жаап жатат же жамгыр жаап жатат", "жаңылыктар өзгөрмөсү 12ден жогору жана 20дан аз".

Мындай учурларда эки же андан ашык жөнөкөй логикалык сөздөрдү айкалыштырган атайын операторлор колдонулат. Логикалык (and) (жана), (же) (or).деп аталган эки оператор кеңири колдонулат

Жана операторун колдонгондо Чындыкка жетүү үчүн, ушул оператор байланыштырган оң жана сол жакта турган эки жөнөкөй туюнтманын натыйжалары чын болуш керек. Эгер жок дегенде бир учурда жыйынтык Жалган болсо, анда бүтүндөй татаал туюнтма мааниси жалган болот.

Же операторун колдонгондо Чындыкка жетүү үчүн, комплекске киргизилген жок дегенде бир жөнөкөй туюнтманын натыйжасы чын болушу керек. Же операторуна байланыштуу, татаал туюнтма анын эки жөнөкөй туюнтмасы жалган болгондо гана жалган болуп калат.

X өзгөрмөсүнө 8 (x = 8), y өзгөрмөсүнө 13 (y = 13) маани берилген деп коёлу. Буль сөзү y <15 жана x> 8 төмөнкүдөй аткарылат. Алгач y <15 туюнтмасы аткарылат, анын

натыйжасы True болот. Андан кийин x> 8 сөз айкашы аткарылат, анын жыйынтыгы False болот. Андан кийин туюнтма Чындыкка жана Жалганга кыскарат, ал Жалганды кайтат. >>> x = 8
>>> y = 13
>>> y < 15 and x > 8
False

Эгерде биз туюнтманы мындайча жазсак: x> 8 жана y <15, анда ал дагы False деп кайтып келмек. Бирок y <15 салыштыруу котормочу тарабынан жүргүзүлбөйт, анткени аны жасоонун кажети жок. Кантсе да, биринчи жөнөкөй логикалык туюнтма (x> 8) буга чейин жалган болуп кайткан, жана, операторунда болсо, бүтүндөй сөз айкашын жалган кылат.
>>> y < 15 or x > 8
True

Python ошондой эле логикалык эмес, башкача айтканда, жокко чыгаруу операторуна ээ. Чындыкты калпка, калпты чындыкка айландырат. Бул бирдиктүү, анткени ал экилик жана жана же абалындагыдай анын оң жана сол жагына эмес, андан кийинки бир эле сөзгө тиешелүү. >>> not y < 15 False
Здесь y < 15 возвращает True. Отрицая это, мы получаем False.
Бул жерде y <15 True кайтарат. Муну жокко чыгаруу менен, биз Falseге ээ болобуз. >>>
a = 5
>>> b = 0
>>> not a
False
>>> not b True
Число 5 трактуется как истина, отрицание истины дает ложь. Ноль приравнивается к False. Отрицание False дает True.
5 саны чындык деп чечмеленет, чындыкты тануу жалган берет. Нөл Жалганга барабар. Жалганды четке кагуу Чындыкты берет.

Колдонулган адабият
Пол Бэрри. Изучаем программирование на Python. 2-е издание.

Үй тапшырмасы
Колдонуучудан эки сан сураган жана биринчи сан экинчисинен чоң же чоң эместигине жараша True же False чыгарган программа түзүү.

Тема 8. Программалоонун бутактануу түрү. Шарттуу оператор 8-сабак (лекция)

Негизги суроолор: Бутактануу алгоритм түшүнүгү. Python программалоо тилинде шарттуу операторду колдонуу. Бутактануу программасынын структурасы. Бир нече бутактануу ifelif-else
Студенттердин мурунку алган билимдери: “Информатика”, “Кесиптик математика”
Окуунун натыйжасы: бутактануу алгоритмин блок схемасын түзө билет. Python программасында шарттуу операторду жана бир нече бутактануу if-elif-else колдонуп программа түзө алат.
Тема боюнча маалымат:

Программанын аткарылуусу сызыктуу болушу мүмкүн, башкача айтканда, туюнтмалар биринин артынан экинчиси аткарылып, программанын бир да сабы аткарылбай калбайт. Бирок, программаларда мындай учурлар көп кездешпейт. Программаны аткарууда, белгилүү бир шарттарга жараша, анын айрым бөлүктөрү аткарылбай калып, ал эми калгандары аткарылышы мүмкүн. Башкача айтканда, программа шарт коюп аткарылуучу, программалоо тилинин атайын курулушу - шарттуу оператор тарабынан жүзөгө ашырылуусу мүмкүн.

Келгиле, мисал келтирип карап көрөлү. Адам өзүнө ылайык план түзүп жашайт. План - бул алгоритм, ал эми анын программалык коду аткарыла турган иштери деп атасак. 18.00гө белгиленген график - бассейнге баруу. Бирок, бизге, бассейнден суу агып кеткендиги жөнүндө маалымат келсе, сууда сүзүү сабагын жокко чыгаруу, башкача айтканда, график программасынын жүрүшүн өзгөртүү керек экендигин билип, бассейнге баруунунун ордуна башка жумушун аткарууну чечебиз.

Мындай сызыктуу эмес иш-аракеттерди компьютердин программасында ишке ашырса болот. Мисалы, коддун бир бөлүгү белгилүү бир өзгөрмөнүн белгилүү бир маанисинде гана аткарылат. Программалоо тилдеринде болжол менен шарттуу операторду төмөнкү конструкциясы колдонулат: if логикалык_туюнтма {

туюнтма 1;

туюнтма 2;

…

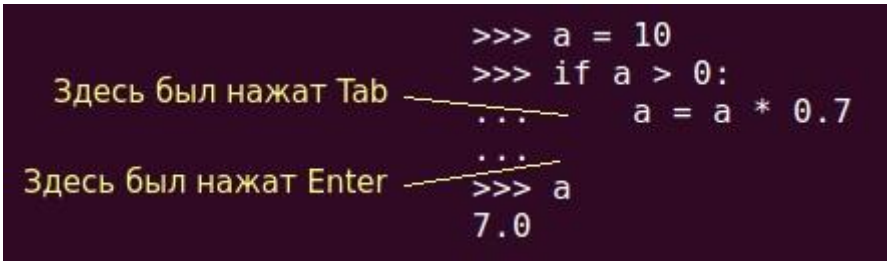
}

сөздүк алгоритмге төмөнкүчө которууга болот: эгер логикалык туюнтма чын болуп чыкса, анда тармал кашаанын ичиндеги сөздөр аткарылат; логикалык туюнтма жалган болсо, анда тармал кашаанын ичиндеги код аткарылбайт. Англис тилинен "if" "эгер" деп которулат. If логикалык_туюнтма конструкцисын шарттуу операторду башы деп аталат. Тармал кашаанын ичиндеги сөз айкаштары - шарттуу операторду телосу деп аталат. Тело көптөгөн туюнтмаларды камтышы мүмкүн, же бирөөнү гана, же бош дагы болушу мүмкүн.

Python программалоо тилинде шарттуу операторду колдонуу

if n < 100: b = n + a

Pythondo тармал кашаалардын ордуна кош чекит колдонулат. Киргизилген кодду бөлүү, башкача айтканда, операторду телосу оступтардын эсеби менен жетет. Программалоодо төрт пробелге барабар болгон отступ кылуу кадимки көрүнүш. Бирок, клавиатурадагы Tab баскычын колдонсо да болот. Көпчүлүк программалоо чөйрөлөрү сиз кош чекитти жана жаңы сапты койгондо эле автоматтык түрдө отступ коет. Бирок, интерактивдүү режимде иштеп жатканда кол менен отступ коюуу керек.



18-сүрөт

Шарттуу операторду телосунда табуу, бул жерде үч чекит менен көрсөтүлгөн. Скрипт менен файл түзүүдө мындай чекиттер, ошондой эле >>> чакырыгы да болбошу керек.

Python тили ачык синтаксистүү жана окууга оңой коду бар тил деп эсептелет. Бул, кашаа жана үтүрлүү чекит жардамчы элементтерди минималдаштыруу жолу менен ишке ашат. Туюнтмаларды бөлүү үчүн жаны катарга өтүүнү колдонобуз, ал эми киргизилген туюнтмаларды белгилөөдө – катардын башынан отступ коюуу колдонулат. Башка тилдерде дагы бул программалоо стили колдонулат, бирок адамга окумдуулук үчүн гана Pythonдо ал синтаксистик эреженин деңгээлине көтөрүлгөн.

Жогорудагы мисалда логикалык туюнтма n <100 болуп саналат. Эгер ал чыныгы мааниге ээ болсо, анда b = n + a кодунун сабы аткарылат. Эгерде логикалык туюнтма жалган болсо, анда b = n + a аткарылбай калат.

Бул мисал контексттен алынган, өзүнчө иштебей калат. Программанын толук версиясы төмөнкүдөй болушу мүмкүн:

```
b = 0 a =
50 n = 98
if n < 100:
b = n + a
print(b)
```

Коддун акыркы катары print(b)шарттуу оператор катарына кирбейт, себеби анын алдында отступ жок. Ал шарттуу операторд о уялануучу (вложенн ый) болуп эсептелин бейт, демек, ага таандык эмес.

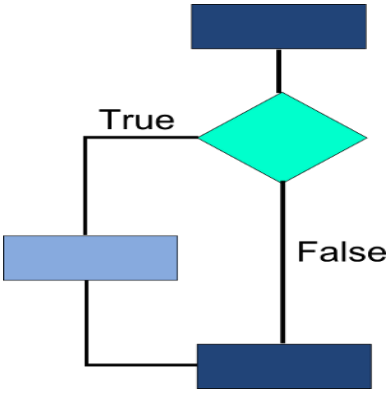
Өзгөрүлмө N= 98, бул 100гө жетпегендиктен, b= 148ге. Бул маани экранда көрсөтүлөт. Эгерде n өзгөрмөсү башында, мисалы, 101 маанисине байланган болсо, анда 0 көрсөтүлөт, n 101ге барабар болсо, шарттуу операторду башындагы логикалык туюнтма жалган болуп кайтып келет. Бул тело аткарылбай, b өзгөрмөсү өзгөрүлбөйт дегенди билдирет.

Бутактануу программасынын структурасы

19-сүрөт

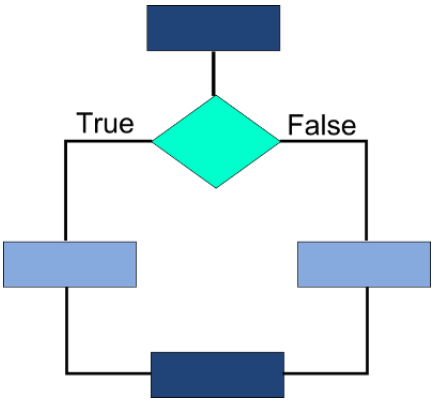
Программанын негизги бутагы ар дайым аткарылат, ал эми уюлдук код (вложенный код) качан гана шарттуу операторду башын белгилеген кочкул жашыл катарда чын (истина) болсо.

Чакан программалар үчүн кээде аткаруу агымын чагылдырган блок-схемалар түзүлөт. Блок-схемалардын тилинде айрым конструкциялар алардын формалары менен көрсөтүлөт. Ошентип, иш-аракеттер блогу тик бурчтук менен, ал эми логикалык туюнтмасы ромб менен көрсөтүлөт. Жогорудагы код үчүн блок-схема төмөнкүдөй болушу мүмкүн:



20-сүрөт

Шарттуу оператор бир бутакты эмес, экөөнү камтышы мүмкүн, ошону менен толук кандуу бутактанууну ишке ашыра.



21-сүрөт

Логкалык туюнтмасы жалган кайтарса, программанын агымы түз негизги бутакка кайтпайт. True уюулунан айырмаланып, False учуруна бөлөк уюлдук бутак бар. Башка сөз менен айтканда, шарттуу операторду кеңейтилген версиясына туш болгондо, программанын агымы, жок дегенде, бир нече уюлдук кодду аткарбастан, негизги бутакка кайтып келбейт.

Программалоо тилдеринде, эки бутакка бөлүү else блогун кошуу менен ишке ашат жана if– else пайда болот. (эгерде-антпесе). Синтаксис мындайча көрүнөт: if логическое_выражение {

```
   выражение                1;
выражение 2;
   …
} else
{
   выражение 3;
   … }
```

Эгерде if операторунун шарты жалган болуп чыкса, анда else инструкциясындагы блогунун коду аткарылат. Эки бутактануу тең аткарыла турган жагдай мүмкүн эмес. Же if ге таандык код же else ге таандык код аткарылат. Else де эч качан логкалык туюнтма болбойт Python программалоо тилиндеги else аркылуу бутактануу:

```
tovar1 = 50 tovar2 =
32 if tovar1+ tovar2 >
99 :
   print("99 рублей недостаточно")
else:     print("Чек оплачен")
```

If теги логикалык туюнтма стандарттуу көрүнбөөрүн эске алуу керек б.а. a> b дай жөнөкөй эмес. ЖАНА же ЖЕ логикалары аркылуу эки жөнөкөй туташканда, ал жерде жөнөкөй бир өзгөрүлмө, сан, сөз True же False жана тататаал логикалык туюнтма турушу мүмкүн. a = ?
if a: a = 1

Если вместо знака вопроса будет стоять 0, то с логической точки зрения это, значит выражение в if не будет выполнено. Если а будет связано с любым другим числом, то оно будет расцениваться как True, и тело условного оператора выполнится. Другой пример: Эгерде суроо белгисинин ордунда 0 турса, анда логикалык эрежеге ылайык бул False, анда if аткарылбайт. Эгерде а кандайдыр бир башка сан менен байланышкан болсо, анда ал True катары бааланат жана шарттуу оператордун телосу аткарылат. Башка мисал: a = 5 > 0 if a: print(a)

Бул жерде а логикалык маани менен байланыштырылган. Бул учурда, бул чын. A = 5> 0 туюнтмасында ыйгаруу, салыштыруу операторунан кийин аткарылат, ошондуктан адегенде 5> 0 аткарылып, андан кийин анын жыйынтыгы а га барабарболот. Эсиниздеболсун, эгерде билдирүүлөрдүн аткарылышынын ырааттуулугуна күмөн санасаңыз, анда кашааны пайдаланыңыз, мисалы: a a = (5 > 0).

Үчүнчү мисал: if
a > 0 and a < b:
print(b - a)

Бул жерде, салынган коддун аткарылышы үчүн, а нөлдөн чоң жана бир эле учурда bдан кичине болушу керек. Ошондой эле Pythondo, башка программалоо тилдеринен айырмаланып, татаал логикалык туюнтма үчүн төмөнкү кыскартылган жазуу мүмкүн: if 0 < a < b:
   print(b - a)

Бышыктоочу суроолор

Программада шартты берүүдө кайсы операторду колдонобуз

Бир нече бутактануунун блок схемасына жана программада иштөөсүнө аныктама киргиз.

Колдонулган адабият

Пол Бэрри. Изучаем программирование на Python. 2-е издание.

Тема 9. Программалоонун бутактануу түрү. Бир нече бутактануу if-elif-else 9-сабак (лабораториялык)

Максаты: if, if-elif-else операторлорун колдонуп программа түзүү

Негизги суроолор: Бутактануу алгоритм түшүнүгү. Python программалоо тилинде шарттуу операторду колдонуу. Бутактануу программасынын структурасы. Бир нече бутактануу ifelif-else

Студенттердин мурунку алган билимдери: “Информатика”, “Кесиптик математика”

Окуунун натыйжасы: бутактануу алгоритмин блок схемасын түзө билет. Python программасында шарттуу операторду жана бир нече бутактануу if-elif-else колдонуп программа түзө алат.

Тема боюнча маалымат:

Буга чейин, if оператору кандай иштээрин карап чыкканбыз. Анын кеңейтилген версиясы if-else менен, эки бөлөк бутактанууну аткарууну ишке ашыра алабыз. Бирок, программанын алгоритминде экиден ашык жолду тандоо болушу мүмкүн, мисалы, үч, төрт, ал тургай беш. Бул учурда, бир нече бутактактануу зарыл экендиги байкалат.

Келгиле, белгилүү бир мисалды карап көрөлү. Мисалы колдонуучунун жашына жараша, ага белгилүү бир видео контент сунушталат. Ошол эле учурда 3 жаштан 6 жашка чейинки, 6 жаштан 12 жашка чейинки, 12 жаштан 16 жашка чейинки, 16 жаштан жогору топтор белгиленет. Жалпысынан 4 диапазон бар. Тапшырманы if-else конструкциясын гана колдонуп кантип ишке ашырсак болот.

Эң жөнөкөй жооп - төмөнкү ирээттелген шарттуу операторлордун жардамы менен киргизилген жаш санынын белгилүү бир диапазондо экендигин ырааттуу текшерүү болуп саналат:

```
old = int(input('Сиздин жашыныз: '))

print('Сунушталган:', end=')

if 3 <= old < 6:
   print('Жаш бала')

if   6   <=   old   <   12:
print('Бала')

if 12 <= old < 16:
   print('Өспүрүм')

if 16 <= old:
   print('Чоң адам')
```

Эскертүү. Кинонун аталышы кош тырмакча менен чыгарылат. Ошондуктан, программада саптарды аныктоо үчүн кош эмес, бирдик тырмакча колдонулат.

Сунушталган код сонун иштейт, бирок бир маанилүү "бирок" бар. Ал эффективдүү эмес, анткени анда келтирилген ар бир if бири бирине байланышпаган өзүнчө алынган if оператор. Процессор убакытты жана "нервдерди" алардын ар биринин иштетүүсүнө

сарптайт, бул керек болбой калса дагы. Мисалы, 10 саны киргизилген. Биринчи if де, логикалык туюнтма жалганды кайтарат, аткаруу экинчи if ке өтөт. Башындагы логикалык туюнтма чындыкты кайтарат жана тело аткарылат. Бүттү, программа ошол жерде токтоп калышы керек болчу.

Бирок, кийинки if мурункусуна эч кандай тиешеси жок ошондуктан, кийин old өзгөрүлмөсүнүн маанисиси 12ден 16га чейинки аралыкта экендиги текшерилет, буга зарылчылык жок болсо да. Андан кийин акыркы if теги логикалык туюнтма False болоору белгилүү болсо да каралат. Эмне кылуу керек? Жооп болуп, шарттуу операторлорду бири биринин ичине жайгаштыруу саналат.

```
old = int(input('Сиздин жашыныз: '))

print('Сунушталган:', end=' ')

if 3 <= old < 6:
   print('Жаш бала')    else:
   if 6 <= old < 12:
print('Бала')     else:
      if 12 <= old < 16:
print('Өспүрүм')        else:
if 16 <= old:
print('Чоң адам')
```

Коддун ушул вариантынын агымын карап көрөлү. Биринчиден, биринчи if деги шарт текшерилет (эң сырттагысы). Эгер бул жерде True алынган болсо, анда if тин телосу аткарылат, ал эми else бутагына кайрылбайбыз дагы, анткени ал качан гана if шартында жалган пайда болгондо гана иштейт.

Эгерде сырткы If False кайтарса, программанын агымы ага туура келген сырткы else ге кирет. Анын денесинде өзүнүн else си менен башка if турат. Эгерде кийирилген сан 6 дан 12 ге чейинки диапазонго туура келип калса, анда, камтылган if тин телосу аткарылат, андан кийин программа аяктайт. Эгер сан 6дан 12ге чейинки диапазонго түшпөсө, анда else бутагына өтүү аткарылат. Анын телосунда өзүнүн шарттуу оператору бар, ал үчүнчү уя деңгээлине ээ.

Ошентип, акыркы текшерүү (16 <= old) га котормочу качан гана мурункулардын баары False кайтарганда гана жетет. Эгер программанын аткарылу учурунда True пайда болсо, анда кийинки текшерүүлөрдүн бардыгы алынып салынат, бул процессордун ресурстарын үнөмдөйт. Мындан тышкары, программанын аткарылышынын мындай логикасы тууралаак.

Эми кийинки суроону өзүбүзгө берели. Кандайдыр бир жол менен көп тармактуу кодду оптималдаштырып, уяланган шарттуу операторлордон тепкич курбай коюуга болобу? Көпчүлүк программалоо тилдеринде, чегинүү программист тарабынан окуу женил болушу үчүн колдонулат, бирок синтаксистик маанисиси жок, мындай стиль көп колдонулат: if логикалык_туюнтма {

```
   … ; }
else if логикалык_туюнтма {
   … ;
} else if логикалык_туюнтма
{
```

```
   … ;
} else
{
… ;
}
```

Тиркелүүнүн бир гана деңгээли бардай көрүнүшү мүмкүн, жана if үчүн else if тей көрүнгөн жаңы кеңейүүлөр пайда болушу мүмкүн. Бирок бул ушундай болуп көрүнүшү гана окшойт. Чындыгында, else ден кийин турган if ушул else ге тиркелүүчү болуп саналат. Жогорудагы келтирилген схема ушундай эле

```
if логикалык_туюнтма {
   … ; } else      if
логикалык_туюнтма {
       … ;
}     else
       if логикалык_туюнтма {
          … ;
}         else
{
… ;
       }
```

Тилмеч же компилятор аны ушундайча "түшүнөт". Бирок, адамга биринчи вариантты кабыл алуу оңой деп эсептешет.

Pythonдо камтылган if ти сырттагы elseге көтөрүү, иштебей калат, анткени анда чегинүүлөр жана жаңы тилкелер синтаксистик мааниге ээ. Ошондуктан, Python тилинде чыныгы бир денгээлде камтылган бир нече бутактануу мүмкүнчүлүгүн түздү, ал elif бутактарын колдонуу менен ишке ашырылат.

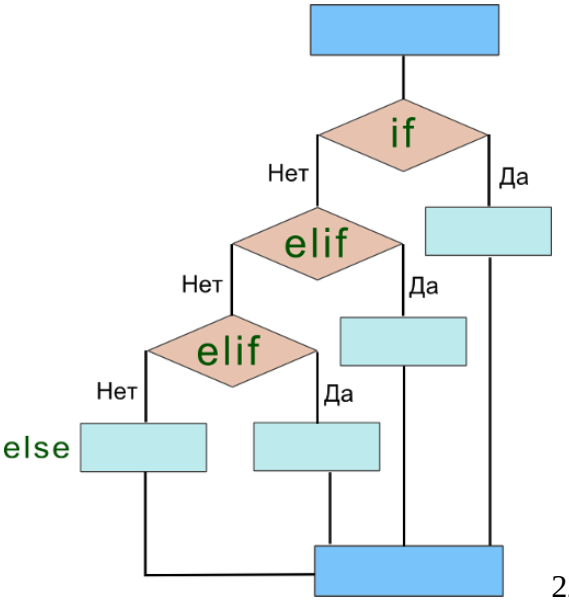
" elif " сөзү "else" сөзүнүн биринчи эки тамгасынан, андан кийин "if" сөзүнө уланган. Муну "эгер болбосо" деп которсо болот.

Else ден айырмаланып, elif баш сөзүндө if сыяктуу эле логикалык сөз айкашы камтылышы керек. Көп бутактуу конструкцияны колдонуп, программабызды кайра жазалы:

```
old = int(input('Сиздин жашыныз: '))
print('Сунушталган:', end=' ')  if 3 <=
old < 6:
   print('Жаш бала')    elif
6 <= old < 12:
   print('Бала')   elif
12 <= old < 16:
   print('Чоң адам') elif
16 <= old:
            print()
```

Акырында, бардык элифтерден кийин, учурларды обработкалоо үчүн бардык elif терден if бутагынын шартына туш келбеген else нин бир бутагы колдонулушу мүмкүн.

Толук if-elif-… -elif-else курулушунун блок-схемасын төмөндөгүдөй көрсөтсө болот:



22-сүрөт

If тин же кандайдыр бир elif тин телосу аткарыла баштаганда, программа дароо эле негизги буткк кайтат.(астынкы ачык көк тикбурчтук) андан кийинки бардык elif жана else өткөрүлүп жиберилет.

Колдонулган адабият Пол Бэрри. Изучаем программирование на Python. 2-е издание.

Үй тапшырмасы

Кандайдыр бир сан кийиргенде ал нөлдөн чоң болсо, анда экранга жооп иретинде 1 саны көрсөтүлгөн, эгерде киргизилген сан оң болбосо, анда экранда -1 көрсөтүлгөн программаны түзүү.

Тема 10. Циклдык For оператору 10, 11-сабак (лекция)

Негизги суроолор: For цикли. Range() функциясы. For жана range() циклы. Enumerate() функциясы.

Студенттердин мурунку алган билимдери: “Информатика”, “Кесиптик математика”

Окуунун натыйжасы: циклдык операторду алгоритмин блок схемасын түзө билет. Python программасында циклдык for операторун, Range() жана Enumerate() функцияларын колдонуп программа түзө алат.

Тема боюнча маалымат:

Python программалоо тилиндеги for цикли маалымат структураларынын жана башка кээ бирки объектилердин элементтерин кайталоо үчүн иштелип чыккан. Бул башка программалоо тилдеринде колдонулган счетчиктүү for цикли эмес.

Элементтердин кайталанышы эмнени билдирет? Мисалы, бизде элементтердин катарынан түзүлгөн тизме бар деп коёлу. Биринчи, андан биринчи элементти, андан кийин экинчисин, андан кийин үчүнчүсүн ж.б. ушундай тартипте катары менен алалы. Ар бир элемент менен for телосунда ошол эле амалды аткарабыз. Бизге элементтерди алардын индекстери боюнча алуунун, цикл кайсы жерден аякташы керектиги жөнүндө ойлонуунун кажети жок, жана

кийинки кайталоо маанисиз. For цикли өзү кайталанып, аягын өзү аныктайт. >>> spisok = [10, 40, 20, 30] >>> for element in spisok:
... print(element + 2)
...
12
42
22
32
For ачкыч сөзүнөн кийин element деп аталган өзгөрүлмө колдонулат. Бул жерде аты каалагандай болушу мүмкүн. Көп учурда i колдонушат. For циклинин ар бир кайталанышында, ага spisok тизмесинен кийинки элемент ыйгарылат. Ошентип, циклдин биринчи кайталанышында, element идентификатору 10 саны менен, экинчисинде 40 саны менен байланыштырылат ж.б. spisok то элементтер түгөнгөндө for цикли өз ишин аяктайт. Англис тилинен "for" "үчүн", "in" орусча "в" мүчөсү улангандай которулат. Конструкцияны программалоо тилинен адам тилине мындайча которууга болот: тизмедеги ар бир элемент үчүн төмөнкүнү аткарыңыз (циклдин денесиндегидей).
Мисалда, биз ар бир элементти 2ге көбөйтүп, аны экранга чыгардык. Тизме өзү, албетте, өзгөргөн жок: >>> spisok
[10, 40, 20, 30]
Эч жерде анын элементтерин кайра жазуу жөнүндө сөз болгон жок, алар жөн гана алынып, колдонулуп жатты. Бирок, тизменин өзүн өзгөртүү керек болуп калуусуда мүмкүн, мисалы, андагы ар бир элементтин маанисин өзгөртүү же белгилүү бир шартты канааттандырган айрым гана элементтердин маанисин өзгөртүү. Бул жерде, көпчүлүк учурларда, элементтин индексин көрсөткөн өзгөрмөсүз жасоо кыйын: >>> i = 0
>>> for element in spisok:
... spisok[i] = element + 2
... i += 1
...
>>> spisok
[12, 42, 22, 32]
Эгер биз эсептегичти колдонууга аргасыз болуп калсак, анда for циклин колдонуунун пайдасы байкалбайт. Эгерде тизменин узундугу белгилүү болсо, анда while ды колдонуу ыңгайлуу. Узундукту Python до орнотулган len () функциясынын жардамында өлчөөгө болот.
>>> i = 0
>>> while i < len(spisok):
... spisok[i] = spisok[i] + 2 # или spisok[i] += 2
... i = i + 1 # или i += 1
...
>>> spisok
[14, 44, 24, 34]
Ошондой эле, while цикли менен, биз element өзгөрмөсүнөн арылдык.
Range() функциясы

Эми Python до орнотулган range() функциясы менен таанышууга кез келди. "Range" "диапазон" деп которулат. Ал бир, эки же үч аргумент кабыл ала алат. Алардын максаты random модулундагы randrange () функциясы менен бирдей. Эгерде бирөө гана берилсе,

анда аны кошпогондо, 0дөн көрсөтүлгөн санга чейин сандар пайда болот. Эгерде экөө берилген болсо, анда аны кошпогондо, биринчиден экинчисине чейин сандар пайда болот. Эгерде үчөө берилген болсо, анда үчүнчү сан бул-кадам.

Бирок, randrange () дан айырмаланып, range () функциясы көрсөтүлгөн диапазондогу бирда кокустук санды жаратпайт. Ал кокустук сандарды таптакыр жаратпайт. Ал көрсөтүлгөн диапазондогу сандардын ырааттуулугун жаратат. Мисалы, range (5, 11) диапазону 5, 6, 7, 8, 9, 10 ырааттуулугун пайда кылат. Бирок ал тизме маалыматтарынын структурасы болбойт. Range () функциясы өзүнүн классындагы объекттерди жаратат - диапазондор:

```
>>> a = range(-10, 10)
>>> a range(-10,
10) >>> type(a)
<class 'range'>
```

Сандардын ирээт тизмегин көрө албасак дагы, ал бар жана анын элементтерине кайрылсак болот: >>> a[0]

```
-10
>>> a[5]
-5
>>> a[15]
5
>>> a[-1]
9
```

Аларды өзгөртүү мүмкүн эмес, себеби тизмелерден айырмаланып, range () объекттери өзгөрүлбөгөн тайпаларга караштуу: >>> a[10] = 100

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'range' object does not support item assignment
```

For жана range() цикли

Анда эмне үчүн for темасында range() функциялары керек болуп калды? Чындыгында, алар чогуу жакшы тандем түзүшөт. Анткени элементтерден өткөн цикл катары, айырмаланып, ал структуранын аягына жеткен-жетпегенин көзөмөлдөбөйт. Бул үчүн эсептегичти киргизүүнүн кажети жок, аны өзгөртүп, башындагы шартты текшерип кереги жок. Башка жагынан алганда, range() ошол эле тизмедеги элементтердин индекси катары колдонула турган бүтүн сандардын ырааттуулугун берет.

```
>>> range(len(spisok)) range(0,
4)
```

Бул жерде len () функциясы тизменин узунун өлчөйт. Бул учурда, ал төрткө барабар. Андан кийин, 4 саны range () функциясына өтүп, ал 0 ден 3кө чейинки сандардын ыраатын жаратат. Бул биздин тизме элементтеринин индекстери.

Эми for жана range(): ти "туташтыралы": >>>

```
for i in range(len(spisok)):
...     spisok[i] += 2
...
>>> spisok
[16, 46, 26, 36]
```

For циклинин башында элементтер такыр тизмеден эмес, range объектисинен алынат. Элементтеринин үстүнөн жазуу пландаштырылган тизме чындыгында бул жерде жок. Эгер сиз тизменин узундугун алдын ала билсеңиз, анда башталышы төмөнкүдөй болушу мүмкүн: for i in range(4), анда i циклдин телосунда кантип колдонулаары экинчи суроо. Эскертүү. i идентификатордун ордуна башка болушу мүмкүн.

Enumerate() функциясы

Pythondo дагы бир орнотулган функциясы бар, ал көбүнчө for дун башында колдонулат. Бул enumerate () функциясы. Эгер range() тизмектин элементтеринин индекстерин гана алууга мүмкүнчүлүк берсе, анда enumerate() элементтин индексинен жана элементтин маанисинен турган кортеждердин жуптарын жаратат.

```
>>> spisok = [16, 46, 26, 36]
>>> for i in enumerate(spisok):
...     print(i) ...
(0, 16)
(1, 46)
(2, 26)
(3, 36)
```

Бул картеждерди таңгактан чыгарууга болот, башкача айтканда, индексти жана маанини циклдин денесинен алууга болот: >>> for item in enumerate(spisok):

```
...     print(item[0], item[1])
...
0 16
1 46
2 26
3 36
```

Бирок, көбүнчө, ушуну in дин алдынан эки өзгөрүлмөнү колдонуп for дун башында кылат:

```
>>> for id, val in enumerate(spisok):
...     print(id, val)
...
0 16
1 46
2 26
3 36
```

Enumerate () функциясы ушунчалык жакшы болсо, анда for дун башнда range() ти колдонуп эмне кереги бар? Чындыгында, сизге ушундай жеңил болбосо гана, эч кандай колдонуп кереги жок. Мындан тышкары, маанисинин кереги жок, индекстер гана керек болгон жагдайлар да болот. Бирок, бир нерсени эске алуу керек. Enumerate () функциясы объектитераторду кайтарып берет. Мындай объектилер маанилерди жаратканда, алар "бош" болуп калат. Сиз алардын үстүнөн экинчи жолу баса албайсыз.

Range () функциясы кайталануучу объектини кайтарат. Мындай объект итераторго айланышы мүмкүн, ал өзүнчө андай эмес.

Range() жана enumerate() For дун башында колдонулганда, эч кандай айырмачылыгы жок, анткени range() жана enumerate() объектилер өзгөрүлмөлүүлөргө ыйгарлбайт жана цикл бүткөндөн кийин жоголуп кетет. Бирок биз бул объектилерди өзгөрүлмөлөргө ыйгарсак, айырма болот:

```
>>> r_obj = range(len(spisok))
>>> e_obj = enumerate(spisok)
>>> for i in r_obj: ...     if i ==
1: ...             break ...
>>> for i in e_obj: ...
if i[0] == 1:
...             break ...
>>> for i in r_obj:
...     print(i) ...
0
1
2
3
>>> for i in e_obj:
...     print(i) ...
(2, 26)
(3, 36)
```

Алгач, 1 индекстүү элементтен объекттердеги элементтерди алууну токтотобуз. Объекттерди for цикли аркылуу адаганда, r_obj учурунда, өтүү башнан башталат, ал эми e_obj учурунда, токтогон жеринен уланат. E_obj объекти мурун алынган элементтерди камтыбайт.

Бышыктоочу суроолор
For циклинин блок схемасын түзүп бер
Range() функциясына аныктама бер
For жана range() цикли иштөө тартиптери
Enumerate() функциясынын программада иштетилишин көрсөт

Колдонулган адабият
Пол Бэрри. Изучаем программирование на Python. 2-е издание.

Үй тапшырмасы
У=х>10 мисалга For операторун колдонуп программа түзүү.

Тема 12. Программалоодогу циклдар. While циклы 12-сабак (лабораториялык)

Максаты: циклдык операторлорду колдонуп программа түзүүнү өздөштүрүү
Негизги суроолор: While циклы. While циклинин блок-схемасы. while циклин колдонууда эки өзгөчө кырдаал.
Студенттердин мурунку алган билимдери: “Информатика”, “Кесиптик математика”
Окуунун натыйжасы: While циклдык оператордун алгоритмин блок схемасын түзө билет. Python программасында циклдык While операторунун эки өзгөчө кырдаалына мисал келтирет жана программада колдонот.
Тема боюнча маалымат:
Контурлар программалоонун бөлүктөрү шарттуу шарттар сыяктуу эле маанилүү. Циклдерди колдонуп, коддун бөлүктөрүнүн аткарылышын кайталоону уюштура аласыз.

Мунун зарылдыгы көп учурда келип чыгат. Мисалы, колдонуучу ырааттуу түрдө сандарды киргизет жана алардын ар бирин жалпысына кошуу керек. Же натуралдык сандардын катарларынын квадраттарын жана ушул сыяктуу маселелерди көрсөтүү керек. While циклы "While" англис тилинен "чейин" деп которулат. Бирок "коштошуу" деген мааниде эмес, " ушундай жыйынтыкка ээ болгонго чейин бул аткарлат" деген мааниде.

Универсалдуу цикл деп айта алабыз. Ал структураланган программалоону колдогон бардык тилдерде, анын ичинде Pythondo бар. Анын синтаксисин бардык тилдер үчүн кыскача келтирүүгө болот:

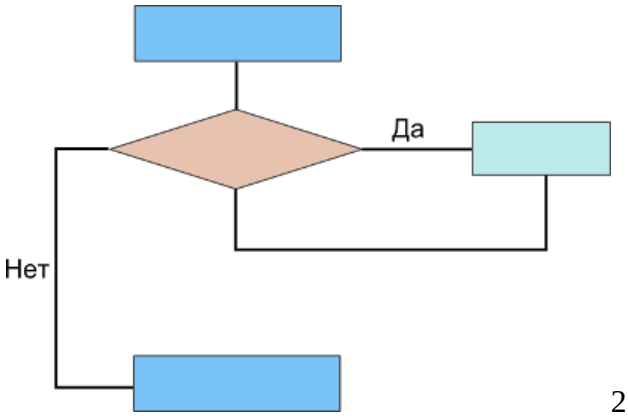
while логикалык_туюнтма {
туюнтма 1;
…
туюнтма n; }

Бул шарттуу if оператору сыяктуу. Бирок, циклдик операторлордо алардын телосу бир нече жолу кайталоо менен аткарылышы мүмкүн. Эгерде, баштагы логикалык туюнтма чыныгы мааниге ээ болсо, анда дене бир жолу аткарылат. Андан кийин, программанын аткарылуу агымы негизги бутакка кайтып келип, шарттуу оператордун бүтүндөй конструкциясынын астында төмөнкү сөздөрдү аткарат.

Ал эми, анын телосу аткарылгандан кийин, айлануу циклдин башына кайтып келип, шартты кайрадан текшерет. Эгерде логикалык туюнтма чыныгы мааниге ээ болсо, анда айлануу дагы бир жолу аткарылат. Андан кийин биз кайрадан башына барабыз ж.б.у.с.

Качан гана логикалык амал башына жалган маани кайтарганда гана цикл өз ишин токтотот, башкача айтканда, циклдин аткарылыш шарты аткарылбай калат. Андан кийин аткаруу агымы амалга аралашат да бүткүл циклдин астында жайгашкан сөздөргө өтөт. "циклден чыгуу" б.а. цикл аягын чыгат.

While циклинин блок-схемасы



23- сүрөт

Анда ачык көк түстөгү төрт бурчтуктар программанын негизги бутагын көрсөтөт, ромб - логикалык туюнтма менен циклдин баш аты, ток көк түстөгү тик бурчтук - циклдин денеси.

While циклин колдонууда эки өзгөчө кырдаал

Эгерде циклге биринчи киргенде, логикалык туюнтма False деп кайтарса, анда циклдин тулкусу бир жолу дагы аткарылбайт. Бул кырдаалды нормалдуу деп эсептесе болот, анткени белгилүү бир шарттарда программанын логикасында цикл денесинин сөздөрүн аткаруунун кажети жок деп болжолдонушу мүмкүн.

Эгерде баштагы логикалык туюнтма эч качан False деп кайтпаса, бирок ар дайым Trueге барабар болуп калса, анда анын денесинде циклден (break) аргасыз чыгуу же программанын функцияларынан чыгууга чакыруу болбосо, цикл эч качан бүтпөйт - quit (), exit () Python

учурда. Эгерде цикл чексиз көп жолу кайталанып жана кайталана берсе, анда программа циклге барат. Бул учурда ал тоңуп (зависает), өзүнөн өзү бүтө албайт.

Өзгөчө кырдаалдар жөнүндө сабактан алган мисалыбызды эстейли. Колдонуучу бүтүн санды киргизиши керек. Input () функциясы сапты кайтарып бергендиктен, программанын коду int () функциясын колдонуп, киргизүүнү бүтүн типке которушу керек. Бирок, сандык эмес белгилер киргизилген болсо, анда ValueError өзгөчө учуру киргизилет жана башка пункту менен иштейт. Ушуну менен программа аяктады.

Башка сөз менен айтканда, эгерде программа мындан аркы иш-аракеттерди сан менен жүргүзсө (мисалы, паритетти текшерүү), бирок ал аны албаса, анда программанын колунан келе турган нерсе - бул өз ишин мөөнөтүнөн мурда бүтүрүү.

Бирок сиз колдонуучудан номерди киргизгенге чейин туура киргизүүнү сурап жана сурана аласыз. Муну ишке ашырган кодекс төмөнкүдөй болушу мүмкүн:

```
n = input("бүтүн сан кийириниз: ")

while type(n) != int:
try:
    n = int(n)
except ValueError:
    print("Туура эмес кийирдиниз!")
n = input("Бүтүн сан кийириниз: ") if n
% 2 == 0:
  print("Жуп сан")
else:     print("Так
сан")
```

Эскертүү 1. Python программалоо тилинде татаал көрсөтмөлөрдүн баштарынын аягына чекит коюларын унутпаңыз.

Эскертүү 2. type(n) != int туюнтмасында n (n) өзгөрмөсүнүн тиби type () функциясын колдонуп текшерилет. Эгер ал intге барабар болбосо, башкача айтканда, n мааниси бүтүн эмес, бирок бул учурда сап болсо, анда сөз айкашы чыныгы мааниге ээ болот. Эгерде n түрү int болсо, анда бул логикалык сөз айкашы жалган болот.

Эскертүү 3. Бөлүндүн калган бөлүгүн табуу үчүн Pythonдогу% оператору колдонулат. Демек, эгер сан жуп болсо, анда ал 2ге калдыксыз бөлүнөт, башкача айтканда, калганы нөлгө барабар болот. Эгер сан так болсо, калганы бирге барабар болот.

Келгиле, ушул коддун аткарылыш алгоритмин байкап көрөлү. Колдонуучу маалыматтарды киргизет, алар сап тибиндеги жана n өзгөрмөсүнө берилген. While башы n түрүн текшерет. Циклге биринчи киргенде, n түрү ар дайым сап түрү болот, башкача айтканда, ал intга барабар эмес. Демек, логикалык сөз айкашы чын болуп, циклдин тулку бөлүгүнө өтүүгө мүмкүнчүлүк берет.

Бул жерде, try аракет бутагында, сапты бүтүн типке которуу аракети көрүлөт. Эгер ал ийгиликтүү болгон болсо, анда андан башка пункту өткөрүлүп жиберилип, агым ошол эле мезгилдин башына кайтып келет.

Эми n бүтүн санга байланган, демек, анын түрү int, ал intга барабар болбойт. Ал ага тең. Ошентип, логикалык туюнтманын type(n) != int False кайтып, бүт цикл аяктайт. Андан ары, аткаруунун агымы программанын негизги бутагында жайгашкан if-else операторуна өтөт. Бул жерде шарттуу билдирүү эле эмес кандайдыр бир нерсе болушу мүмкүн,.

Артка кайталы. Эгерде try аракет денесинде бир санга өтүү аракети ишке ашпай калса жана ValueError өзгөчөлүгү таштalса, анда программанын агымы башка тармакка жөнөтүлүп, ушул жерде табылган сөздөрдү аткарат, анын акыркысы колдонуучудан маалыматты кайрадан киргизүүнү суранат. N өзгөрмөсү эми жаңы маанiге ээ болду.
Аяктагандан кийин, цикл башындагы логикалык сөз айкашы except кайрадан текшерилет.
Ал True деп кайтып келет, анткени n дагы эле сап.
Циклден чыгуу n мааниси санга ийгиликтүү которулганда гана мүмкүн болот.
Төмөнкү мисалды карап көрөлү: total = 100

```
i = 0 while i
< 5:
   n = int(input())
total = total - n     i
= i + 1

print("Калды", total)
```

Бул программада цикл "кайталоо" канча жолу болот, б.а., ал канча жолу кайталанат? Жооп: 5.

1. Баштап , i өзгөрүлмө 0 гө барабар. Циклдин башында, i <5 шарты текшерилет жана ал чын. Циклдин тулку бөлүгү аткарылат. Анда цикл айланган сайын 1 кошулуу менен i маанисин өзгөртөт.
2. Эми өзгөрүлмө i 1ге барабар. Бул бештен кичине, жана циклдин тулкусу экинчи жолу аткарылат. Анда i өзгөрөт, анын жаңы мааниси 2 болот.
3. Эки бештен кичине. Телонун цикли үчүнчү жолу аткарылат. I мааниси үчкө барабар болот
4. Үч бештен кичине. Бул кайталоодо, I ге 4 берилет.
5. Төрт баштагыдай эле 5тен кичине. I га 1 кошулат жана анын мааниси 5ке барабар болот. Бул циклдин "семантикалык жүктөмү" - бул жалпы сандардан келген сандарды ырааттуу алып салуу. I өзгөрмөсү бул учурда циклдин кайталануу эсептегичинин ролун гана аткарат. Башка программалоо тилдеринде мындай учурлар үчүн for цикли берилет, ал "счетчик цикл" деп аталат. Анын артыкчылыгы эсептегичтин өзгөрмөсүн цикл денесинде өзгөртүүнүн кажети жок, анын мааниси форманын аталышында автоматтык түрдө өзгөрөт. Pythondo for for цикли да бар. Бирок бул счетчик цикл эмес. Pythondo ал ырааттуулук элементтеринин жана башка татаал объектилердин үстүнөн кайталоо үчүн иштелип чыккан. Бул цикл жана ырааттуулук кийинки сабактарда изилденет. while үчүн счетчик зарыл эмес. Бардыгы нөлдөн чоңураак болсо, сандарды киргизүү керек деп элестетип көрүңүз. Ошондо код мындай болот: total = 100

```
while  total  >  0:
n = int(input())
   total = total - n

print("Ресурс исчерпан")
```

Бул жерде цикл канча жолу аткарылат? Белгисиз, бардыгы киргизилген мааниден көз каранды. Демек, эсептегич бар цикл кайталоонун санын билет, бирок эсептегичсиз цикл билбейт.

While цикли үчүн эң негизгиси, анын баш жагында текшерилген өзгөрүлмөлөрдүн мааниси анын денесинде өзгөрөт жана жок дегенде, качандыр бир баштагы логикалык туюнтма False деп кайтып келген учур болот. Болбосо, тутулуу пайда болот.

Эскертүү 1. total = total - n и i = i + 1 сөздөрүндө бир эле өзгөрмөнү кайталоонун кажети жок. Pythonдо мындай сөз айкаштарын жазуунун кыскартылган ыкмасы кабыл алынат: total -= n и i += 1.

Эскертүү 2. Эсептегичти колдонгондо, ал бирге сөзсүз көбөйтүлүшү шарт эмес, каалаган багытта каалаган мааниге өзгөрүшү мүмкүн. Мисалы, 100дөн 0гө чейин бешке көбөйтүлгөн сандарды көрсөтүү керек болсо, эсептегичтин өзгөрүшү i = i - 5, же i - = 5 болот.

Эскертүү 3. Эсептегич үчүн идентификаторго i өзгөрмөнү колдонуу милдеттүүэмес. Эсептегичтин өзгөрмөсүн каалаган нерсеңиз менен атай аласыз. Бирок программалоодо эсептегичтер i жана j аттары менен белгилөө кабыл алынып калган (кээде бир эле учурда эки эсептегич талап кылынат).

Колдонулган адабият

Пол Бэрри. Изучаем программирование на Python. 2-е издание.

Үй тапшырмасы

While циклин колдонуп, 2 саны үчүн анын кубаттуулугун 0дон 20га чейин көрсөтүп бер. Мисалы төмөнкүдөй жыйынтык чыгуу керек:

...
32
64
128
256
512 1024
...

Тема 13. Каталар жана өзгөчө учурлар 13-сабак (лекция)

Негизги суроолор: Каталар. Өзгөчө учурларды кайра иштетүү. Try-except оператору

Студенттердин мурунку алган билимдери: “Информатика”, “Кесиптик математика”

Окуунун натыйжасы: каталар үстүндө иштей алат. Python программасында Өзгөчө учурларды кайра иштетүүнү билет. Try-except операторун программада колдоно алат.

Тема боюнча маалымат:

Кандайдыр бир, айрыкча чоң программада, анын иштебей калышына же программанын талаптагыдай иштебей калышына алып келген каталар болушу мүмкүн. Катачылыктардын себептери көп.

Программист программалоо тилинин өзүн колдонууда ката кетириши мүмкүн. Башкача айтканда, өзүңүзгө жакпаган нерсени билдирип коюңуз. Мисалы, өзгөрмө аталышты сан менен баштаңыз же татаал сүйлөмдүн башына чекит коюуну унуттуңуз. Мындай каталар синтаксистик каталар деп аталат, алар тилдин синтаксисин жана пунктуациясын бузушат. Python котормочусу, ката сүйлөмгө туш болгондо, аны кантип чечмелөөнү билбейт. Ошондуктан, программанын аткарылышын токтотот жана катанын жайгашкан жерин көрсөткөн ылайыктуу билдирүүнү көрсөтөт: >>> 1a = 10
File "<stdin>", line 1
1a = 10
^

SyntaxError: invalid syntax

Python терминологиясында SyntaxError классына таандык өзгөчө кырдаал түзүлгөн. Python документациясына ылайык, синтаксистик каталар да ката болуп саналат, ал эми калгандарын “өзгөчө” учурларга. Айрым программалоо тилдеринде "өзгөчө" деген сөз колдонулбайт, каталар синтаксистик жана семантикалык болуп экиге бөлүнөт. Семантиканын бузулушу демек, сөз айкаштары тилдин синтаксиси жагынан туура жазылса дагы, программа күткөндөй иштебей калат. Салыштыруу үчүн. Сиз бир нече сүйлөмдү компетенттүү орус тилинде айта аласыз, бирок маниси жагынан керек эмеске айланат, же алар сиз каалагандай түшүнбөйт.

Pythondo семантикалык каталар жөнүндө эмес, четтетүү жөнүндө сүйлөшүшөт. Алар көп. Бул сабакта алардын айрымдарын карап көрөбүз, кийинки сабактарда дагы бир нечесин жолуктурабыз.

Эгерде сиз маани ыйгарылбаган өзгөрмөгө кайрылууга аракет кылсаңыз, анда Python учурда өзгөрмө такыр жарыяланбагандыгын билдирет, ал жок болсо, анда NameError чыгат.

```
>>> a = 0
>>> print(a + b)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'b' is not defined
```

Билдирүүнүн акыркы сабын "Аталыштын катасы: 'b' аты аныкталган жок" деп которсо болот.

Эгерде файлдан кодду аткарууда четтетүү келип чыкса, анда " line 1"дин ордуна ал пайда болгон сап көрсөтүлөт, мисалы, " line 24". "<stdin>" ордуна файлдын аталышы көрсөтүлөт, мисалы, "test.py". Бул учурда, stdin стандарттуу киргизүү агымын билдирет. Демейки боюнча, бул клавиатурадан киргизүү агым. 1-сап - анткени интерактивдүү режимде, ар бир сөз айкашы өз алдынча программа катары өзүнчө интерпретацияланат (чечмеленет). Эгерде сиз бир нече саптардан турган туюнтманы жазсаңыз, анда катанын пайда болуу сабы башка болушу мүмкүн:

```
>>> a = 0 >>>
if a == 0:
...    print(a)
...    print(a + b)
...
0
Traceback (most recent call last):
File "<stdin>", line 3, in <module>
NameError: name 'b' is not defined
```

Эстетип кетчү нерсе мурунку окуу куралдарында кезиккен кийинки эки өзгөчө жагдай - ValueError жана TypeError – маанинин катасы жана типтин катасы.

```
>>> int("Hi")
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'Hi'
>>> 8 + "3"
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Мисалда, "Hi" сабын бүтүн санга айландыруу мүмкүн эмес. ValueError озгөчө учуру пайда болот, анткени int () функциясы мындай маанини өзгөртө албайт.
8 саны жана "3"-катар ар кандай типтерге таандык, алардын ортосунда кошуу операнды колдоого алынбайт. Эгер аларды кошууга аракет кылсаңыз, TypeError өзгөчө учуру чыгат.
Нөлгө бөлүү ZeroDivisionErrorду чыкырат: >>> 1/0
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero

Өзгөчө учурларды кайра иштетүү. Try-except оператору

Программадагы коддорду жазуу же аны текшерүү учурунда катачылыктар пайда болгондо, программист ката кетпеши үчүн кодду түзөтөт. Бирок, колдонуучунун аракеттери көбүнчө программада өзгөчө учурларды алып келет. Мисалы, программа номерди киргизүүнү күтөт, бирок ал адам тамга киргизген. Аны бир санга айландырууга аракет кылуу ValueError бөтөнчөлүгүн жаратат жана программа бузулат.

Бул учурда, программалоо тилдеринде, анын ичинде Pythondo, пайда болгон өзгөчө учурларды кармоого жана программанын иштешин улантууга же аяктоо үчүн аларды иштетүүгө мүмкүнчүлүк берген атайын оператор бар.
Pythondo бул тоскоолдук try-except оператору тарабынан жүзөгө ашырылат. "Try" "аракет кылып көрүү", " except " деп которулган - өзгөчө учур. Сөздөр анын ишин төмөнкүчө сүрөттөй алат: "Муну тигиндей кылууга аракет кыл, эгер өзгөчө кырдаал келип чыкса, анда муну жана тигини кыл". Анын курулушу else филиалы менен шарттуу операторго окшош.
Бир мисалды карап көрөлү: n = input("Бүтүн сан кийир: ") try:
    n = int(n)
print("Ийгиликтүү") except:
    print("Эмнедир туура эмес болуп калды")
Коддун үчүнчү сабында n өзгөрмөсүнүн мааниси бүтүн санга айланганда өзгөчө учур болушу мүмкүн. Эгерде бул мүмкүн эмес болсо, анда аракет денесиндеги сөздөрдүн андан ары аткарылышы токтотулат. Бул учурда, print("Ийгиликтүү") билдирүүсү ишке ашпайт.
Бул учурда, программанын агымы башка бөлүмгө өтүп, анын тулку бою аткарылат.
Эгер try (аракет) денесинде кандайдыр бир өзгөчө кырдаал болбосо, анда башка бутактын тулкусу аткарылбайт.
Колдонуучу бүтүн санды киргизгенде программанын чыгарылышынын мисалы:
Бүтүн сан кийир 100
Ийгиликтүү
А бул жерде – туура эмес киргизилди:
Бүтүн сан кийир: АА туура эмес
киргизилди
Бир көйгөй бар. Жогорудагы код кандайдыр бир өзгөчө кырдаалды жөнгө салат. Бирок, аракет денесинде ар кандай өзгөчө кырдаалдар жаралышы мүмкүн жана алардын ар биринин өзүнчө иштеп чыгуучусу болушу керек. Демек, өзгөчө сөздүн сөзүнөн башка учурларда өзгөчө кырдаалды көрсөтүү туура болот. try:
    n = input('Бүтүн сан кийир: ')
    n = int(n)

```
   print("Баары жакшы. Сиз бүтүн сан кийиргиниз", n) except
ValueError:
   print("Сиз бүтүн сан кийирген жоксуз")
```
Эми денеден башка нерсе иштетилсе, анда эмне ката кетирилгенин жакшы билебиз. Бирок башка бир өзгөчө кырдаал аракет денесинде орун алса, анда ал колдонулбайт. Ал үчүн башка өзүнчө бутак except жазуу керек. Программаны карап көрөлү: try:
```
a = float(input("Бөлүүчүнү кийир : "))
b = float(input("Бөлүнүүчүнү кийир: "))     c = a / b
   print("Жеке: %.2f" % c) except
ValueError:
   print("Катарларды киргизүүгө болбойт") except
ZeroDivisionError:
   print("Нөл санына бөлүүгө болбойт")
```
Ал аткарылганда, коддордун үч сабында өзгөчө учурлар болушу мүмкүн: киргизилген маанилер чыныгы сандарга которулат жана бөлүнүү орун алган жерде. Биринчи учурда, ValueError, экинчисинде, ZeroDivisionError болушу мүмкүн. Өзгөчө кырдаалдын ар бир түрүн филиалдан башка өз алдынча башкарат.
Бир нече өзгөчө учурларды бир бутакка топтоп, чогуу иштетүүгө болот: try:
```
a = float(input("Бөлүүчүнү кийир: "))
b = float(input("Бөлүнүүчүнү кийир: "))     c = a / b
   print("Жыйынтык: %.2f" % c) except
(ValueError, ZeroDivisionError):
   print("Катарларды киргизүүгө же нөлгө бөлүүгө мүмкүн эмес")
```
Башка except учурлардан тышкары, өзгөчө кырдаалды жөндөө оператору, акырында жана башка finally и else бутактарга ээ болушу мүмкүн (сөзсүз түрдө экөө тең бирдениге эмес). Акырында тулку, башка блоктор ыргытылган өзгөчө кырдаалдарга жооп катары аткарылгандыгына карабастан, finally телосу ар дайым аткарылат. Эгер башка учурларда, өзгөчө блоктордон башка өтүүлөр болбогондо, башка дене иштей баштайт. try:
```
   n = input('Введите целое число: ')
n = int(n) except ValueError:
   print("Вы что-то попутали с вводом")
else: # выполняется, когда в блоке try не возникло исключения
print("Все нормально. Вы ввели число", n) finally: #
выполняется в любом случае
   print("Конец программы")
```
Эскертүү. Бул код комментарийлерди колдонот. Pythonдо алардын алдына фунт белгиси # коюлган. Программанын кодундагы комментарийлер адам үчүн гана жазылып, котормочу же компилятор көңүл бурбайт.
Программа өзгөчө шартта жана ансыз кандайча аткарыларын караңыз:
```
pl@pl-desk:~$ python3 test.py
Введите целое число: 4.3
Вы что-то попутали с вводом
Конец программы pl@pl-desk:~$
python3 test.py
Введите целое число: 4
Все нормально. Вы ввели число 4
```

Конец программы

Бул сабакта өзгөчө учурларда иштөөнүн бардык өзгөчөлүктөрү камтылган эмес. Ошентип, бир нече деңгээлдеги коддордун уяларын, функцияларын, модулдарын жана класстарын камтыган ири программаларда өзгөчө учурлар пайда болгон жерде эмес, чалуулардын иерархиясы боюнча өтүшү мүмкүн.

Также исключение может возникнуть в блоке except, else или, и тогда им нужен собственный обработчик. Модифицируем немного предыдущую программу и специально сгенерируем исключение в теле except:

Ошондой эле, except, else же finally блокто да өзгөчө кырдаал келип чыгышы мүмкүн, андан кийин алардын өз иштетүүчү тутуму керек. Мурунку программаны бир аз өзгөртүп, бөтөнчө денеге өзгөчө кырдаал киргизели: try:

```
   n = input('Бүтүн сан кийир: ')
   n = int(n) except
ValueError:
   print("Кийирүү туура эмес аткарылды")
   3 / 0
except ZeroDivisionError:
   print("Ноль санына бөлүү") else:
   print("Баары жайында. Сан кийирилди", n) finally:
   print("Программанын аягы")
```

Башында, баары жайында болуп туюлушу мүмкүн. 3/0 сөз айкашы менен чыгарылган өзгөчө кырдаал except ZeroDivisionError бутагы менен обработкаланат. Бирок,бул андай эмес. Try блогунда пайда болгон бул бутак өзү таандык болгон сыноо блогуна киргизилген өзгөчө учурларды гана жөнгө салат. Эгер бүтүн эмес сан киргизсеңиз, программанын натыйжасы төмөнкүдөй:

```
Бүтүн сан кийир: a
Кийирүүдө адашып калдыныз
Программанын аягы
Traceback (most recent call last):
File "test.py", line 15, in <module> n
= int(n)
 ValueError: invalid literal for int() with base 10: 'a'
 During handling of the above exception, another exception occurred:
 Traceback (most recent call last):
File "test.py", line 18, in <module>
3 / 0
ZeroDivisionError: division by zero
```

Нөлгө бөлүү иштетилбегени аз келгенсип, except телосу ValueError ийгиликсиз бүткөндүгү, өзгөчө ValueError өзү иштетилбейт деп эсептелди. Маселенин чечилиши, мисалы төмөнкүдөй болушу мүмкүн:

```
…
except ValueError:
   print("Вы что-то попутали с вводом")
try:       3 / 0
```

```
    except                    ZeroDivisionError:
print("Деление на ноль")
…
```

Бул жерде, дене бөлүгүнүн өзүнүн ички өзгөчө иштеп чыгуучусу бар.

Бышыктоочу суроолор

Өзгөчө учурларды кайра иштетүүнүн усулдарын айтып бер.
Except операторунун аткарган кызматын аныкта.
Лекцияда мисал келтирилген программаларга анализ жүргүз.

Колдонулган адабият

Пол Бэрри. Изучаем программирование на Python. 2-е издание.

Үй тапшырмасы

Эки маанини кийирүүнү сураган программа. Эгерде алардын бирөө сан болбосо, анда, бириктирүү жүргүзүлүшү керек. Экөө тен сан болгон учурда кошулушу керек болгон программа түзүү. Мисалга:

1-учур
Биринчи маани: 4
Экинчи маани: 5
Жыйынтык: 9.0
2-учур
Биринчи маани: a
Экинчи маани: 9
Жыйынтык: a9

Тема 14. Программалоодогу функциялар 14, 15-сабак (лабораториялык)

Максаты: программада функцияны чакырууну өздөштүрүү
Негизги суроолор: Программалоодогу функцияларды аныктоо. Функцияны аныктоо. Def оператору. Функцияны чакыруу. Функциялардын программага структура берүүсү.
Студенттердин мурунку алган билимдери: “Информатика”, “Кесиптик математика”
Окуунун натыйжасы: функциялар менен иштей алат. Python программасында функцияны аныктоочу Def операторун колдонуп программа түзө алат.
Тема боюнча маалымат:

Программалоодогу функцияларды аныктоо

Программалоодогу функция - коддун өзүнчө бир бөлүгү, аны аталган аты менен атоого болот. Чакырганда, функциянын телосунун буйруктары аткарылат.

Өз алдынча, башкача айтканда, автономдуу түрдө аткарылбай, бирок кадимки программага киргизилген чакан программалар менен функцияларды салыштырууга болот. Көбүнчө аларды подпрограммалар деп аташат. Функциялар менен программалардын ортосундагы башка негизги айырмачылыктар жок. Функциялар, ошондой эле, зарылчылыкка жараша маалыматтарды кабыл алып, кайтарып бере алат. Адатта, алар аларды киргизүүдөн эмес (клавиатура, файл, ж.б.), бирок чакыруучу программадан алышат. Бул жерде алар өз ишинин натыйжасын кайтарып беришет.

Программалоо тилине курулган көптөгөн функциялар бар. Pythondон буга чейин кээ бирлери менен иштедик. Бул print (), input (), int (), float (), str (), type (). Алардын телосунун коду бизге көрүнбөйт, ал кандайдыр бир жерде "тилдин ичинде катылган". Бизге интерфейс гана берилген - функциянын аталышы.

Экинчи жагынан, программист ар дайым өзүнүн функцияларын аныктай алат. Аларды колдонуучулук (пользовательскими) деп аташат. Бул учурда "колдонуучу" программаны колдонгон эмес, программист деп түшүнүлөт. Келгиле, бул функциялар эмне үчүн бизге керек экендигин жана аларды кантип түзүүгө боло тургандыгын аныктайлы.

Кийирүү үчүн бир нече сандарды үч жолу сурап аларды кошуу керек. Бул учурда циклди колдонсо болот:

```
i = 0 while i
< 3:
    a = int(input())
b = int(input())
print(a+b)     i +=
1
```

Бирок, эгер ар бир санды кийирүүдө кандайдыр бир жазуу чыгаруу керек болсо, алардын зарылчылыгы эмнеде жана ар бир жолу бул жазуу ар башка болуп турса. Биз циклди үзүп, андан кийин ошол эле циклге кайра кайта албайбыз. Андан баш тартууга туура келет, ар кайсы жерлерде бир эле бөлүмдү камтыган узун код алабыз: print("Сколько бананов и ананасов для обезьян?") a = int(input()) b = int(input())
print("Всего", a+b, "шт.")

```
print("Сколько жуков и червей для ежей?")
a = int(input()) b = int(input())
print("Всего", a+b, "шт.")

print("Сколько рыб и моллюсков для выдр?")
a = int(input()) b = int(input()) print("Всего",
a+b, "шт.") Программаны аткаруу мисалы:
Сколько бананов и ананасов для обезьян?
15 5
Всего 20 шт.
Сколько жуков и червей для ежей?
50
12
Всего 62 шт.
Сколько рыб и моллюсков для выдр?
16
8
Всего 24 шт.
```

Функцияны кошуп иштөө программанын ар кайсы жерлеринде окшош коддун кайра кайра кайталанбоосуна мүмкүндүк берет. Алардын жардамы менен, ошол эле кодду аткара бербей, керек учурда гана чакырып аткарууга болот.

Функцияны аныктоо. Def оператору

Python программалоо тилинде функциялар def операторунун жардамында аныкталат. Коду карап көрөбүз: def countFood(): a = int(input()) b = int(input()) print("Всего", a+b, "шт.")

Бул функцияны аныктоо мисалы. Башка татаал инструкциялардай эле, мисалы шарттуу жана циклдуу, функция баштапкы темадан жана телодон турат. Башы кош чекит жана жаңы катарга өтүү менен аяктайт. Тело отступка ээ.

Def ачкыч сөзү котормочуга анын алдында аныктама бар экендигин айтат функциялар. Defден кийин функциянын аты жазылат. Бул башкалар сыяктуу эле, ар кандай болушу мүмкүн идентификатор, мисалы, өзгөрмө. Программалоодо бул абдан жагымдуу бардыгына маани-луу ысымдарды беруу. Ошентип, бул учурда функция деп аталат "тамакты саноо" орус тилинде.

Кашалар функциянын аталышынан кийин жайгаштырылат. Көрсөтүлгөн мисалда, алар бош. ал функциясы чакыруучу программадан эч кандай маалыматты кабыл албайт дегенди билдирет.

Бирок, ал аларды кабыл алса болот, андан кийин кашаанын ичине төмөнкүдөй көрсөтүлөт параметрлер деп аталат.

Кош чекиттен кийин аткарыла турган көрсөтмөлөрдү камтыган тулку бойдон турат функцияны чакырганда. Функциянын аныктамасы менен анын чакырылышынын ортосунда айырмачылык болушу керек. IN программа кодунда, алар жакын эмес жана бирге эмес. Функцияны аныктоого болот, бирок экөө тең бир жолу чалба. Аныктала элек функцияны чакыра албайсыз.

Функцияны аныктап, бирок аны эч качан чакырбасаңыз, анын денесин эч качан аткарбайсыз.

Функцияны чакыруу

Функция менен аткарылган программанын толук версиясын карап көрөбүз:

```
def countFood():     a = int(input())     b = int(input())
   print("Всего", a+b, "шт.")

print("Сколько бананов и ананасов для обезьян?") countFood()

print("Сколько жуков и червей для ежей?") countFood()

print("Сколько рыб и моллюсков для выдр?") countFood()
```

После вывода на экран каждого информационного сообщения осуществляется вызов функции, который выглядит просто как упоминание ее имени со скобками. Поскольку в функцию мы ничего не передаем скобки опять же пустые. В приведенном коде функция вызывается три раза.

Когда функция вызывается, поток выполнения программы переходит к ее определению и начинает исполнять ее тело. После того, как тело функции исполнено, поток выполнения возвращается в основной код в то место, где функция вызывалась. Далее исполняется следующее за вызовом выражение.

В языке Python определение функции должно предшествовать ее вызовам. Это связано с тем, что интерпретатор читает код строка за строкой и о том, что находится ниже по

течению, ему еще неизвестно. Поэтому если вызов функции предшествует ее определению, то возникает ошибка (выбрасывается исключение NameError):

```
print("Сколько бананов и ананасов для обезьян?") countFood()

print("Сколько жуков и червей для ежей?") countFood()

print("Сколько рыб и моллюсков для выдр?") countFood()

def countFood():     a =
int(input())     b =
int(input())     print("Всего",
a+b, "шт.")
```

Результат:

```
Сколько бананов и ананасов для обезьян?
Traceback (most recent call last):   File
"test.py", line 2, in <module>
countFood()
NameError: name 'countFood' is not defined
```

Для многих компилируемых языков это не обязательное условие. Там можно определять и вызывать функцию в произвольных местах программы. Однако для удобочитаемости кода программисты даже в этом случае предпочитают соблюдать определенные правила.

Функциялардын программага структура берүүсү

Функциялардын артыкчылыктары бир эле нерсени кайра-кайра чакыруу мүмкүнчүлүгүндө гана эмес. Программанын ар кайсы жерлеринен алынган коддорду чакыруу менен бирдей маанилүү чыныгы түзүлүштү алат. Функциялардын ар бир бөлүктөрү өзүнүн конкреттүү тапшырмасын аткарат.

Ар кандай формадагы аймактарды эсептеген программа жазуу керек деп коёлу. Колдонуучу кайсы фигураны эсептегиси келгенин көрсөтөт. Андан кийин баштапкы маалыматтарды киргизет. Мисалы, тик бурчтуктун узундугу жана туурасы. If-elif-else операторун аткаруу процессин бир нече бутакка бөлүү үчүн, колдонобуз:

```
figure = input("1-прямоугольник, 2-треугольник, 3-круг: ")

if figure == '1':
        a = float(input("Ширина: "))
b = float(input("Высота: "))
print("Площадь: %.2f" % (a*b)) elif
figure == '2':
        a = float(input("Основание: "))
h = float(input("Высота: "))
print("Площадь: %.2f" % (0.5 * a * h))
elif figure == '3':
        r = float(input("Радиус: "))
print("Площадь: %.2f" % (3.14 * r**2)) else:
        print("Ошибка ввода") бул жерде эч кандай функция жок, бирок функция
колдонулган вариантты жазып көрөбүз: def rectangle():     a = float(input("Ширина: "))     b =
float(input("Высота: "))
   print("Площадь: %.2f" % (a*b))
```

```
def triangle():     a =
float(input("Основание: "))     h
= float(input("Высота: "))
   print("Площадь: %.2f" % (0.5 * a * h))

def circle():
   r = float(input("Радиус: "))
   print("Площадь: %.2f" % (3.14 * r**2))

figure = input("1-прямоугольник, 2-треугольник, 3-круг: ") if
figure == '1':
        rectangle() elif
figure == '2':
        triangle() elif
figure == '3':
        circle() else:
        print("Ошибка ввода")
```

Бул кыйла татаал окшойт, жана үч функциянын ар бири бир гана жолу аталат.

Бирок, программанын жалпы логикасынан, көрсөтмөлөрү бул аймактарды табуу. Эми программа өзүнчө "курулуш блокторунан" турат Lego. "Негизги бутакта биз аларды каалагандай бириктире алабыз. Башкаруу механизми ролду ойнойт. Бышыктоочу суроолор

Подпрограмма деген эмне?

Функцияны аныктоодо кайсы оператор колдонулат.

Def операторунун программада колдонулушун айтып бер

Колдонулган адабият

Пол Бэрри. Изучаем программирование на Python. 2-е издание.

Үй тапшырмасы Үч бурчтуктун жана беш бурчтуктун периметрин табуунун программасын түзүү.

Тема 16. Функциядан маанилерди кайтаруу. Return оператору 16-сабак (лабораториялык)

Максаты: программада функцияларды колдоно билүү, Return операторун колдонуп программа түзүү

Негизги суроолор: Функциядан маанилерди кайтаруу. Бир нече маанилерди кайтаруу. Камтылган функциялар

Студенттердин мурунку алган билимдери: “Информатика”, “Кесиптик математика”

Окуунун натыйжасы: камтылган функциялар менен иштей алат. Python программасында маанилерди колдонуп функцияны кайтарууну билет.

Тема боюнча маалымат:

Функциядан маанилерди кайтаруу

Функциялар өзүлөрүнүн денесинен каалаган маалыматтарды негизги бутакка өткөрүп бере алышат программалар. Функция маанини кайтарат деп айтылат. Көпчүлүк

тилдерде программалоо, анын ичинде Python, функциясынан чыгуу жана ага маалыматтарды берүү ал чакырылган жер кайтаруу билдирүүсү менен аткарылат.
Эгерде функциянын тулку бөлүгүн аткарган Python котормочу return жооп берсе, анда ал ушул буйруктан кийин көрсөтүлгөн маанини "алат" жана функцияны "калтырат".def cylinder():
r = float(input())
h = float(input())
# площадь боковой поверхности цилиндра:
side = 2 * 3.14 * r * h
# площадь одного основания цилиндра:
circle = 3.14 * r**2
# полная площадь цилиндра:
full = side + 2 * circle return
full

square = cylinder()
print(square) аткаруу
мисалы :
3
7
188.4
Бул программада функция функциядан маниси негизги бутакка кайтарылат жергиликтүү өзгөрүлмө толук . Бул учурда өзгөрмөнүн өзү эмес, анын мааниси - цилиндрдин аянтын эсептөөнүн натыйжасында алынган каалаган сан.
Программанын негизги тармагында бул маани глобалдык мааниге ээ өзгөрүлмө чарчы . Башкача айтканда, квадрат = цилиндр () туюнтмасы мындайча аткарылат:
1. Цилиндр () функциясы деп аталат .
2. Андан бир маани кайтарылат.
3. Бул маани өзгөрүлмө квадратка ыйгарылат .
Жыйынтыкты өзгөрмөгө ыйгаруунун кажети жок, ал түздөн-түз чыгарылышы мүмкүн экран:
...
басып чыгаруу (цилиндр ())
Бул жерде цилиндрден алынган сан түздөн-түз функцияга берилет print ().
Алынган нерсени ыйгарбастан программада цилиндр () деп жазсак өзгөрүлмө маалыматтар же андан ары эч жакка өткөрбөсөңүз, анда бул маалыматтар калат жоголгон. Бирок синтаксистик ката болбойт.
Функцияда бир нече кайтаруу билдирүүлөрү болушу мүмкүн. Бирок, ал ар дайым кармалып турат
алардын бири гана. Аткаруу жипи биринчи жетет. Келиңиз жараксыз киргизүүгө ташталган өзгөчө кырдаалды жөндөөнү чечтик. Болсун анда өзгөчө иштеп чыгуучунун башка бутагында функциясы жок чыгат ар кандай эсептөөлөр жана өткөрүп берүү мааниси:
def cylinder():
try:

```
        r = float(input())
h = float(input())
except ValueError:
return
    side = 2 * 3.14 * r * h
circle = 3.14 * r**2      full
= side + 2 * circle
return full

print(cylinder())
```

Эгер сиз сандардын ордуна тамгаларды киргизүүгө аракет кылсаңыз, анда кайтып келүү уясына жайгаштырылган башка. Ал төмөнкүлөрдүн бардыгы иштеши үчүн, функциянын аткарылышын токтотот эсептөөлөр, анын ичинде толугу менен кайтарылып берилбейт. Аткаруу мисалы: r

Жок

Бирок күт! Бул эмне деген сөз Эч ким "бош" кайтып бизге кайтып келген? ал эч нерсе, мындай объект эч нерсе эмес. Бул NoneType классына таандык. Ага чейин биз маалыматтардын төрт түрүн билген, алар дагы төрт класс: int, float, str, bool. Убакыт келди Кайра кайтып келгенден кийин эч нерсе көрсөтүлбөсө, демейки шартта ошол жерде болот деп эсептелет объект жок. Бирок эч ким так жооп кайтаруу деп жазуу менен сизди убара кылбайт. Анын үстүнө. Буга чейин, биз артка кайтпаган функцияларды карап чыктык эч кандай маани жок, анткени аларда кайтаруу билдирүүсү болгон эмес. Чындыгында кайтып келди, биз ага маани берген жокпуз, эч бирин туура көргөн жокпуз өзгөрүлмө жана көрсөтүлгөн эмес. Pythondo ар бир иштин мааниси бар кайтып келет. Эгер анда эч кандай кайтаруу билдирүүсү жок болсо, анда ал Жокту кайтарат. Ошондой эле анын "бош" кайтарымы бар сыяктуу.

Бир нече маанилерди кайтаруу

Pythondo, функциялардан бир нече объектилерди тизмеге кайтарып берүүгө уруксат берилет кайтаруу командасынан кийин аларды үтүр менен бөлүп: def cylinder():

```
    r = float(input())     h
= float(input())     side =
2 * 3.14 * r * h     circle
= 3.14 * r**2     full =
side + 2 * circle
    return side, full

sCyl, fCyl = cylinder() print("Площадь боковой
поверхности %.2f" % sCyl) print("Полная площадь
%.2f" % fCyl)
```

Цилиндр () функциясынан эки маани кайтарылат. Биринчиси дайындалат sCyl өзгөрүлмө , экинчиси - fCyl . Мындай топтук дайындоонун мүмкүнчүлүгү - адатта, башка тилдерде кездешпеген Python функциясы:

```
>>> a, b, c = 10, 15, 19
>>> a
10
>>> b
15
>>> c
```

19

Бул жерде амалкөйлүк мааниси - тизме мааниси үтүр менен ажыратылган (мисалы, 10, 15, 19) кортеж типиндеги объектти түзөт. Ал орус тилине "кортеж" деп которулат. ал кийинчерээк изилдене турган маалыматтардын структурасынын бир түрү. Бир жолу бир нече өзгөрмөлөргө кортеж дайындалганда, ал ишке ашат анын элементтерин кезектеги тиешелүү өзгөрмөлөргө дал келтирүү. ал таңгактан чыгаруу деп аталат.

Ошентип, функциядан бир нече маани кайтарылса, чындыгында

Чындыгында, андан класстык кортеждин бир объектиси кайтарылат. Буларды кайтаруудан мурун бир нече маанилер кортежге салынган. Эгерде оператор кайтып келгенден кийин эгер бир гана өзгөрмө же объект болсо, анда анын түрү ошол бойдон сакталат. Таңгактан чыгаруу милдеттүү эмес. Ал мындай иштейт:… print(cylinder()) аткаруу мисалы:

4

3

(75.36, 175.84)

Экранга кашаа менен көрсөтүлгөндөй кортеж басылып чыгарылат. Ошондой эле болушу мүмкүн бир өзгөрмө ыйгарып, андан кийин анын маанисин экранда көрсөтүү. Камтылган функциялар

Python тили буга чейин аныкталган, башкача айтканда, ага орнотулган көптөгөн функцияларды камтыйт.

Программист алардын аныктамаларын көрбөйт, алар тилдин "ичегисинде" бир жерде жашырылган.

Бул функциялар эмнени кабыл алып, эмнени кайтарарын, башкача айтканда, алардын эмне экендигин билүү жетиштүү интерфейс.

I / O жана маалымат түрлөрүнө байланыштуу бир катар камтылган функциялар, буга чейин эле бар

колдонулган. Бул print (), input (), int (), float (), str (), bool (), type (). Баарынын тизмеси Python орнотулган функцияларын расмий документтерден таба аласыз тили: https://docs.python.org/3/library/functions.html . Орус тилине которсоңуз болот бул жерге караңыз: https://pythoner.name/documentation/library/functions .

Бул окуу куралы, биз төмөнкү орнотулган функцияларды карап, аларды шарттуу түрдө бөлүштүрөбүз топтор:

белгилер менен иштөө функциялары - ord (), chr (), len () математикалык функциялар - abs (), round (), divmod (), pow (), max (), min (), sum () Ord () функциясы белгилердин номерин Юникод таблицасынан алууга мүмкүнчүлүк берет. Демек, бир белгини аргумент катары кабыл алат, цитата келтирилген:

```
>>> ord ( 'z' )
122
>>> ord ( 'f' )
1092
>>> ord ( '@' )
64
```

Chr () функциясы тескерисинче иштейт. Бул белгини алууга мүмкүндүк берет анын номери: >>> Оруссия (87) 'W'

```
>>> Оруссия ( 1049 )
"Ү"
>>> chr ( 10045 )
'❇'
```

Ord () жана chr () менен чаташпаш үчүн, функция иш-аракет экендигин унутпаңыз. Анын аты-жөнү
"Эмне кылуу керек?" деген суроого жооп берет. Тартип - бул буйрук. Ошентип, биз каалайбыз катардагы элементтин иреттик номерин алуу. Жана номерин алуу үчүн, сөзсүз керек өткөрүп берүү белгиси. Мүнөз - мүнөз. Ошентип, биз бир белгини алгыбыз келет. Демек, ырааттуулук номери берилиши керек.
Len () функциясы аргумент катары көбүрөөк нерседен турган объектти алат жөнөкөй объектилер, ал алардын санын эсептейт. Сандар жөнөкөй объекттер, аларды len () ге өткөрүү мүмкүн эмес. Саптар болушу мүмкүн: >>> len ('abc')
3
>>> s1 = '------'
>>> s2 = '_____'
>>> len (s1) > len (s2)
Жалган
>>> len (s1)
6
>>> len (s2) 7
Len () тилкелеринен тышкары, биз дагы эле изилдей элек башка структураларды дагы өткөрө аласыз маалыматтар. Abs () функциясы сандын абсолюттук маанисин кайтарып берет: >>> абс (- 2.2) 2.2
>>> абс (9) тогуз
Эгер сиз чыныгы санды белгилүү бир белгиге чейин тегеректөөнү кааласаңыз үтүр, анда round () функциясын колдонуу керек:
>>> бир = 10 / 3
>>> а
3.3333333333333335
>>> тегерек (а , 2)
3.33
>>> тегерек (а)
3
Эгерде экинчи аргумент көрсөтүлбөсө, анда тегеректөө бүтүн санга өтөт. Бирөө бар бул иштин өзгөчөлүгү. Экинчи аргумент болушу мүмкүн терс сан. Бул учурда, бирдиктер, ондогон, жүздөгөн ж.б., башкача айтканда, бүт бөлүгү:

>>> тур (5321 , - 1)
5320
>>> тур (5321 , - 3)
5000
>>> тур (5321 , - 4)
10000
Функция математикадан алынган тегеректөө эрежесине ылайык, так эмес тегеректелет таштайт. Демек, 5 миң күтүлбөгөн жерден онго тегеректелет.
>>> тур (3.76 , 1)
3.8

>>> тур (3.72 , 1)
3.7
>>> тур (3.72) төрт
>>> тур (3.22)
3
Эгер сиз жөн гана бөлчөк бөлүктөн тегеректелбестен кутулуу керек болсо, анда керек int () функциясын колдонуңуз: >>> int (3.78) 3
Көбүнчө, тегерек () функциясы print () функциясы менен бирге колдонулат, качуу натыйжаны форматтоо: >>> a = 3.45673
>>> басып чыгаруу ("Номери:% .2f" % a) Номери:
3.46
>>> басып чыгаруу ("Номери:" , тегерек (a , 2))
Номери: 3.46
Акыркы учурда, код такыраак көрүнөт.
Divmod () бүтүндөй бөлүүнү жана калууну бир эле мезгилде аткарат бөлүнүүдөн:
>>> divmod (10 , 3)
(3, 1)
>>> divmod (20 , 7)
(2, 6)
Бул маалыматтарды казып алууну изилдей элек кортежди кайтарат. IN
Башка тилдерде эки бөлөк функцияны көрүү сейрек эмес: div () жана mod (). Биринчи интегралдык бөлүнөт, экинчиси бүтүн бөлүндүн калдыгын табат (менен бөлүштүрүү модуль). Бул үчүн Python жана башка көптөгөн тилдерде атайын колдонулат.
операнд белгилери: >>>
10 // 3
3
>>> 10 % 3 бир
Pow () функциясы кубаттуулукка көтөрүлөт. Биринчи сан база, экинчиси көрсөткүч:

>>> pow (3 , 2) тогуз
>>> pow (2 , 4) он
алты
Ушундай кылса болот:
>>> 3 ** 2 тогуз
>>> 2 ** 4 он
алты
Бирок, pow () кошумча үчүнчү аргументти кабыл алат. Бул номер, күнү көрсөткүчтүн жыйынтыгы боюнча модулго бөлүнөт:
>>> pow (2 , 4 , 4)
0
>>> 2 ** 4 % 4 0
Биринчи ыкманын артыкчылыгы - анын тезирээк аткарылышы.
Max (), min () жана sum () функциялары тиешелүүлүгүнө жараша максимумду табышат, аргументтин минималдуу элементи жана элементтеринин суммасы:
>>> макс (10 , 12 , 3)
12

>>> мүн (10 , 12 , 3 , 9)
3
>>> a = (10 , 12 , 3 , 10)
>>> сумма (a) 35
Суммага элементтердин тизмесин бере албайсыз (), маалыматтардын структурасы болушу керек,
мисалы кортеж. Ошондой эле Min () жана max () деп аталган биринен өтүү мүмкүнчүлүгү жогору кайталануучу объект:
>>> макс (a)
12

Колдонулган адабият
Пол Бэрри. Изучаем программирование на Python. 2-е издание.

Үй тапшырмасы
Киргизилген саптын узундугун өлчөгөн программа жазыңыз. Эгер он белгиден узун сап болсо, анда эскертүү берилет.

Тема 17. Модулдар 17-сабак (лекция)

Негизги суроолор: Модулдардын түрлөрү. Модулдарды импорттоо. Python программасында өз модулунду түзүү.
Студенттердин мурунку алган билимдери: “Информатика”, “Кесиптик математика”
Окуунун натыйжасы: Модулдардын түрлөрүн жана кызматтарын билишет. Модулдарды импорттой алат. Python программасында өз модулун түзөт.
Тема боюнча маалымат:
Модулдардын түрлөрү
Программалоо тилине орнотулган функциялар дароо жеткиликтүү. Аларды чакыруу үчүн, кошумча аракеттерди жасоонун кажети жок. Бирок, ар кандай популярдуу тил болгон мезгилде, ага көптөгөн программисттер жана ар кандай чөйрөлөрдө талап кылынган көптөгөн функциялар жана класстар жазылгандыктан, мүмкүн болушунча тилдин өзүнө ушул көлөмдөгү кодду киргизүү максатка ылайыктуу эмес.
Тилдин кошумча мүмкүнчүлүктөрүнө жетүүнү чечүү үчүн, модулдарды, пакеттерди жана китепканаларды колдонууну программалоодо кеңири жайылган практика болуп калды. Ар бир модулда белгилүү бир чөйрөдөн чыккан маселелерди чечүүгө арналган функциялардын жана класстардын жыйнагы камтылган. Мисалы, Python догу math модулунда математикалык функциялар камтылат, random псевдо-кокустук сандарды жаратууга мүмкүндүк берет, datetime модулунда даталар жана убакыттар менен иштөө класстары камтылган, sys модулу тутумдун өзгөрмөлөрүнө кирүүгө мүмкүнчүлүк берет ж.б.
Python тили үчүн модулдардын санынын абдан көптүгү, тилдин популярдуулугуна байланыштуу. Модулдардын бир бөлүгү стандарттык китепкана деп аталган жерде топтолгон. Стандарттуу анткени орнотуу пакети менен бирге келгендиги үчүн . Бирок, үчүнчү тараптын китепканалары бар. Алар жүктөлүп, өзүнчө орнотулат.
Модулдарды импорттоо

Модулдун иштешине мүмкүнчүлүк алуу үчүн, аны программага импорттоо керек. Импорттоодон кийин котормочу кошумча класстардын жана функциялардын бар экендигин "билет" жана аларды колдонууга мүмкүндүк берет.

Pythonдо импорттоо import буйругу менен жүргүзүлөт. Ошентсе да, импорттоонун бир нече жолдору бар. Math мисалын колдонуп модуль менен иштөөнү карап көрөлү. Ошентип,

```
>>> import math
```

Ничего не произошло. Однако в глобальной области видимости появилось имя. Если до импорта вы упомянули бы имя, то возникла бы ошибка NameError. Теперь же

Эч нерсе болгон жок. Бирок, math аты глобалдуу чөйрө көрүнүшүндө пайда болду. Эгер импорттоодон мурун math атын эскертип койсо, анда ката NameError болмок. Ал эми азыр

```
>>> math
<module 'math' (built-in)>
```

Программада module классына таандык math объекти пайда болду.

Бул модулга кирген функциялардын тизмесин көрүү үчүн, модулдун атын аргумент катары өткөрүп, орнотулган Python dir () функциясын колдонуңуз:

```
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh',
'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh',
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor',
'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite',
'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf',
'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

Кош сызыкчалар менен асты сызылган аттарды карабай эле коёлу. Калгандары математика модулуна кирген функциялардын жана туруктуу адамдардын аталыштары (маанилерин өзгөртпөстөр). Функцияны модулдан чакыруу үчүн, алдыңа модулдун атын жазып, чекит коюп, андан кийин функциянын атын көрсөтүп, андан кийин аргументтерди керек болсо, кашаанын ичине камтышы керек. Мисалы, математикадан электрондук почтаны чакыруу үчүн сиз мындай деп жазасыз:

```
>>> math.pow(2, 2)
4.0
```

Белгилей кетүүчү нерсе, pow () бул башка функция, бул тилдин өзүнө орнотулган эмес. "Нормалдуу" pow () функциясы бүтүн санды кайтарат, эгерде аргументтер бүтүн сандар болсо: >>> pow(2, 2) 4

Константага кайрылуу үчүн кашаанын кереги жок:

```
>>> math.pi
3.141592653589793
```

Эгерде кандайдыр бир функция эмне кылаарын билбесек, анда Python тилине киргизилген функцияларды колдонуп, ал жөнүндө маалыматты алабыз: help():

```
>>> help(math.gcd)
Help on built-in function gcd in module math:
gcd(...)     gcd(x, y) -> int
```

greatest common divisor of x and y интерактив жардамынан чыгуу үчүн q баскычын басыңыз. Бул учурда, функция бүтүн санды кайтарат деп маалымат берилет жана бул x жана y сандары үчүн эң чоң жалпы бөлүүчү. Модулдардын сүрөттөлүшүн жана алардын мазмунун python.org расмий документтацияларынан көрүүгө болот.

Экинчи импорттоо ыкмасы - модулдун өзү импорттолбостон, анын ичиен керектүү функциялар гана импорттолот.

```
>>> from math import gcd, sqrt, hypot
```

Төмөнкүдөй которууга да болот "math модулунан, gcd, sqrt жана hypot функцияларын импорттоо".

Бул учурда аларды чакырууда функциянын атынын алдына модулдун атын көрсөтүүнүн кажети жок:

```
>>> gcd(100, 150)
50
>>> sqrt(16)
4.0
>>> hypot(3, 4)
5.0
```

Модулдан баардык функцияларды бирденине импорттоо үчүн:

```
>>> from math import *
```

From аркылуу импорттоо кемчиликсиз эмес. Программа буга чейин импорттолгон функциялардын же туруктуу адамдардын биринин аталышы менен бирдей аталыштагы идентификаторго ээ болушу мүмкүн. Ката болбойт, бирок алардын бири "өчүрүлөт":

```
>>> pi = 3.14
>>> from math import pi
>>> pi
3.141592653589793
```

Бул жерде pi өзгөрүлмөсүнө ыйгарылган 3.14 маани жоголот. Бул ат эми math модулундагы санды көрсөтөт. Эгер импорттоону pi ге маани бергенге чейин жүргүзсө, анда баары тескерисинче болот:

```
>>> from math import pi
>>> pi = 3.14
>>> pi
3.14
```

Ушуга байланыштуу, бардык функциялардын бирденинен импорту кооптуу. Анткени, мындай учурда идентификаторлордун маанилеринин алмаштырылышын байкабай калуу оңой.

Ошентсе да, идентификатордун атын модулдан каалагандай өзгөртө аласыз:

```
>>> from math import pi as P
>>> P
3.141592653589793
>>> pi
3.14
```

Бул учурда, модулдан туруктуу pi P аталышы менен импорттолот, мындай импорттун мааниси аталыштарды кыскартуу болот саналат, анткени узун аталыштагы модулдар да бар, алардагы функциялардын жана класстардын аттары дагы да узунураак. Эгерде программага бир-эки гана субъект импорттолсо жана анда алар тез-тез колдонулса, анда аларды кыскараак нускага өзгөртүү туура болот. Салыштыр:

```
>>> import calendar
>>> calendar.weekheader(2)
'Mo Tu We Th Fr Sa Su' жана
>>> from calendar import weekheader as week
>>> week(3)
'Mon Tue Wed Thu Fri Sat Sun'
```

Башка учурларда, модулдун мазмунунун идентификаторлорун модулдун өзүнүн аталыштар мейкиндигинде калтырып, аларга модулдун аталышы аркылуу жетүү, башкача айтканда, import имя_модуля, командасынын жардамы менен импорттоо жана мисалы, имя_модуля.имя_функции(). аркылуу функцияларды чакыруу жакшы.

Python программасында өз модулун түзүү

Программист Python программасында, өзүнүн программаларында колдонуу үчүн ар дайым өзүнүн модулун түзө алат, ал тургай колдонууга бүткүл дүйнөгө жеткиликтүү кыла алат. Машыгуу катары, тик бурчтуктун, үч бурчтуктун жана айлананын аянттарын эсептөө үчүн функциялары менен модуль түзөлү: from math import pi, pow

```
def rectangle(a, b):
    return round(a * b, 2)

def  triangle(a,  h):        return
round(0.5 * a * h, 2)

def circle(r):     return round(pi *
pow(r, 2), 2)
```

Ошондой эле бир модуль башка модулдарды импорттой алат деген принципти чагылдырат. Азыркы учурда функциялар math модулунан импорттолот.

Бул кодду өзүнчө square.py файлына жайгаштырыңыз. Бирок, файлдын өзүн кайда коёбуз? Python интерпретатору импорт командасына туш болгондо, ал белгилүү каталогдордогу модулдун файлын болушун издейт. Алардын тизмесин sys.path мазмунунан көрүүгө болот: >>> import sys >>> sys.path

```
['', '/usr/lib/python35.zip', '/usr/lib/python3.5', '/usr/lib/python3.5/plat-x86_64-linux-gnu',
'/usr/lib/python3.5/lib-dynload', '/home/pl/.local/lib/python3.5/site-packages',
'/usr/local/lib/python3.5/dist-packages', '/usr/lib/python3/dist-packages']
```

Бул Linuxдагы даректердин тизмеси. Windowsто бир аз башкача болот. Биринчи элемент - бош сап, бул учурдагы каталог б.а. модулду импорттогон программанын өзү сакталган жерди маанисин билдирген. Эгерде сиз модуль файлын жана программа файлын бир каталогго сактасаңыз, котормочу модулду оңой эле таба алат. Ошондой эле, модуль тизмеде көрсөтүлгөн каалаган каталогго жайгаштырылышы мүмкүн. Анда ал бардык Python программаларына жеткиликтүү болуп, интерактивдүү режимде импорттоого болот. Sys.path ка өзүбүдүн каталогду кошсо болот. Бирок, бул учурда, программанын коду sys.path маанисин өзгөртүү боюнча буйруктарды камтышы керек, же операциондук системдин конфигурациялык файлы түзөтүлүшү керек. Көпчүлүк учурда муну жасабаган жакшы.

Square.py файлын аткарылуучу программа менен бир каталогго жайгаштырыңыз. Анын кодунда square модулун импорттоо инструкциясы камтылышы керек (импорттоо учурунда файл кеңейтүүсү көрсөтүлбөйт) жана колдонуучу киргизген параметрлер менен ушул эле функцияны чакыруу. Башкача айтканда, колдонуучудан кайсы фигуранын аянтын

эсептегиси келерин сураш керек. Андан кийин, андан тиешелүү функция үчүн аргументтер сураңыз. Аларды square модулунан функцияга өткөрүп берип, ал жактан алынган натыйжаны экранга чыгаруу керек.

Эскертүү. Өз алдынча скрипт катары модулду аткаруу, ошондой эле Pythondo орнотулган help () функциясы тарабынан көрсөтүлүүчү документациялардын катарын түзүү, объектке багытталган программалоо курсунда камтылат.

Бышыктоочу суроолор жана тапшырмалар
Модулдардын импорттоо жолдорун айтып бер
Өз модулунду түзүү эрежелери
Модулдардын түрлөрүн айтып бер

Колдонулган адабият
Пол Бэрри. Изучаем программирование на Python. 2-е издание.

Үй тапшырмасы: төрт бурчтуктун, үч бурчтуктун жана эллипстин аянттарын эсептөө үчүн функциялары менен модуль түз.

Тема 18. Саптар 18-сабак (лекция)

Негизги суроолор: Сап түшүнүгү. Катар ыкмалары Split() жана join() методдору.
Студенттердин мурунку алган билимдери: “Информатика”, “Кесиптик математика”
Окуунун натыйжасы: Сап түшүнүгүн билет. Катар ыкмалары Split() жана join() програмада колдоно билет.
Тема боюнча маалымат:
Сап түшүнүгү
Бүтүн жана бөлчөк сандар менен катар биз буга чейин саптарды жөнөкөй маалыматтардын түрү катары караганбыз, катар бул- бирдик же экилик тырмакчага алынган символдордун катары.
Pythondo символдук тип жок, башкача айтканда, объект болуп жалгыз символдор саналат. Бирок программалык тил катарларды объект катары мүмкүндүк берет. Узундугу бир же андан көбүрөөк символдордон турган катар асты. Ошол эле учурда, тизмелерден айырмаланып, катарлар маалымат структурасы катары классификацияланбайт. Маалымат структуралары бир кыйла жөнөкөй маалыматтардын типтеринен турган үчүнда болсо керек. Ал эми Pythondогу катарлар үчүн мындан өтөөр жөнөкөйрөөк тип жок.
Экинчи жагынан, катар тизмеде эле - бул элементтердин ирээттелген ырааттуулугу. Демек, андан айрым белгилерди жана тилмелерди бөлүп алууга болот.

```
>>> s = "Hello, World!"
>>> s[0]
'H'
>>> s[7:]
'World!'
>>> s[::2]
'Hlo ol!'
```

Акыркы учурда, чыгаруу экиге барабар кадам менен жүргүзүлөт, башкача айтканда, ар бир экинчи белги чыгарылат.

Эскертүү. Ошондой эле, бөлүктөрдү тизмелерден кадам менен бөлүп алууга болот. Тизмелерден маанилүү айырмачылык - Pythonдогу саптардын өзгөрүлбөгөндүгү. Сиз катардан кандайдыр бир символду же тилкени кайра жазууга болбойт:

```
>>> s[-1] = '.'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Str тибиндеги объект элементтерге ыйгарууну колдобоорун интерпретатор кабар берет. Эгер катарды өзгөртүү талап кылынса, анда эски кесиндилерден жаңысын түзүү керек:

```
>>> s = s[0:-1] + '.'
>>> s
'Hello, World.'
```

Мисалда баштапкы саптан кесинди алат, башка сап менен бириктирилет. S өзгөрмөсүнө ыйгарылган жаны катар пайда болот. Мындай учурда анын эски мааниси жоголот. Катар ыкмалары

Pythonдо катарлар үчүн көптөгөн ыкмалар бар. Аларды dir (str) буйругу боюнча көрө аласыз, ар бири жөнүндө маалымат аласыз - help(str.имя_метода). Алардын арасынан эң кызыктуусун карап көрөлү.

Split() жана join() методдору

Split () методу катарларды пробел боюнча бөлүүгө мүмкүндүк берет. Натыйжада сөздөрдүн тизмеси пайда болот. Эгерде колдонуучу бир сапка сөздөрдүн же сандардын катарларын киргизсе, алардын ар бири программада өзүнчө иштетилиши керек болсо, анда split () сөзсүз колдонуу керек.

```
>>> s = input() red
blue orange white
>>> s
'red blue orange white'
>>> sl = s.split()
>>> sl
['red', 'blue', 'orange', 'white']
>>> s
'red blue orange white'
```

Список, возвращенный методом split(), мы могли бы присвоить той же переменной s, т. е. s = s.split(). Тогда исходная строка была бы потеряна. Если она не нужна, то лучше не вводить дополнительную переменную.

Метод split() может принимать необязательный аргумент-строку, указывающей по какому символу или подстроке следует выполнить разделение:

Split () ыкмасы менен кайтарылган тизмени ошол эле s өзгөрмөсүнө, башкача айтканда, s = s.split () дайындай алабыз. Ошондо баштапкы сап жоголуп кетмек. Эгер ал кереги жок болсо, анда кошумча өзгөрмө киргизбөө жакшы.

Split () ыкмасы кайсы символго же подстронго бөлүнүү керектигин көрсөткөн милдеттүү эмес аргументти кабыл алат:

```
>>> s.split('e')
['r', 'd blu', ' orang', ' whit', '']
>>> '40030023'.split('00')
['4', '3', '23']
```

Join () string методу тескерисинче кылат. Ал тизмеден сап түзөт. Бул сап ыкмасы болгондуктан, бөлгүч сабы алдына жайгаштырылып, тизме кашаага жазылат: >>> '-'. кошулуу (sl)
'кызыл-көк-кызгылт сары-ак'
Эгерде эч кандай сепаратордун кереги жок болсо, анда метод бош сапка колдонулат:
Бул сап методдору субстрингдер менен иштешет. Find () ыкмалары саптагы субстрингди издеп таап, табылган подстринанын биринчи элементинин индексин кайтарат. Эгерде субстринг табылбаса, анда -1 кайтып келет. >>> s
'red blue orange white'
>>> s.find('blue')
4
>>> s.find('green')
-1
Издөөнү бүтүндөй сапта эмес, анын айрым сегменттеринде гана жүргүзсө болот. Бул учурда сегменттин биринчи жана акыркы индекстери көрсөтүлөт. Эгерде экинчиси көрсөтүлбөсө, анда саптын аягына чейин издейт:
>>> letters = 'ABCDACFDA'
>>> letters.find('A', 3)
4
>>> letters.find('DA', 0, 6) 3
Бул жерде үчүнчү индекстен аягына чейин, ошондой эле биринчиден алтынчыга чейин издейбиз. Find () ыкмасы биринчи көрүнүштү гана кайтарып берерин эске алыңыз. Ошентип, letter.find ('A', 3) сөз айкашы акыркы 'A' тамгасын таппайт, анткени 'A' буга чейин 4 индекси менен жолугушкан. Replace () методу бир подстринканы экинчисине алмаштырат:
>>> letters.replace('DA', 'NET')
'ABCNETCFNET'
Исходная строка, конечно, не меняется:
>>> letters
'ABCDACFDA'
Так что если результат надо сохранить, то его надо присвоить переменной:
>>> new_letters = letters.replace('DA', 'NET')
>>> new_letters
'ABCNETCFNET'
Format() методу
Format() сап методу буга чейин print() функциясын өздөштүрүүдө айтылып өткөн:
>>> print("This is a {0}. It's {1}.".format("ball", "red")) This is a ball. It's red.
Print () функциясын колдонуп экранга чыгууну карап жатканда format () сап ыкмасы мурунтан эле айтылган:
>>> басып чыгаруу ("Бул {0}. Бул {1}.". формат ("топ", "кызыл"))
Бул топ. Бул кызыл.
Бирок, ал print () менен эч кандай байланышы жок, бирок саптарга тиешелүү. Ошондо гана жаңы түзүлгөн сап чыгаруу функцияларына өткөрүлүп берилет.
Format () кеңири мүмкүнчүлүктөргө ээ, алардын негизгисин карап көрөлү.
>>> size1 = "length - {}, width - {}, height - {}"
>>> size1.format(3, 6, 2.3)

'length - 3, width - 6, height — 2.3'
Эгерде баштапкы саптын тармал кашалары бош болсо, анда аргументтердин орун алмашуусу алардын ырааттуулугу боюнча болот. Эгерде аргумент индекстери саптагы ийри кашаанын ичинде көрсөтүлсө, анда орун алмаштыруу тартиби өзгөрүлүшү мүмкүн:
>>> size2 = "height - {2}, length - {0}, width - {1}"
>>> size2.format(3, 6, 2.3)
'height - 2.3, length - 3, width - 6'
Мындан тышкары аргументтер сөз-ачкыч боюнча да берилиши мүмкүн:
>>> info = "This is a {subj}. It's {prop}."
>>> info.format(subj="table", prop="small") "This
is a table. It's small."
Бөлчөк сандарды форматтоо мисалы:
>>> "{1:.2f} {0:.3f}".format(3.33333, 10/6)
'1.67 3.333'

Бышыктоочу суроолор
Сап түшүнүгүн айтып бер
Катар ыкмалары Split() жана join() методдорун программада колдонулушун көрсөт

Колдонулган адабияттар
Гуриков, С.Р. Основы алгоритмизации и программирования на Python:
Основы алгоритмизации и программирования: лабораторный практикум:
Изучаем программирование на Python. 2-е издание. Пол Бэрри

Үй тапшырмасы
Split() жана join() методдоруна мисал келтирип программа түз.

1-тиркеме Экзамендик суроолордун тизмеси

1. Формалдаштыруу түшүнүгү
2. Сыпаттамалык моделди математикалык түргө өткөрүү
3. ЭЭМде берилген тапшырманы аткаруудагы 4 этап
4. Тексттик алгоритмди жазуу
5. Графикалык-блок схемасын түзүү
6. Программа түрүндө жазуу (иштөө)
7. Программа. Программалоо тили
8. Программалоо тилдеринин тарыхый өнүгүшүнүн негизги этаптары
9. Программалоо тилдеринин түрлөрү
10. Трансляциялоо. Которуу
11. Программалоодогу операцияларды атоо
12. Маалымат түрлөрүн өзгөртүүнү өздөштүрүү
13. Өзгөрүлмөлөр менен иштөө
14. Жыйынтыкты чыгаруучу print() функциясы
15. Өзгөрүлмөлөрдү кийирүүчү input() функциясы
16. Integer, fload, string типтеринин программада колдонулушу

17. Логикалык туюнтмаларды программада колдонуу, блок схемаларды түзүү 18. Татаал логикалык туюнтмаларга мисалдар
19. Өзгөчө учурларды кайра иштетүү.
20. Try-except операторун питон программасында колдонуу
21. Шарттуу оператор. Бутактануу программасынын структурасы
22. Python программалоо тилиндеги else аркылуу бутактануу
23. Бир нече бутактануу if-elif-else
24. While циклы
25. While циклинин блок схемасы
26. While цикли менен болгон эки өзгөчө кырдаал
27. Функцияны аныктоо. Def оператору
28. Функцияны чакыруу
29. Функциялардын программага структура берүүсү