[Obshaja informacija](#Obshaja_informacija)

[Tipi dannih(kljuchevije slova, operatori)](#Tipi_Dannih)

[Kollekcii](#Kollekcii)

[Iskljuchenija i funkcii](#Iskljuchenije_i_funkcii)

[Moduli](#Moduli)

[OOP](#OOP)

[Faili](#FAILI)

[Procesi i potoki](#Procesi_i_potoki)

[Seti](#Seti)

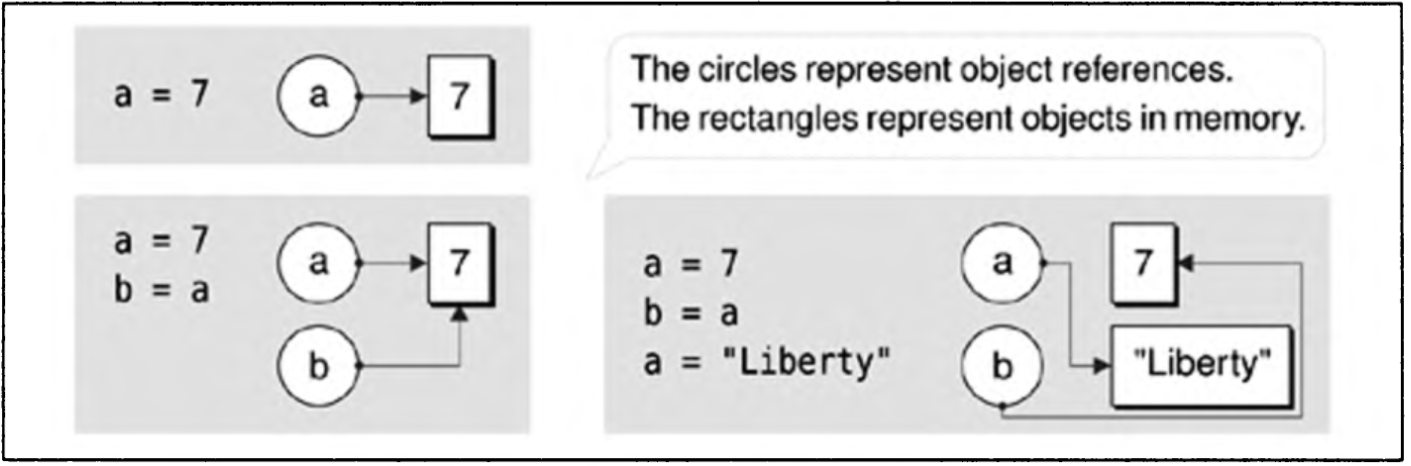
[Bazi dannih](#Bazi_Dannih)

[Reguljarnije virazhenija](#Reguljarnije_virazhenija)

[Grafika](#Grafika)

Obshaja informacija

* V Python vse objekti znajut k kakomu tipu oni prinadlezhat, poetomu ne trebujut tipizirovanija v ruchnuju
* V python ne nuzhno stavitj tochku s zapjatoj posle zavershenija komandi
* V Python vse javljajetsja objektami
* V Python kraine ne zhelateljno ispolzovatj tabuljaciju(tab). Vmesto etogo nuzhno ispolzovatj toljko probeli. Eto svjazano s tem, chto raznije sredi mogut po raznomu vosprinimatj odin i tot zhe kod napisannij s ispolzovanije tabuljacii
* Interpretator vipolnjajet vse posledovateljno (vesj kod sverhu do nizu)
* dva klassa (int i str). int ne imejet konkretnogo ogranichenije po pamjati v baitah i ogranichivajetsja lish pamjatju kompjutera.
* Int i str – javljajutsja ne izmennimi
* V python peremennije javljajutsja ekzempljarami klassov. Kazhdaja peremennaja eto ssilka na objekt opredelennogo tipa. No v Python net strogoj privjazki kak v Java, poetomu esli a = "blue" b = "green" a = b b = "violet" eto ne znachit, chto i a budet tozhe "violet"



* Vse znachenija kotorije ne ispolzujutsja v dalneishem, pogloshajutsja garbage collector
* str tip ispolzujet unicode
* V python false rassmatrivajetsja (peremennaja imejushaja znachenije false, None, pustaja kollekcija, znachenije peremennoj 0, ) Vse ostaljnoje ocenivajetsja kak true
* Vse ssilochnije peremennije ukazivajut na adres objekta. Poetomu, esli objekti javljajutsja ne izmenjajemimi(unmutable), to nikakih problem vozniknutj ne mozhet. No esli objekti (mutable) takije kak list, to:

List\_a = [1,2,3] List\_b = List\_a List\_a.append(4)

Print(List\_a) [1,2,3,4] oba avtomaticheski izmenilisj

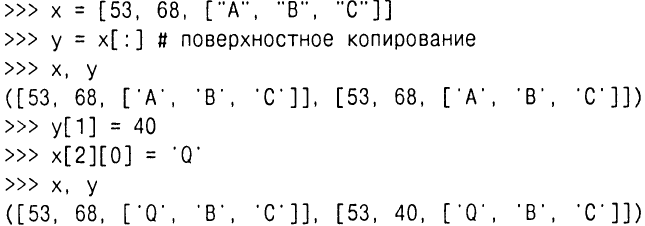
Print(List\_b) [1,2,3,4]

* Konvertacija ljuboj peremennoj s pomoshju zapisi int ili str pered nimi
* V Python vse biblioteki nazivajutsja moduljami
* V Python, sobstvennije funkcii luchshe ne nazivatj cherez \_\_, tak kak takogo roda imena, uzhe zarezervirovani pythonom dlja ego vstrojennih metodov
* Python < (version.3) imejet ne input, no raw\_input
* Python < (version.3) slovari ne javljajutsja uporjadachennimi i esli ispolozvatj for, kazhdij raz mozhet vivoditj druguju posledovateljnostj. V python 3.6 i vishe eta problema ishezla
* ljubije peremennije zapisannije do objavlenija funkcii, javljajutsja globaljnimi peremennimi i mogut ispolzovatsja v tele funkcii
* a = 1 + 2 + 3 + \

4 + 5 + 6

Simvol \ ispolzujetsja tak zhe dlja svjazanija strok. Stroka a = 1 + 2 + 3 + svjazalasj so strokoj 4 + 5 + 6

* V python 5 tipov posledovateljnostej: bytearray, bytes, str, tuple, list
* **!!!** V python, kogda udaljajetsja ssilki na objekti, i eti objekti bolshe ne svjazani ni s odnimi ukazateljami, sistema prisvaivajet im status “Gotovij k utilizacii”. No stoit pomnitj, chto tochno ne izvestno kogda mozhet proizoiti garbage collecting v pamjati i proizoidet li on voobshe. Poetomu, chtobi garantirovanno ochistitj pamjatj, nuzhno delatj eto vruchnuju – cherez try finally ili cherez instrukciju with
* Iterirujemij object, shitajetsja tot, kotorij sposoben vozvrashatj svoi znachenija po elementno (imejushij v soderzhanii metod \_\_next\_\_(), kotorij vozvrashajet po elementno znachenija i vozbuzhdajet iskljuchenije StopIteration posle ischerpanija vseh elementov)
* Kogda vizivajetsja cikl for... in to Python sozdajet iterator i zatem, ispolzujet vstrojennij metod \_\_next\_\_() chtobi proiti po vsem elementam i perehvativajet vobuzhdajushejesja iskljuchenije StopIteration v sluchaje okonchanija iteratora.
* Poverhnostnoje i glubokoje kopirovanije:



x i y javljajutsja spiskami (izmenjajemimi objektami). Esli prisvoitj y ssilku x, to mi poluchim obe ssilki, ssilajushijesja na odin i tot zhe object. Chtobi kopirovatj object mozhno ispolzovatj operaciju sreza kak v pokazannom vishe primere ili zhe ispolzovatj metod copy.copy() iz modulja copy. I v tom i v drugom sluchaje, budet proizvedeno poverhnostnoje kopirovanije. Pri poverhnostnom kopirovanii, proishodit poelementnoje kopirovanije (V nashem sluchaje 53, 68, [‘A’,’B’,’C’]). Elementi 53 i 68 budut skopirovani v y kak znachenija, no spisok [‘A’,’B’,’C’] budet skopirovan kak ssilka na etot object. Poetomu izmenjaja znachenije v etom spiske v x mi tak zhe menjajem znachenija v y. Chtobi sdelatj kopiju elementov spiska ljuboj glubini, ispolzujetsja metod copy.deepcopy().

* Class object – vse klassi v Python javljajutsja nasledujushimi ot klassa object
* .pyo – takoje rashirenije oznachajet, chto fail preobrazovan v bytecode i v dalneishem budet ispolzovan interpretatorom (obichno v bytecode preobrazujutsja moduli, dlja bole bistrogo obrashenija k nim )
* \_\_init\_\_.py – papka soderzhashaja takoj fail, avtomaticheski stanovitsja katalogom module i interpretator znajet, chto mozhet obrashatsja k nej. (sam init mozhet bitj pustim). Esli vpisatj v fail \_\_init\_\_.py peremennuju \_\_all\_\_ = {...}, gde v figurnih skobkah oboznachajutsja moduli kataloga kotorije neobhodimo budet zagruzhatj pri from ... import \*
* Python pozvoljajet programirovatj v funkcionaljnom, procedurnom, objektnom i smeshannom stile
* V terminologii jazika C++, vse metodi Python javljajutsja virtualjnimi
* V python ne sushestvujet ponjatija peregruzki metodov (blagodarja gibkoj sisteme peredachi argumentov metodu, otsutstvije peregruzki, ne javljajetsja problemoj)
* V python net ponjatija dostupov k objektu: protected, public, private, default (estj chastnije atributi \_\_ nachinajushijesja s simvolov dvoinogo podcherkivanija)
* Esli vizivajetsja metod klassa i on v nem neopredelen, to Python ishet etot metod v ego superKlasse i podnimajetsja vse vishe po drevu nasledovanija. Esli zhe metod ne nahoditsja vizivajetsja iskljuchenije AttributeError
* V Python imena peremennih ili nazvanije metodov nachinajushijesja s nizhnego podcherkivanija, USLOVNO shitajutsja PROTECTED. (eto lish uslovno. Na samom dele nikakogo specifikatora (identifikator) dostupa protected net)
* Imenam nachinajushimsja s dvoinogo podcherkivanija prisvaivajetsja status private. (V otlichii ot protected, private v python ne uslovnij i deistviteljno perenimajet vse osobennosti specifikatora dostupa)
* Peremennaja s imenem \_ v python vsegda sohranjajet posledneje znachenije v vipolnjajemoj programme
* “python.exe –m dis \*.py” – komanda pozvolit disassemblirovatj py fail v bytecode pitona

[TOP](#TITLE)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tipi dannih(Kljuchevije slova,operatori)

* Tipi dannih v jazike Python(kotorije mozhno ispolzovatj v procese konvertacii peremennih):
* int – celochislennije znachenije (int() sozdajet konvertaciju v celochislennoje znachenije. Int(“1101”,2) – konvertirujet stroku 1101 po osnovaniju 2 v celochislennoje znachenije. Esli v metode estj vtoroj argument, to funkcija ispolzujet ego, chtobi ponjatj iz kakoj sistemi schislenija proizvoditj konvertaciju. V DANNOM SLUCHAJE IZ BINARNOGO)
* float – znachenije s plavajushej tochkoj (po umolchaniju imejet 17 znakov posle zapjatoj dlja sobljudenija tochnosti)
* str – strokovije znachenija

*str(first,second,third):* – tri argumenta, pervij iz kotorih preobrazujet znachenije v tip string, votoroj ukazivajet na tip kodirovki pri preobrazovanii I tretij ukazivajet porjadok obrabotki oshibok kodirovanija

primer strok:

*tipi strok:* a = “””AAAAAA \ AAAA \ BB””” – kak kommentarii v kotorih simvol

ekranirovanija vosprinimajetsja kak prostoj backslash. Ekranirovatj zdesj pridetsja lish troinij kovichki ili troinij apostrofi – kotorije javljajutsja nachalom I koncom stroke

a = “Zdesj nuzhno ekranirovatj kavichki”

b = ‘Zdesj nuzhno ekranirovatj apostrofi’

ekran ne vosprinimajetsja v skobkah stroke (), [],{} I v “”” “””, ‘’’ ‘’’

a = “aaaaaaa” + \

“bbbbbbb” – odin iz sposobov objedinenija strok

A = (“AAAAAAAAAAAAAA”

“BBBBBBBBBBBBBB”

“CCCCCCCCCCCCC”) – votorj sposob objedinenija strok. Zdesj ne nuzhno stavitj ekran poskoljku vse virazhenija zakljucheni v skobki. Objazateljnoj chastju javljajutsja kavichki pri kazhdoj novoj stroke

A = r“AAAA \ BBBB “ CCCC ‘ TTTT ” – esli hotim ispolzovatj apostrofi I prochije simvoli bez ekranirovanija, stavim vperedi stroki – r. Takije stroke nazivajutsja raw stroki

* \uhhhh – UTF-16
* \hhhhhhhh – UTF-32
* ascii(symbol) – vivedet simvol v kode asci (Metod estj toljko v python 3.3 I vshe!!!!!)
* ord(symbol) – vivodit unicode sootvetstvujueshego simvola
* chr(Unicode) – vivodit simvol sootvetstvujushego Unicode
* str.capitalize() – vivodit stroku, pervij simvol kotoroj zaglavnaja bukva
* str.find(t, start, end) – metod poiska podstroki v diapazone start I end. (idet poisk s levo na pravo I vivodit poziciju pervogo vstrechnogo t v diapazone). Esli nichego ne budet naideno, metod vernet znachenije -1
* str.rfind(t, start, end) – metod reversnogo poiska podstroki v diapazone start I end. (idet poisk s parvo na levo I vivodit poziciju poslednego vstrechnogo t v diapazone). Esli nichego ne budet naideno, metod vernet znachenije -1
* str.replace() – zamenjajet ukazannije simvoli
* str.zfill(str1) – vozvrashajet stroke s, esli dlina kotoroj menshe dlini str1, do zapolnajetsja sleva nuljami do dlini str1
* reversed(str1) – vozvrashajet iterirujemij object, dlja vivoda kotorogo trebujetsja konvertacija v iterirujemij tip
* str = “aaa %s bbbb %s “ % (str1, str2) – string format, change %s with str1 and str2**(shitajetsja ustarevshim metodom v Python 3!!!!!!)**
* str.encode(“ASCII”, ) – vozvrashajet object tipa bytes. Vivodit kod simvolov stroke v zadannom standarte. Posle ukazanija standarta v kachestve argument metoda, ukazivajetsja tak zhe vtorostepennij argument – “ignore” – uberet vse simvoli ne vhodjashije v pervuju polovinu ascii tablici, “replace” – zamenit vse simvoli ne vhodjashije v pervuju polovinu ascii tablici na voprosi, “backslashreplace” – zamenit simvoli ne vhodjashije v pervuju polovinu tablici ascii na hex kod simovla
* str.decode(“utf-8”) – vozvrashajet simvoli, kod kotorih sootvetstvujet znachenijem v tablice ukazannogo standartaeps
* str.format() – metod preobrazovanija stroki

str = “aaa {0} bbbb {1}”.format(str1, str2) – change values {0} and {1} with str1 and str2 values. Tak zhe pozvoljajetsja vstavljatj figurnije skobki pered I posle skobok nulja “{{{0}}}”.format(str1)

Vozmozhno tak zhe v formate zapisivatj integer chisla dlja konkatenacii strok I vivoda TypeError ne budet (“aaa {0} BBBB {1}”.format(“ssss”, 200)) – zdesj 200 javlajjetsja integer no format avtomaticheski prevratit ego v string.

Tak zhe mozhno zapisatj (“{Who} turned {age} this year”.format(Who = “She”, age=88). Zdesj v kachestve polei metoda format ispolzujutsja imena argumentov metoda

Esli zapisatj “The {Who} was {0} last week”.format(12, who=”boy”) to v nolj avtomaticheski budet podstavleno 12. {0} javlajetsja pozicionnim argumentum, a {Who} imennovannim STOIT POMNITJ, CHTO POZICIONNIJE ARGUMENTI V FORMAT, DOLZHNI BITJ PERED IMENNIMI!!!

Mi mozhem ispolzovatj spiski v kachestve argumenta v format:  
stock = [“paper”, “envelopes”, “notepads”, “pens”, “paper clips”]

“We have {0[1]} and {0[2]} in stock”.format(stock)

Mi takzhe mozhem ispolzvoatj dictionary v kachestve argumentov format:

D = dict(animal=”elephant”, weight=12000)

“The {0[animal]} weighs {0[weight] kg}”.format(D)

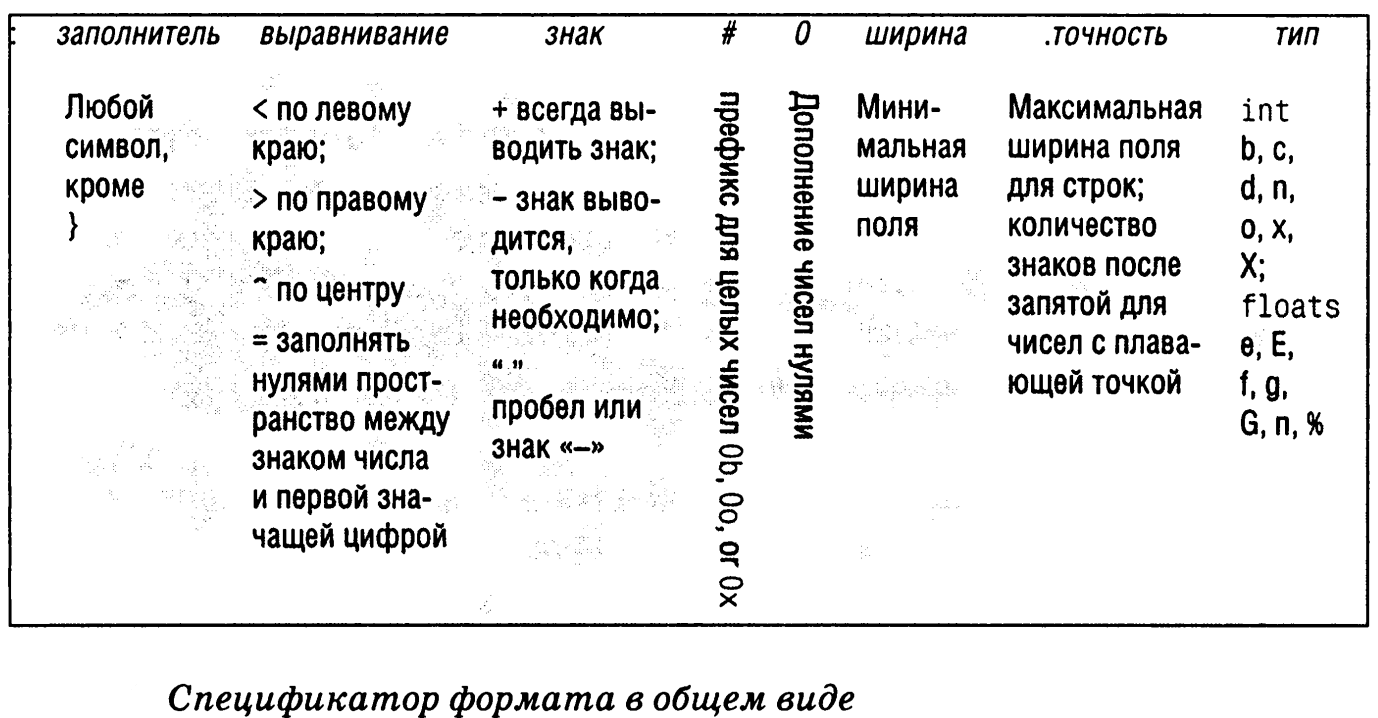
Mozhno dobavljatj imena module:

“random character {0.randint(0,100)}”.format(random)

!s – strochnoje otobrazhenije, !r – reprezentativnoje otobrazhenije, !a – vinuzhdennoje otobrazhenije ASCII simvolov

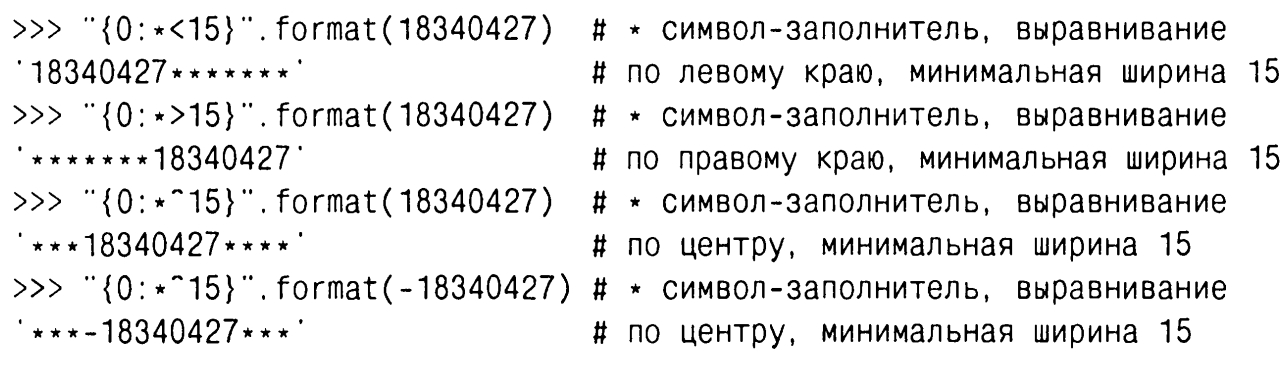
"AAAA {0!s} BBBB{1!r}".format(Decimal("3.94"), Decimal("3.94"))

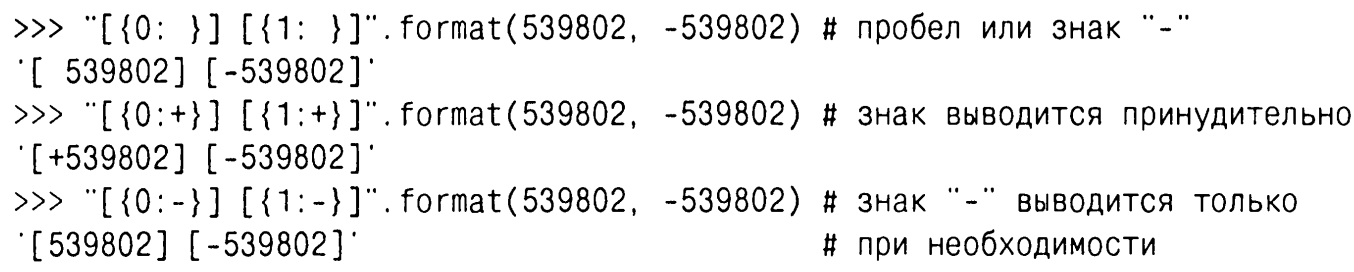
"AAAA 3.94 BBBB Decimal("3.94")"

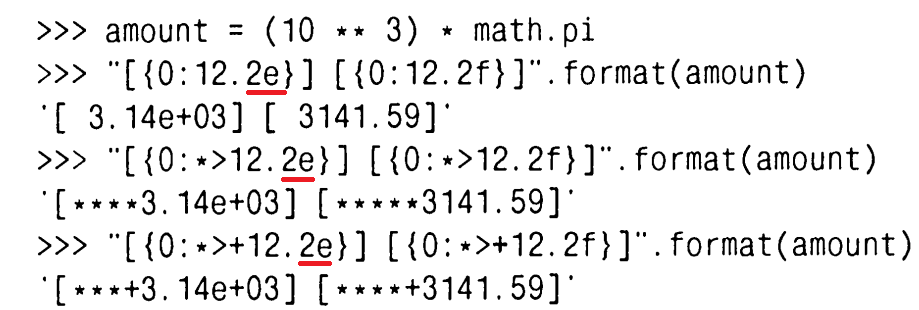


Int tipi dlja formatov – (b – binarnij, c – simvol unikoda sootvetstvujushij celomu chislu, d – decimaljnij, n – vivod chisel s uchetom regionaljnih nastroek(v zavisimosti ot nastrojek, razdeliteljnij znak drobnoj chaste beretsja tochkoj ili zapjatoj), o – vosmirichnij format, x – hex, X – hex s verhnim registrom)

Float tipi dlja formatov – (e – dlja vivoda eksponencialjnogo chisla s e v nizhnem registre, E - dlja vivoda eksponencialjnogo chisla s E v verhnem registre, f – standartnij vivod float chisla, g – dlja vivoda chisla v obshej forme(to estj, dlja nebolshih chisel ispolzujetsja rezhim f a dlja bolshih dopolnjajetsja eksponentoj), G – tozhe chto I format g(toljko ispolzujet libo f libo E formati), n - vivod chisel s uchetom regionaljnih nastroek (v zavisimosti ot nastrojek, razdeliteljnij znak drobnoj chaste beretsja tochkoj ili zapjatoj), % - procenti (umnozhajet znachenije chisla na 100 I dobavljajet znak procentov v konce. V kachestve formata ispolzujetsa f))







Podcherknutije znachenije krasnim, svidetelstvujut o kol-ve chisel posle zapjatoj

ASCII – 1 bait dlja sohranenija simvolov (pervaja polovina bitov dlja standartnih amerikanskih simvolov I vtoroje dlja vtorostepennih)

UCS-2 – kodirovka Unicode ispolzujushaja 2 baita dlja hranenija simvolov

UCS-4 – ispolzujet 4 baita dlja hranenija simvolov

Problema UCS standartov v tom, chto ne zavisimo ot koda simvola, oni zanimajut v pamjati vesj svoi razmer. K primeru

V ascii simvol kodirujetsja kak fe. Tozhe samoje v UCS-4 budet 00 00 00 fe. Dlja reshenija etoj problem bil vveden standart UTF dlja dinamicheskogo videlenija pamjati

UTF-8 – pri nebhodimosti videljajet pamjatj ot odnogo baita do 4

UTF-16 – ot 2 do 4

UTF-32 – toljko 4 baita

Esli vstrechajutsja simvoli iz tablici ascii, to kodi utf-8 I ascii identichni

* bool – boolevi znachenija (vivod javljajetsja str tipom)
* hex – shestnadcatirichnaja (vivod javljajetsja str tipom)
* bin – dvoichnaja (vivod javljajetsja str tipom) -> 0b01010101
* oct – vosjmerichnaja (vivod javljajetsja str tipom)
* Fractions – znachenija s neogranichennoj tochnostju(dlja matematicheskih i nauchnih oblastej)
* complex – kompleksnije chisla (javljajutsja ne izmenjajemimi) primer chisla: z = 2 + 3j

*Hranit dva atributa:* real I imag – print(z.real, z.imag) => 2.0, 3.0 – bez j

Print(2 + 3j + 5) => 7 + 3j – bez .0 dobavlenija na konce

*Primeri chisel*: 3.5+2j, 0.5j(zdesj real = 0), 4+0j(zdesj nuzhno ukazivatj nolj), -1-3.7j

*Metodi: //, divmod, %, pow* – ne primenimi k kompleksnim chislam

*Bolshinstvo metodov modulja math ne rabotajut s complex chislami. Dlja etogo estj cmath*

* complex() – metod complex bez argumentov, kotorij vernet 0j
* complex(6j) – metod complex s odnim argumentum vernet kopiju 6j
* complex(StrObj) – popitajetsja vernutj znachenije argumenta preobrazovannoje v complex
* complex(float a) – esli odin argument tipa float, to mnimoj chasti avtomaticheski prisvajetsja 0j
* complex(float a, float b) – obichnoje preobrazovanije v object complex
* complex.conjugate() – metod izmenenija znaka mnimoj chasti
* Sistema schislenija:

123 – desjatichnaja

0b0000 – dvoichnaja (esli ne ukazani vse biti baita, to shitajetsja chto ne ukazannije, eto starshije bit ii im prisvaivajetsja znachenije 0)

0o111 – vosjmerichnaja

0x0101 – shestnadcatirichnaja

* 7.1 javljajetsja tipom float, a 7.0 javlajetsja int
* print() – funckija vivoda, imejet ne ogranichennoje chislo pozicionnih argumentov i tri imenovannih (sep, end,file) kotorije javljajutsja eshe i argumentami po umolchaniju (to estj imejut nachaljnoje znachenije). Esli rassmatrivatj primer inicializacii funkcii print, to on budet vigljadet tak

print(\*arg,sep=....,end=....., file=.....)

sep – eto simvol razdelenija mezhdu elementami arg. Po umolchaniju probel

end – simvol stojashij v konce stroke pri zavershenii print. Po umolchaniju \n

file – potok vivoda. Po umolchaniju eto sys.stdout – potok standartnogo vivioda kotorij obichno predstavljajet konsolj vivoda

* print(var, type(var)) - vivedet na ekran znachenije peremennoj var i ejo klass
* print(), - zapjataja v konce printa govorit chto vse posledushije elementi budut vivedeni v toj zhe stroke
* print(“hello %s mister %s” **%** (string1,string2)) – procenti pozvoljajet objedenjatj stroke s peremennimi
* print(["o"] \* 5) - vivedet na ekran ["o","o","o","o","o"]
* print(“%.2f” % (variable)) – ogranichivajet kol-vo simvolov posle zapjatoj dlja tipa float
* print(string\_a[1:4]) – vivodit elementi stroke s 1 jacheiki do 4 ne vkljuchja
* print(string\_a[1:4:2]) – vivodit elementi stroke s 1 jacheiki do 4 ne vkljuchaja s shagom 2
* print(string\_a[::]) – vivodit znachenije vsej stroke ot nulja do dlini stroke ne vkljuchaja
* print(string\_a[::-1]) – kak toljko v strokah stavitsja otricateljnij shag, nachaljnoj jacheike prisvaivajetsja index -1

*v strokah indeksirovanije jacheik proishodit sledujushim obrazom:*

a = “abcde” – a imejet odnovremenno index 0 I -5. Index e imejet 4 I -1

* is, is not – operator sravnenija adresov ( a is b =– proverjajet sovpadajut li adresa objektov ssilochnih peremennih) osnovnoje otlichije ot ==:

a = “x” b=”y” a is b – false

a = “x” b=”y” a = b a is b – true

a = “x” b=”x” a is b – false a == b – true (== operator proverjajet znachenije ssilochnoj peremennoj a ne adress)

* print(0 < a < 10) – pri sluchaje, esli a = 5 i vhodit v diapason, print komanda avtomaticheski sgenerirujet logicheskije operacii i vivedit true
* operatori “in” i “not in” – proverjajut javljajetsja li peremennaja v spiske (stroki, kollekcii)

a = [1,2,3,4,5] b =3 b in a – true

* and, or, not – logicheskije operatori,

one = 1, two = 2 three = 3

one and two = 2

two and three = 3

three and one = 1

one or two = 1

two or three = 2

three or one = 3

* &, |, ^, <<, >>, ~ - bitovije operatori (i & j budet prisvojen rezultat konjunkcii bitov) (operatori bitovogo sdviga ispolzovatj toljko dlja int tipa)
* Pass – kljuchevoje slovo, kotoroje ispolzujetsja v teh mestah, v kotorih trebujetsja blok koda, no na dele, blok ne nuzhen. Pass oznachajet ne delatj nichego
* If, elif, else – operatori uslovij. Elif podrazumevajet else s novim uslovijem. Sam else ne podrazumevajet uslovija. (ne zabivatj o “:”, signalizirujusheje o nachale bloka koda. Tak zhe pomnitj o chetireh probelah (zhelateljno ne tabuljaciju) )

If (expression1):

Some code

elif (expression2):

Some code

elif (expression3):

Some code

else:

Some code

* Vozmozhno sozdavatj znachenije s uslovijem

A = 100 if i < 10 else 200

A budet prisvojeno chislo 100 esli znachenije i menshe 10, v protivnom sluchaje budet prisvojeno chislo 200

* global – pozvoljajet sozdatj lokaljnuju peremennuju v funkcii, oblastj vidimosti kotoroj budet javljatsja globaljnoj

def func1():

global a

a = “something”

print(a)

func1() => vivedet na ekran something

print(a) => vivedet na ekran something bez vizova funkcii

* while – cikl, vipolnajemij poka deistvujet zadannoje uslovije

While expression1:

Some code

* while/else - blok v kotorom else vipolnitsja esli cikl while zakonchitsja po usloviju i ne prervetsja. Esli zhe v tele While realizovatj break to cikl prervetsja i else ne vipolnitsja
* for/else - tot zhe princip dlja etogo bloka, kak i dlja while/else
* break – operator prerivanija vnutrennego cikla
* continue – prerivanije iteracii cikla i vozvrat v nachalo bloka cikla
* for ... in – cikl

for x in list\_a:

print(x)

* iter() – sozdajet iterator iterirujemogo objekta

a = “abcdef”

b = iter(a)

while(1):

try:

print(next(b))

except StopIteration as T:

break

V primere proizvoditsja poolementnoje izvlechenije iz objekta b. (analog for in)

* next() – vstrojennij metod kotorij prohodit po iteratoru i vobuzhdajet iskljuchenije StopIteration v sluchaje okonchanija iteratora
* enumerate

choices = ['pizza', 'pasta', 'salad', 'nachos']

print ('Your choices are:')

for index, item in enumerate(choices):

print index + 1, item

V etoj programme item zadajetsja znachenije choices i index zadajetsja avtomaticheski

sgenerirovannij spisok indeksov (enumerate)

* zip

list\_a = [3, 9, 17, 15, 19]

list\_b = [2, 4, 8, 10, 30, 40, 50, 60, 70, 80, 90]

for a, b in zip(list\_a, list\_b):

# Add your code here!

if (a > b):

print(a)

elif (a < b):

print(b)

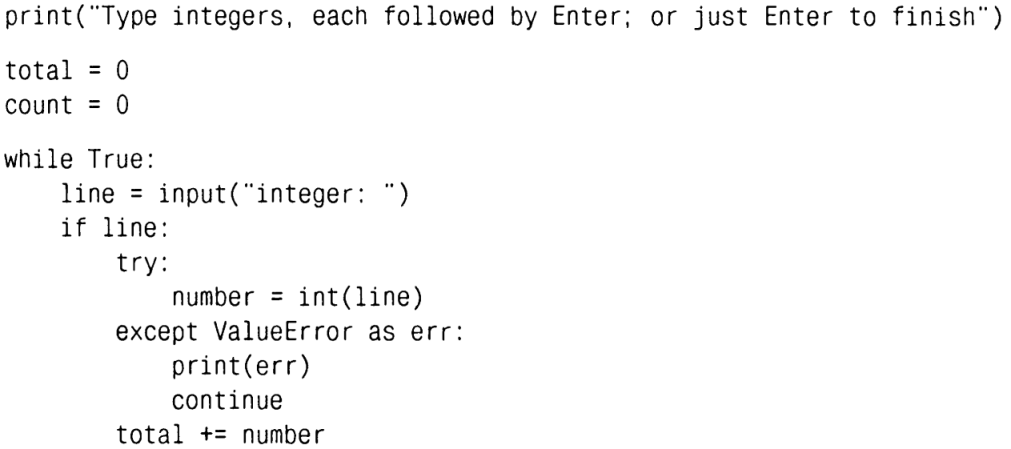
Funkcija zip pozvoljajet objedinjatj spiski i zakanchivajet cikl, kogda zakonchatsja elementi

na samom korotkom spiske. zip mozhet prinimatj i bolshe chem dva spiska. (a rinimajet list\_a, b beret list\_b)

* +,-,\*,/ Arifmeticheskije operatori. Esli delitj dva chisla 3/2=1.5 to rezuljtat ne budet celim kak eto mozhet bitj v C jazike dlja int. V takom delenii,rezultat vsegda budet s plavajushej tochkoj (v otlichii ot togo zhe C jazika v kotorom nuzhno uchitivatj tipi delitelja i delimogo)
* Esli a = 10 a+=2 print(a) poluchim rezuljtat 12. No stoit zapomnitj, chto int objekti javljajutsja ne izmennimi, poetomu posle prisvojenije a dvoiki, sozdajetsja novij object ravnij 12 i ssilochnoj peremennoj a prisuzhdajetsja address novogo objekta. V sluchaje, esli na starij object = 10 ne budet ukazivatj ni odna ssilochnaja peremennaja, on budet unichtozhen sborshikom musora

Takoj operator rabotajet bistre chem a = a + 2

* Operatori + i += mozhno peregruzhatj. A imenno ispolzovatj dlja konkatenacii strok
* Input – komanda vvoda s klaviaturi. A = input(“write something”) – Zdesj A imejet znachenije vpisannoje s klaviaturi, a tekst write something budet viveden pered vvodom.



V etoj programme budet vizvana oshibka, esli chislo vvedennoje polzovatelem ne vozmozhno konvertirovatj v int

* // - operator celochislennogo delenija
* / - delenije vsegda, dazhe esli delitelj i delimoje javljajutsja celimi chislami, vozvrashajet znachenije s plavajushej tochkoj
* Kommentarii v Python oboznachajutsja:

# - dlja odnostrochnih

‘’’ ‘’’ ili “”” “”” – dlja blochnih

* True i False – objazateljno pisatj s bolshoj bukvi
* return – vozvrashajet znachenije funkcii i prerivajet ejo
* as – vozmozhno pereimenovatj objekti (ili modulj)
* \*\* - eksponencialjnij operator. 2 \*\* 10 = 1024 - znachit 2 v stepeni 10
* % - moduljnij operator (vivodit ostatok ot delenija) 3 % 2 = 1
* divmod(x,y) – funkcija vozvrashajushaja tuple sostojashego iz dvuh znachenij tipa int: Pervoje znachenije eto chastnoje delenija x na y, a vtoroje znachenije, eto ostatok ot delenija
* pow(x,y) – vozvodit x v stepenj y ( analog x\*\*y)
* pow(x,y,z) – korotkaja zapisj (x\*\*y)% z
* round(x,n) – vozvrashajet okruglennij argument x(tipa float) preobrazuja ego v int.(Esli argument n ne zadan). Esli zhe n prisutstvujet, togda okrugljajet x do (argument n) znakov posle zapjatoj( zdesj x ostajetsja float)
* range(a) – avtomaticheski sozdajet spisok iz chisel ot nulja do a
* range(6) - sozdajet spisok ot nulja do 6 ne vkljuchiteljno
* range(1,6) - list s 1 do 6 ne vkljuchiteljno
* range(1,6,2) - sozdast list s 1 do 6 ne vkljuchaja, ispolzuja shag 2. rezultirujushij spisok = [1,3,5]
* \ - operator otdelenija simvolov (ekranirovanije)

“There isn\’t point in this” – ispolzujetsja dlja otdelenija apostrofa. (Inache Python dumajet chto v etom meste zakanchivajetsja stroka)

* String\_a.lower() – prevrashajem stroku v malenjkije simvoli
* Str**.**islower – proverjajet zapisani li simvoli v stroke v nizhnem registre
* String\_a.upper() – prevrashajem stroke v bolshije simvoli
* Str**.**isupper – proverjajet zapisani li simvoli v stroke v verhnem register
* str.swapcase – vozvrashajet stroke s izmenenijem simvolov verhnego registra na niznhnie, a nizhnije na verhnije
* String\_a.isalpha() – vozvrashajet True esli vse simvoli stroke javljajutsja bukvami. V inom sluchaje False
* String\_a.isdigit() - vozvrashajet True esli vse simvoli stroke javljajutsja ciframi. V inom sluchaje False
* String\_a.split() – prevrashajet stroke v spisok slov razdelennih probelom po umolchaniju
* String\_a.split(“,”,5) – v etom metode, razdelitelem javljajetsja zapjataja, a znachenije 5 ukazivajet na maksimaljnoje kol-vo elementov pri razbijenii slova. Primer:

Str = “aa bb cc dd ee ff” print(str.split(“ ”,2) => [‘aa’,’bb’,’cc dd ee ff’]

Rezultat soderzhit tri elementa – aa, bb, cc dd ee ff

* str.splitlines() – vozvrashajet spisok strok, razbitih po simvolu
* String.strip() – udaljajet opredelennije simovli iz stroke (po umolchaniju probeli). Primer string.strip(“[]{}()”) – eti simvoli budut udaleni iz stroke
* String.rstrip() – udaljajet simvoli s konca stroke (po umolchaniju probeli i simvoli perevoda stroke takije kak \n )
* join

for a in list\_a:

print(" ".join(a))

zdesj join pozvoljajet objedenitj jacheiki ODNOJ STROKI TIPA STR, eto pozvolit otobrazitj elementi spiska list\_a bez apostrofov i zapjatih, objedeniv ih vmeste cherez (probel v dannom sluchaje) (dlja spiska spiskov objazateljno for, kotorij proidet po vsem strokam)

* dir() – vozvrashajet spisok imen iz konkretnoj oblasti vidimosti (vstrojennih atributov jazika Python) K primeru \_\_builtins\_\_ javljajetsja module v kotorom opredeleni vse vstrojennije atributi jazika (sum, ArithmeticError, min,...)

print(dir(math)) – vernet spisok vseh atributov modulja math (math modulj dolzhen bitj predvariteljno importirovan)

* abs() – modulj
* isinstance(x, bu) – proverjajet, javljajetsja li object x ekzempljarom klassa bu
* help() – vizivajet help dlja konkretnogo argumenta

help(type) – vidast vse atributi I metodi dannogo klassa

* \_\_code\_\_ - vstrojennij atribut python, soderzhashij kod funkcii. K primeru esli estj funkcija def a i funkcija def b, to esli a.\_\_code\_\_=b.\_\_code\_\_ to pri vizove funkcii a , avtomaticheski vipolnitsja funkcija b, tak kak v attribute koda funkcii a teperj soderzhitsja kod funkcii b

[TOP](#TITLE)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Kollekcii

* list = [] tuple = () dictionary = {“key” : value}
* spiski hranjat ssilki na objekti a ne sami objekti
* tuple i list - tuple eto spisok kotorij posle inicializacii ne vozmozhno izmenitj (beretsja v kruglije skobki) list javljajetsja modificirovannim spiskom (beretsja v kvadratnije skobki)
* dictionary, tak zhe kak i spiski, javljajetsja modificirujemim(mutable)
* len(a) - dlina spiska
* Imenovannije kortezhi (tuple)   
   collections.namedtuple(‘Sales’, ‘product price id’)

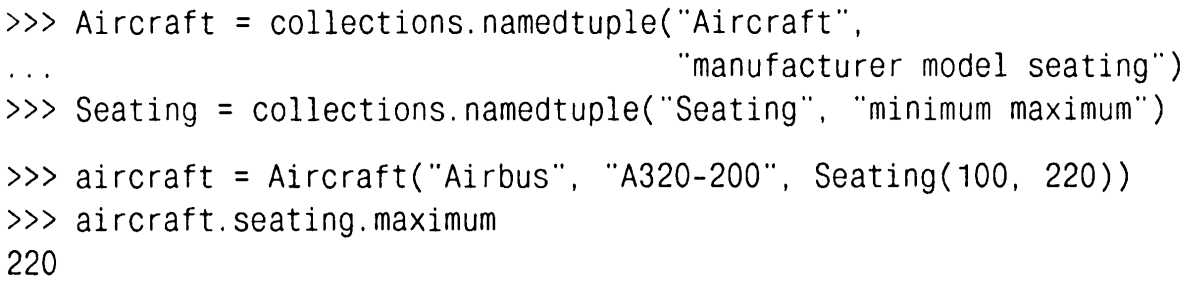
Modulj collections pozvoljajet ispolzovatj metod namedtuple v kotorom ispolzujetsja dva argumenta: Sales – eto nazvanije tuple, a vtoroj argument opredeljajet imena polej v tuple razdelennih probelom. V kachetsve vozvrashajemogo tipa metoda, ispolzujetsja objekt tipa (v dannom sluchaje Sales)

Primer: sales = []; sales.append(Sales(„lemon”, 25, „id-076”)); sales = append(Sales(„meat”, 80, „id-006”)

Total = „”

For sale in Sales:

Total += sale.product + sale.id



Eshe odin primer ispolzovanija imenovannih tuple. Pri prisvojenii objekta imenovannogo tuple peremennoj, obrashajtsja k poljam prihoditsja cherez tochku (obrashenije k polam objekta)

world\_map **=** collections.namedtuple**("World\_Map"**, **"continent country")**country **=** collections.namedtuple**("Country"**, **"country city street")**street **=** collections.namedtuple**("Street"**, **"street home\_id apartment")**a **=** world\_map**("America"**, country**("USA"**, **"Chicago"**, street**("White Street"**,**"25"**,**"108")))**print**(**a.country.street.apartment**)**

Drugoj primer pokazivajet kak mozhno obrashatsja k poljam objektov

* list\_a.append(var) - dobavljajet v list list\_a ssilku na objekt var (dopolnjajet spisok). Prinimajet lish odin argument
* list\_a.append(["o"]\* 5) - dobavljajet v jacheiku list\_a spisok iz ["o","o","o","o","o"]
* list.append(list\_a,”some value”) – list zdesj kljuchevoje slovo i pozvoljajet modificirovatj list\_a dobaviv v nego znachenije “some value”
* list\_a.extend(“AAAA”,””BBBB”) – rashirjajet spisok list\_a znachenijami AAAA i BBBB
* list\_a.insert(3,”Some value”) – vstavljajet v opredelennuju poziciju spiska list\_a znachenije “Some value”
* list\_a[2:2] = [“AAA”] – odin iz sposobov vstavitj znachenije AAA v spisok list\_a na poziciju 2 s pomoshju sreza nulevoj dlini.

esli imejetsja spisok list\_a = [‘A’,’B’,’C’,”D”,”E”,”F”]

list\_a[2:5] = [‘X’,’Y’]; print(list\_a) -> [‘A’,’B’,’X’,’Y’,’F’]

V takoj vstavke, za kulisami proishodit vnachale udalenije vseh objektov s 2 po 5 poziciju ne vkljuchiteljno i spisok imejet vid list\_a = [‘A’,’B’,’F’], a zatem proishodit sama vstaka X i Y.

Esli zhe v spiske srez menshe chem kol-vo vstavljajemih elementov, spisok uvelichitsja

List\_a[2:3] = [‘XX’,’YY’,”ZZ”,’JJ’]

Print(list\_a) -> [‘A‘,‘B‘,‘XX‘,‘YY‘,‘ZZ‘,‘JJ‘,‘D‘,‘E‘,‘F‘]

Esli zhe srez s nulevoj dlinoj, to takim obrazom mi nichego ne udaljajem iz spiska putem vstavki drugih znachenij, a lish vstavlajem znachenije na opredelennuju poziciju

List\_a[4:4] = [1,2,3]

Print(list\_a) -> [‘A’,’B’,’C’,”D”,1,2,3,”E”,”F”]

Zamena kazhdogo nechetnogo indeksa spiska na znachenije 0

List\_a[1::2] = [0]\*3 – zdesj kol-vo nulej dolzhno sovpadatj s razmerom izvlechennogo spiska( v dannom sluchaje u nas razmer raven 3. Index 1, 3, 5)

Print(list\_a) -> [‘A’,’0’,’C’,”0”,”E”,”0”]

Stoit pomnitj, chto dlja prisvaivanija srezu znachenija, sami znachenija dolzhni bitj iterirujemimi

* list\_a.remove(“Some value”) – udaljajet jacheiku spiska v kotorom hranitsja znachenije “Some value” (nomer jacheiki ukazivatj nelzja, lish znachenije)
* list\_a.pop(index\_number) - udaljajet jacheiku lista pod nomerom index\_number
* list\_a.pop() – metod pop bez argumentov, vozvrashajet znachenije poslednego elementa spiska. Togo, kotorij on udalil
* sum(list\_a) – summirujet vse znachenije spiska
* min(list\_a) – nahodit minimaljnoje znachenije spiska
* max(list\_a) – nahodit maksimaljnoje znachenije spiska(pervoje sleva)
* sort(key=’’) - STR1.sort(key=str.upper) – etot metod imejet kljuchevoje slovo key, kotoroje beret v kachestve znachenije kakoj-to metod, v dannom sluchaje upper() i peredajet emu v kachestve parametra, kazhdij element spiska STR1. Takim obrazom, na vihode mi poluchim kazhdij element spiska STR1 s simvolami verhnego registra i otsortirovannogo po vozrastajushemu znacheniju ili po alfavitu
* map(function,list) – Etot metod pozvoljajet primenitj funkciju function ko vsem elementam list
* sorted(list\_a,key=...) – vozvrashajet sortirovannij list. Key ukazivajetsja metod kotorij budet ispolzovatsja nad kazhdim znachenijem. (key ne izmenjajet sami znachenija, lish ispolzujet ego dlja proverki)

sorted(iterable,\*,key=None,reverse=False) – shablon metoda

* list\_a.reversed() – spisok v obratnom porjadke
* \*list\_a – operator raspakovivanija posledovateljnosti

primer prisvaivanija spiskam znachenii (so znakom \*):

a,b,\*list\_a,c = 1,2,3,4,5,6,7,8,9,10 ili a,b, \*list\_a,c = [1,2,3,4,5,6,7,8,9,10]

v oboih sluchajah, a=1,b=2,c=10, list\_a = [3,4,5,6,7,8,9]   
Python avtomaticheski ponimajet kak raspredeljatj prisvaivanije elementov

esli v print vivesti lista\_a so zvezdochkoj, to proizoidet avtomaticheskaja raspakovka lista  
print(\*list\_a) -3 4 5 6 7 8 9

print(\*list\_a[1:]) – 4 5 6 7 8 9

* filter(uslovije, list\_a) – filter vozvrashajet spisok elementov list\_a, udovletvorjajushih usloviju. (Filter eto analog for... in vmeste s uslovijem if)

filter(Lambda x : x % 3 ==0,list\_a) – vazhno ponimatj, chto v etoj zapisi x javljajetsja kazhdoj jacheikoj list\_a, kotorij proverjajetsja na uslovije lambdi

* Vozmozhno sozdavatj spisok s pomoshju opredelennih uslovij, a imenno generatorov spiskov. Primer:

evens\_to\_50 = [i \* 2 for i in range(51) if i % 2 == 0]

print evens\_to\_50

i \* 2 govorit, chto znachenije iteratora, posle odnogo cikla, budet uvelicheno vdvoje (v dannom sluchaje, esli “i” udovletvorjajet logicheskomu usloviju) **Esli nuzhno sozdatj spisok vishe opisannim sposobom, to pered for objazateljno dolzhen stojatj argument. (imenno on zadajet, chto budet zapisano v spisok) v nashem sluchaje i \* 2, no mozhet bitj prosto i.**

* List\_a.index(“Big”) – vivedet nomer jacheiki imejushej znachenije Big. Esli znachenije ne budet naideno, vernetsja ValueError.
* List.count() – shitajet kol-vo objektov v spiske
* List\_a.rindex(“Big”) - vivedet nomer jacheiki imejushej znachenije Big vstretivshijsja v poslednij raz. Esli znachenije ne budet naideno, vernetsja ValueError.
* del dict[“key\_name”] – del javljajetsja analogom metoda remove v liste, udaljajet jacheiku key/value pair v kotoroj kljuch raven key\_name. Posle udalenija, sdvigajet vse elementi dictionary (fakticheski, nelzja skazatj chto metod del udaljajet object. Del lish udaljajet ssilku na object, a sam object ostajetsja v pamjati poka ego ne uberet garbage collector)
* dict1.items() – vozvrashajet vesj spisok key/value pair dictionary. Kazhdaja jacheika etogo spiska, sodzerzhit tuple znachenije key/value => [(key1, 1),(key2,2),(key3,3)]

(v Python 3 vozvrashajemije dannije etogo metoda, javljajutsja predstavlenijem slovarja. Dopolniteljnij spisok dlaj hranenija rezultata ne sozdajetsja, a sushestvujet otobrazhenije slovarja. Vse chto proishodit so slovarem, proishodit i s otobrazhenijem. V python 2, v kachestve vivoda metoda, sozdavalsja otdeljnij spisok s rezultatami)

* dict1.keys() – vivodit vse kljuchi slovarja (v tipe tuple)

(v Python 3 vozvrashajemije dannije etogo metoda, javljajutsja predstavlenijem slovarja. Dopolniteljnij spisok dlaj hranenija rezultata ne sozdajetsja, a sushestvujet otobrazhenije slovarja. Vse chto proishodit so slovarem, proishodit i s otobrazhenijem. V python 2, v kachestve vivoda metoda, sozdavalsja otdeljnij spisok s rezultatami)

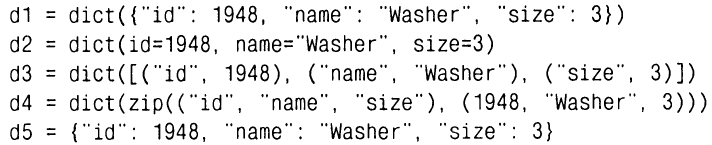
* dict1.values() – vivodit vse znachenija values slovarja (v tipe tuple)

(v Python 3 vozvrashajemije dannije etogo metoda, javljajutsja predstavlenijem slovarja. Dopolniteljnij spisok dlaj hranenija rezultata ne sozdajetsja, a sushestvujet otobrazhenije slovarja. Vse chto proishodit so slovarem, proishodit i s otobrazhenijem. V python 2, v kachestve vivoda metoda, sozdavalsja otdeljnij spisok s rezultatami)

* print(dict[“key”]) – vivedet znachenije jacheiki s kljuchom key
* s = dict(key1 = 1, key2 = 2, key = 3) – inicializacija dictionary cherez metod dict s ispolzovanijem imenovannih argumentov
* dict.copy() – funkcii kopirovanija slovarja
* {}.fromkeys(“ABCD”,2) - > {A : 2, B : 2, C : 2, D : 2} – fromkeys metod sozdajet dictionary s kljuchami ukazannimi v pervom argument funckii ( argument dolzhen bitj iterirujemim objektom), a vtoroj argument zadajet znachenije dlja vseh kljuchei
* for a in dictionary:

print(a) - vivedet vse kljuchi i znachenija dictionary

Sposobi sozdanija dict:



* **Esli k slovarju budet dobavlen kljuch, kotorij uzhe sushestvoval do etogo v dict, predidushij kljuch budet zamenen novim**
* **Esli obratitsja k kljuchu dict, kotorogo ne sushestvujet, budet vozbuzhdeno iskljuchenije KeyValue**
* **Esli obrashatsja k dict cherez for:**

**For a in dict1:**

**Print(a) - > key1, key2, key3 # to budut vivedeni toljko kljuchi**

* Dict1.clear() – otchishajet slovarj
* defaultdict – word = collections.defaultdict(int) – slovarj so znachenijem po umolchaniju
* Dict1.get(“key”) – vozvrashajet znachenije sootvetstvujushego kljucha ili None esli kljuch otsutstvujet
* Dict1.get(“key1”, other) - vozvrashajet znachenije sootvetstvujushego kljucha ili znachenije peremennoj other esli kljuch otsutstvujet
* Dict1.pop(“key1”) – udaljajet znachenije kljucha key1 iz slovarja i vozvrashajet eto znachenije kak rezultat

Map = dict(A = 1, B = 2, C = 3)

Map = Map.pop(“A”)

Print(Map) -> 1

Map= dict(A = 1, B = 2, C = 3)

Map.pop(“A”)

Print(Map) -> {‘B’ : 2, ‘C’: 3}

* Dict1.update(dict2) – dobavljajet v slovarj dict1, slovarj(ili iterirujemij object imejushij pari kljuch-znachenije) kotorij otsutstvujet v slovare dict1. Esli kljuchi v dict2 uzhe estj v dict1, to novije kljuchi zamenjat starije.
* files **= {**i **:** os.path.getsize**(**i**) for** i **in** os.listdir**()}**

generator znachenij dlja dictionary.

* + i += dlja spiskov tak zhe mogut bitj ispolzovani, no poskoljku spiski (list) javljajutsja izmenjajemimi, to object modificirujetsja a ne sozdajetsja zanogo

A = [“a”, “b”, “c”]

A+=[”d”]

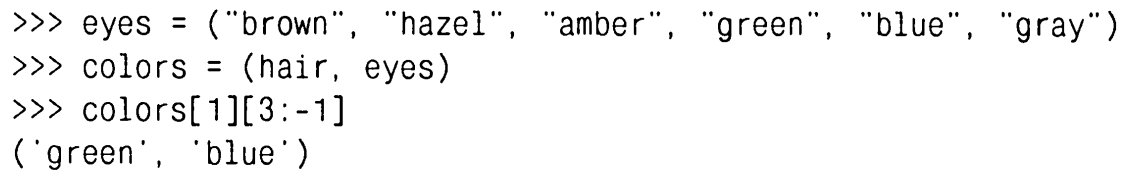
Print(A) – [a,b,c,d]

Esli

A+=”ba” (bez kvadratnih skobok, to budet dobavleni ba kak raznije jacheiki. Eto proishodit potomu, chto += operator s pravoj storoni, v spiskah tak zhe ispolzujet iterirujemij object. A stroke javljajetsja takovim, poetomu on razbivajet ejo na chasti)

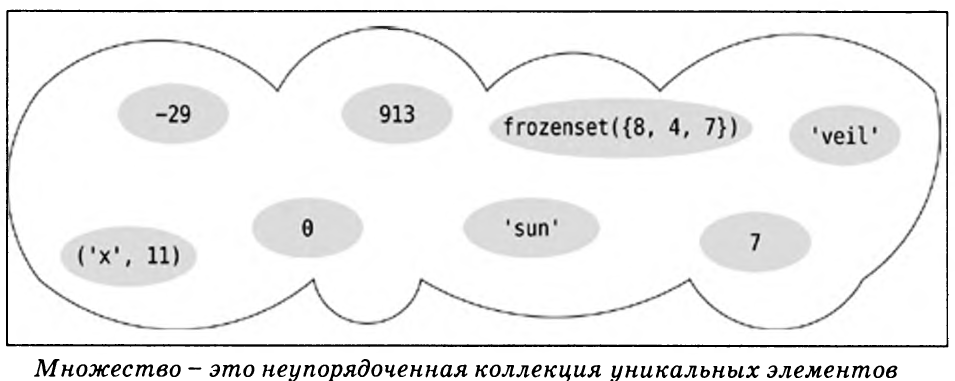
Print(A) – [a,b,c,d,b,a]

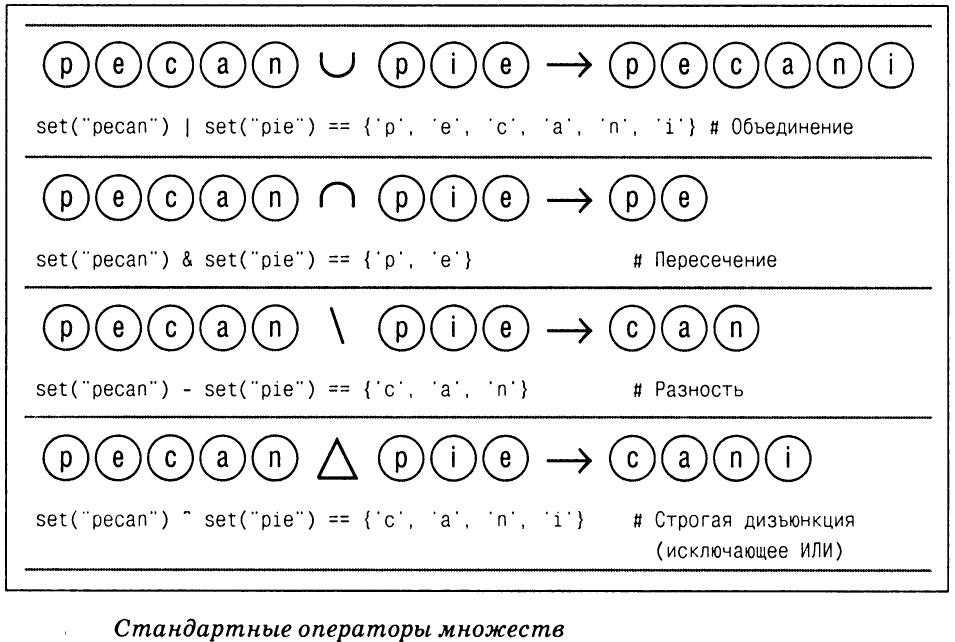
* Pervij operator += v spiskah dolzhen bitj iterirujemim objektom
* V tuple (kortezhi) mozhno prisvaivatj tak zhe elementi, kak pokazano s hair v primere nizhe:



* set() – mutable (izmenjajemij) ne uporjadochennoje mnozhestvo. Soderzhit lish unikaljnije znachenija (bez dublikatov) i ne imejet nikakoj indeksacii. K elementam mnozhestva set, nelzja obratitsja cherez indeks i k dannomu tipu dannih ne primenima operacija sreza. Prisvojenije znachenij objektu set mozhno sdelatj dvumja sposobami   
  S = {“a”, 12, “bu”} – stavjatsja figurnije skobki (kak v dictionary) no bez kljuchej. Esli neobhodimo sozdatj pustoj set. Eto delajetsja cherez funkciju set(). Sozdatj S = {} neljza, tak kak eto budet znachitj, chto objavlen pustoj dictionary. Tak zhe mozhno ispolzovatj metod set(“boom”) dlja sozdanija mnozhestva. Metod set(“boom”) vernet {‘o’,’b’,’m’}   
  Primer, kogda nuzhno izbavitsja ot dublikatov v spiske x: y = list(set(x)) budet sozdan spisok, bez dublikatov i s sluchainom raspolozhenii elementov.

K mnozhestvam primenimi logicheskije operatori (v to vremja kak spiski, tuple i dict ih ne imejut)





* set1.add(x) – dobavljajet element x vo mnozhestvo Set1, esli etot element, eshe ne prisutstvoval v nem
* set1.clear() – otchishajet mnozhestvo
* set1.copy() – vozvrashajet poverhnostnuju kopiju mnozhestva
* set1.difference(x) – ekvivalent (set1 – x). Vozvrashajet znachenije mnozhestva set1, kotorije ne bili v mnozhestve x
* S.discard(x) – udaljajet element x iz mnozhestva S, esli on prisutstvujet. Esli zhe net, nikakogo iskljuchenije vivoditsja ne budet.
* S.remove(x) – udaljjaet znachenije x iz mnozhestva S. Esli zhe element x ne naiden, budet vozbuzhdeno iskljuchenije ValueError
* S.intersection(x) – analog s & x. Vozvrashajet toljko obshije elementi mnozhestv
* S.isdisjoint(x) – vozvrashajet True esli oba mnozhestva ne imejut obshih elementov
* S.issuperset(x) – vozvrashajet True, esli x javljajetsja podmnozhestvom mnozhestva S ili zhe, oba mnozhestva identichni (analog s >= x)
* S.issubset(x) – analog s <= x. Vozvrashajet True esli s i x mnozhestva identichni ili esli s javljajetsja podmnozhestvom x
* S.pop() – udaljajet sluchainij element mnozhestva S. Esli S pustoje, vozvrashajet iskljuchenije KeyError
* S.symmetric\_difference(x) – analog S ^ x. Vozvrashajet vse unikaljnije elementi oboih mnozhestv. To estj teh, kotorih netu odnovremnno v mnozhestve x i v mnozhestve S
* S.union(x) – analog S | x. Vkljuchajet vse elementi oboih mnozhestv .

Primer ispolzovanija argumentov komandnoj stroke:

If (len(sys.argv) != 0) or (sys.argv[1] in {‘-h’, ‘--help’}):

* frozenset() – ne izmenjajemij tip mnozhestv

esli ispolzovatj logicheskije operatori na frozenset i set, to rezultat vsegda budet takoj tip, kotorij imejet levij argument logicheskogo operatora. Primer

a = frozenset(‘AB’) & set(“BC”) -> a = {“B”} tipa frozenset

a = set(‘AB’) | frozenset(‘BC’) -> a = {“ABC”} tipa set

* (VSTRECHALOSJ V RAZDELE OBSHAJA INFORMACIJA)

**Vse ssilochnije peremennije ukazivajut na adres objekta. Poetomu, esli objekti javljajutsja ne izmenjajemimi(unmutable), to nikakih problem vozniknutj ne mozhet. No esli objekti (mutable) takije kak list, to:**

**List\_a = [1,2,3] List\_b = List\_a List\_a.append(4)**

**Print(List\_a) [1,2,3,4] oba avtomaticheski izmenilisj**

**Print(List\_b) [1,2,3,4]**

* (VSTRECHALOSJ V RAZDELE TIPI DANNIH)

**range(a) – avtomaticheski sozdajet spisok iz chisel ot nulja do a**

* Izvlechenije iz spiska opredelennih elementov. Primeri:

List\_a[1:4] – vivodit elementi spiska s 1 jacheiki do 4 ne vkljuchja

List\_a[1:4:2] – vivodit elementi spiska s 1 jacheiki do 4 ne vkljuchaja s shagom 2

List\_a = list\_b[1:3] – zapishet v novij list\_a iz list\_b jacheiki s 1 po 3 ne vkljuchiteljno

List\_b = [1,2,3,4,5] list\_a = list\_b[3:5] print(list\_a) -> [4,5]

String my\_list[2:] – vozmet vse elementi so vtoroj jacheiki do konca

String my\_list[:5] – vozmet vse elementi s nachala do 5 elementa ne vkljuchiteljno

String my\_list[::2] – vozmet s nachala, do konca vse jacheiki po shagu 2. (po umolchaniju nachalo = 0, konec = len(), a shag = 1)

String my\_list[::-1] – otricateljnij shag, pozvoljajet proitisj po spisku s konca do nachala (reverse list)

* (VSTRECHALOSJ V RAZDELE TIPI DANNIH)

**enumerate**

**choices = ['pizza', 'pasta', 'salad', 'nachos']**

**print ('Your choices are:')**

**for index, item in enumerate(choices):**

**print index + 1, item**

**V etoj programme item zadajetsja znachenije choices i index zadajetsja avtomaticheski**

**sgenerirovannij spisok indeksov (enumerate)**

* (VSTRECHALOSJ V RAZDELE TIPI DANNIH)

**zip**

**list\_a = [3, 9, 17, 15, 19]**

**list\_b = [2, 4, 8, 10, 30, 40, 50, 60, 70, 80, 90]**

**for a, b in zip(list\_a, list\_b):**

**# Add your code here!**

**if (a > b):**

**print(a)**

**elif (a < b):**

**print(b)**

**Funkcija zip pozvoljajet objedinjatj spiski i zakanchivajet cikl, kogda zakonchatsja elementi**

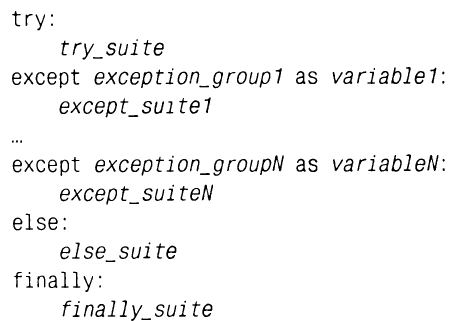
**na samom korotkom spiske. zip mozhet prinimatj i bolshe chem dva spiska. (a rinimajet list\_a, b beret list\_b)**

[TOP](#TITLE)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

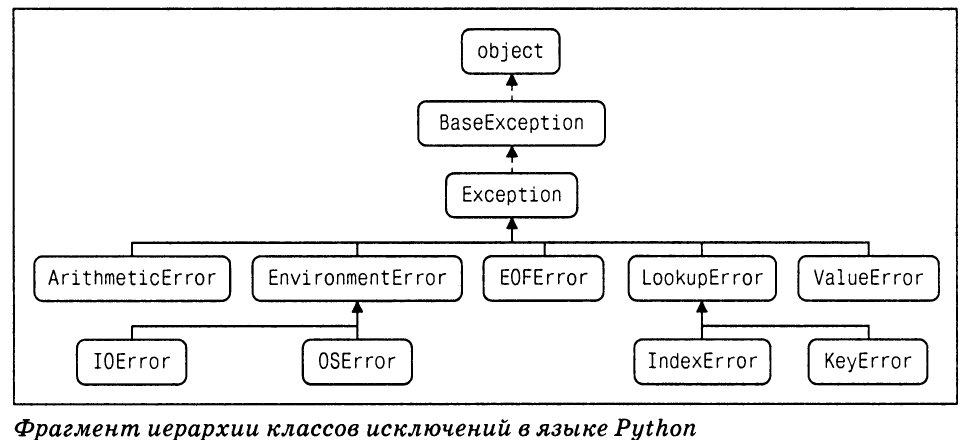
Iskljuchenija i funkcii

* Iskljuchenije v Python, tozhe javljajutsja objektami, kotorije pri preobrazovanii v stroke, vivoditsja na ekran kak tekstovoje soobshenije



Vazhno otmetitj, chto esli nichego iz napisannogo v try-suite ne budet narusheno, exception ne budut vizvani. Esli zhe proizoidet narushenije, programma poidet po vsem exception sverhu vniz, poka ne naidet sootvetstvujusheje. Esli budet naideno iskljuchenije, nizhe stojashije exception vipolneni ne budut(except vipolnjajetsja esli vozbuzhdennoje iskljuchenije sootvetstvujet iskljucheniju v bloke except ili vozuzhdennoje iskljuchenije javljajetsja podklassom klassa iskljuchenija v bloke except). Else vipolnitsja v tom sluchaje, esli blok try zakonchitsja bez vizova iskljuchenija. Finally budet vipolnen v ljubom sluchaje (budet naideno iskljuchenije ili net), no vipolnjajetsja on vsegda v samuju poslednuju ocheredtj(Esli ne budet naideno sootvetstvujusheje except i sisteme idja po steku vizovov, neobhodimo vivesti iskljuchenije i prekratitj rabotu programmi, dazhe v etom sluchaje, finally budet vipolnen pered iskljuchenijem). Object Exception\_group1 mozhet bitj sohranen v ssilochnuju peremennuju variable1 chtobi k nemu v bloke except mozhno bilo obratitsja

* Ierarhija klassov iskljuchenija:



* def func1:

Some code - objavlenije i realizacija funkcii func1. Def govorit o tom, chto ssilochnaja peremennaja func1 prisvaivajetsja object funkcii

* IndentationError – iskljuchenije ukazivajusheje na otsutstvije probelov v nuzhno meste
* TypeError – esli pri konvertacii, ne podderzhivajetsja dannij tip
* ValueError - Esli pri konvertacii podderzhivajutsja tipi no process po toj ili inoj prichine bil prervan
* KeyError – vizov ne sushestvujushego kljucha slovarja
* StopIteration – iterator objekta doshel do konca kollekcii
* AssertionError – vozbuzhdenije iskljuchenija cherez assert
* AtributeError – iskljuchenije pri oshibke obrashenija k atributu klassa
* assert a, ‘’Error occured’’ – esli znachenije peremennoj a istinno, iskljuchenije ne budet vozbuzhdeno. V protivnom sluchaje vozniknet iskljuchenije s tekstom Error occured
* raise – vozbuzhdajet iskljuchenije
* raise exception – vozbuzhdajet opredelennoje iskljuchenije
* vozmozhno tak zhe sozdavatj svoje sobstvennoje iskljuchenije cherez klass

class new\_exception(exception) : pass

zdesj bil sozdan klass new\_exception nasledujushij klass exception i v svojem tele ne imejushij nichego (new\_exception i estj nashe novoje iskljuchenije, kotoroje mi mozhem vizvatj cherez raise i perehvatitj cherez except)

* V python sushestvujet chetire tipa funkcij: Globaljnije, lokaljnije, lambda (anonimnije), metodi
* Argumenti po umolchaniju (neobjazateljnije argumenti) – V python funkcii, na etape svoei inicializacii, mogut imetj parametric so znachenijami.

def func1 (a,b,c=25,d=’bu’)

func1 imejet chetire argument. 2 poslednih javljajutsja argumentami po umolchaniju (eto znachit, pri vizove funckii, polzovatelju sovershenno neobjazateljno prisvaivatj kakije-to znachenija etim argumentam. Argument c i d uzhe budut imetj znachenija)

Vizvatj fucn1 mozhno

func1(20,30) – vizov func1 lish s dvumja pervimi argumentami (a i b)

a i b, shitajutsja objazateljnimi argumentami, a c id ne objazateljnimi.

Func1 mozhno vizivatj kak cherez pozicionnije argument(peredavatj neposredstvennije znachenija v toj posledovateljnosti, v kotoroj stojat parametri funckii pri ejo inicializacii)

func1(1,2,3,’a’) – peredajem pozicionnije argument dlja a,b,c,d

func1 mozhno vizivatj cherez imennovanije argument (peredavatj znachenije cherez prisvojenije k peremennoj).

Func1(1,b=39) – zdesj b javljajetsja imenovannim argumentum, a argument c i d imejut znachenije po umolchaniju

Vazhno ponimatj, chto vse imenovannije argument VSEGDA dolzhni stojatj posle pozicionnih.

Func1(1,b = 32, 4, d = ‘B’) – tak delatj nelzja!!!!! Eto svjazano s tem, chto pozicionnije argument privjazani k konkretnoj pozicii (k primeru argument a javljajetsja pervim argumentum v spiske argumentov funckii), a imenovannije argument pozicii ne imejut i opredeljajutsja lish po imeni. Poetomu, chto Python ponimal k kakomu argumentu prinadlezhitj znachenije, mi dolzhni ukazivatj pozicionnije argument pered imenovannimi

Primeri dopustimih obrashenij k func1:

Func1(1,2) – lish dva objazateljnih pozicionnih argument

Func1(1,2,3,’a’) – vse chetire argument javljajutsja pozicionnimi

Func1(1,b=3) – a pozicionnij, b imenovannij argument, c i d po umolchaniju

Func1(1,b=3,c = 5, d = ‘C’) – vse argument, za iskljuchenijem a, javljajutsja imenovannimi

Func1(a = 1,b=3,c = 5, d = ‘C’) – vse argumenti javljajutsja imenovannimi

Func1(1,2,d = ‘AO’) – blagodarja imenovannim argumentam, mi mozhem obrashatsja k ljubim neobjazateljnim argumentam, bez prisvojenija znachenije drugim argumentam (v nashem sluchaje c)

Func1(d=’A’, b = 4, a = 25, c = 7) – imennovanije argument mogut bitj objavleni v ljuboj posledovateljnosti

**ARGUMENTI PO UMOLCHANIJU, VSEGDA DOLZHNI STOJATJ POSLE OBJAZATELJNIH ARGUMENTOV!!!!!!!**

Func1 (a,b,c=25,d=’bu’) – praviljno

Func1(a = 32, b, c=25,d = ‘bu’) – vizovet oshibku, tak kak b stoit posle argument po umolchaniju

* Parametri funckii –

Func1(a,b,c,d,\*arg,e=45, f = 100)

a,b,c,d javljajutsja objazateljnimi argumentami, kak i arg. Arg javljajetsja kollekcijei i mozhet vmeshatj neopredelennoje kol-vo znachenij. e i f javljajutsja argumentami po umolchaniju. V dannom primere, vazhno otmetitj, chto operator ‘\*’ ne pozvoljajet v spiske argumentov, posle sebja obrashatsja k argumentam kak k pozicionnim.

Func1(a,b,\*,e,f) – zdesj k operatoram e i f mozhno obratitsja toljko cherez imenovannije argumenti

func1(1,2,e=56, f = 75) – eto praviljno

func1 (1,2,3,4) – vizovet oshibku (sushestvujet lish dva pozicionnih argument a obnaruzheno 4)

Esli pomestitj zvezdochku v nachale spiska argumentov funkcii, to mi naproch zapretim vizivatj funkciju cherez pozicionnije argument

Func1(\*,a,b,c,d)

Func1(a=2,b=3,c=4,d=5) – vot primer dopustimogo vizova funkcii

Stoit otmetitj, chto func1(\*arg,\*\*argv) – javljajetsja praviljnim, a func1(\*\*argv,\*arg) - nelzja

* Esli ispolzovatj zagruzku metodov iz modulja kak v sluchaje(from math import \*), nuzhno bitj predeljno ostorozhnim, poskoljku funkcii iz biblioteki math vizivajutsja bez imeni math pered soboj, to nazvanija etih metodov mozhet sovpadatj s imenem metodov kotorije opredelil programist v svojei programme. Primer:

From math import \*

Def sqrt():

Return “Intersting situation”

Print(Sqrt(25))

Print(Sqrt())

Esli mi zapustim takuju programme, budet vivedena oshibka, opoveshajushaja chto funkcija sqrt ne imejet argumentov, tem ne menee ej bil prisvojen argument 25.   
Python vipolnjajet kod strogo sverhu vniz. Do def sqrt() programa realizovivala sqrt() iz math modulja. Kak toljko interpretator doshel do def sqrt, metod sqrt pereopredelilsja i nazad izmenitsja uzhe ne mozhet. Esli bi sqrt(25) bilo bi zapisano ne posle def sqrt a pered, to programa bi vipolnilasj bez oshibok

* def some\_func(a,b,c,\*arg) – funkcija s tremja argumentami: a,b,c i \*arg eto tuple soderzhashij vse posledujushije zapisannije polzovatelem argument. Primer:

some\_func(1,2,3,4,5,6):

print(a) -> 1

print(b) -> 2

print(c) -> 3

print(arg) - > (4,5,6)

* def some\_func(a,b,c, \*\*arg) – funkcija s tremja argumentami:a,b,c i \*\*arg eto dictionary kotoromu nuzhno ukazivatj peremennije i ih znachenija chtobi sozdatj key/value pair. Primer:

somefunc(1,2,3, key1=4, key2 = 5, key3 = 6):

print(a) -> 1

print(b) -> 2

print(c) -> 3

print(arg) - > {“key1”: 4,”key2”: 5,”key3” : 6}

* def func\_1(arg1, arg2, arg3):

some code

list\_1 = [“A”, “B”, “C”]

func\_1(\*list\_1) – vizov funckii s tremja argumentami, ispolzuja spisok s operatorom avto-raspakovki. Dannij metod avtomaticheski raspredelit znachenija spiska po argumentam funkcii

* lambda - (anonimnaja funkcija) Primer:

lambda x : x % 3 ==0 to zhe samoje kak (v lambda ne nuzhen if)

def by\_three(x):

return x%3 == 0

* from collections import namedtuple – import metoda namedtuple iz biblioteki collections dlja imennovanih tuple
* DEKORATOR???!!!

[TOP](#TITLE)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Moduli

* import - komanda zagruzki opredelennoj biblioteki, soderzhashej klassi, metodi i polja

import sys - dlja sys.argv – peremennaja zapisivajushaja vvod s komandnoj stroki

argv[0] – vsegda soderzhit imja skripta, a vse ostaljnije jacheiki zapisivajut znachenija

import random - random.randint(1,6) – biblioteka sozdanija sluchainih chisel ot 1 do 6

* random.choice([“apple”,”banana”,”orange”]) – viberet sluchainoje znachenije iz spiska
* random() – sozdajet psevdo sluchainije chisla ot 0.0 do 1.0
* random.uniform(a,b) – generirujet sluchainij float chisla ot i do vkljuchiteljno.
* from math import \* - kljuchevije slova from govorjat chto iz modulja math vzjatj toljko opredelennije komponenti. \* znachit, chto vse soderzhimoje biblioteki budet zagruzheno. V takih sluchajah, funkcii math modulja pri vizove bolshe ne trebujut imja samogo modulja pered nimi.

(VSTRECHALOSJ V RAZDELE ISKLJUCHENIJA I FUNKCII)

**Esli ispolzovatj zagruzku metodov iz modulja kak v etom sluchaje, nuzhno bitj predeljno ostorozhnim, poskoljku funkcii iz biblioteki math vizivajutsja bez imeni math pered soboj, to nazvanija etih metodov mozhet sovpadatj s imenem metodov kotorije opredelil programist v svojei programme. Primer:**

**From math import \***

**Def sqrt():**

**Return “Intersting situation”**

**Print(Sqrt(25))**

**Print(Sqrt())**

**Esli mi zapustim takuju programme, budet vivedena oshibka, opoveshajushaja chto funkcija sqrt ne imejet argumentov, tem ne menee ej bil prisvojen argument 25.   
Python vipolnjajet kod strogo sverhu vniz. Do def sqrt() programa realizovivala sqrt() iz math modulja. Kak toljko interpretator doshel do def sqrt, metod sqrt pereopredelilsja i nazad izmenitsja uzhe ne mozhet. Esli bi sqrt(25) bilo bi zapisano ne posle def sqrt a pered, to programa bi vipolnilasj bez oshibok**

* From datetime import datetime – zagruzhajet datetime metod iz modulja datetime. Metod datetime pozvoljajet vivoditj datu(god,mesjac,denj) i vremja ()chasi,minuti,sekundi).

Var1 = datetime.now() – metod now() modulja datetime, vozvrashajet object soderzhashij nastojasheje vremja na kompjutere

Var1.year , Var1.month, Var1.day, Var1.hour, Var1.minute, Var1.second – vse eto javljajetsja atributami objekta Var1 posle prisvojenija emu vivoda metoda datetime.now()

* import math – sqrt funkcija vichisljajushaja kvadratnij korenj
* import mat.floor() – okruglenije do blizhaishego nizhnego celogo chisla
* import math.ceil() – okruglenije do blizhaishego verhnego celogo chisla
* import decimal.Decimal( n ) – modulj decimal I metod Decimal sozdajushij chisla, tochnostj kotorih posle zapjatoj ne ogranichena (no objekti dannogo tipa rabotajut medlenee tak kak siljneje nagruzhajut processor) Primenjatj toljko dlja celochislennih znachenij I strokovih no ne dlja float.
* import sys.float\_info.epsilon – modulj sys I object klassa float\_info soderzhit takije atributi, kak epsilon – ukazivajushije tochnostj(minimaljnoje uchitivajemoje chislo mashinoj – 2 \* 10^-16. Eto chislo mozhet menjatsja v zavisimosti ot kompjutera I sredi) tipa float.
* Import cmath – modulj soderzhashij metodi dlja raboti s kompleksnimi chislami
* Import re – biblioteka reguljarnih virazhenij (regular expression) – samij moshnij modulj obrabotki strok
* Import locale

locale.setlocale(locale.LC\_ALL, “”) – ispolzovanije lokaljnih nastroek

* Import sys – sistemnij modulj

version – atribut modulja. Vivodit versiju python

argv – peremennaja spisok, prinimajushaja znachenije s konsoli.

path.... – katalog path v kotorom hranjatsja vse puti k drugim katalogam

* Import os

Startfile(filelocation,) – zapuskajet ukazannij fail

listdir() – sozdajet spisok vseh katalogov v ukazannom puti. Po umolchaniju ispolzujetsja tekushij katalog

path.getsize() – vivodit razmer ukazannogo kataloga ili faila

walk(directory) – metod prohodit po vsem subdirektorijam direktorii directory i vozvrashajet object kotorij neobhodimo preobrazovatj v list (fail ii katalogi directory takzhe vivodjatsja). V kazhdoj jacheike list budet tuple s tremja znachenijami -> (tekushaja direktorija, list iz subdirektorij, list iz failov).

* Import locale
  + locale.setlocale(locale.LC\_ALL, “”) – ustanavlivajet iznachaljnuju kodirovku teksta v ukazannuju (v kavichkah stavitsja standard. “C” – oboznachenije nastroek po umolchaniju) OBJAZATELJNO VIZIVATJ METOD V NACHALE PROGRAMMI, CHTOBI USTANOVKI DEISTVOVALI NA VESJ POSLEDUSHIJ KOD DLJA INTERPRITATORA
* Import collections – biblioteka kollekcij
  + From collections import namedtuple – imenovannije kortezhi
  + From collections import defaultdict – modulj slovarej so znachenijem po umolchaniju. Esli k primeru, slovarj word = collections.defaultdict(int) javljajetsja slovarem so znachenijem po umolchaniju (znachenija kotorogo tipa int) to mi mozhem obratitsja k kljuchu, kotorogo v slovare net. Togda Python avtomaicheski sozdast etot kljuch i prisvojet emu znachenije nolj. Klass defaultdict javljajetsja podklassom klassa dict
* Import copy – modulj kopirovanija
  + From copy import copy – metod copy() - pozvoljajet sozdatj poverhnostnoje kopirovanije elementov kak izmenjajemih objektov tak i net
  + From copy import deepcopy – pozvoljajet sozdatj kopirovanije elementov ljuboj glubini
* Sobstvennije moduli – vozmozhno sozdavatj sobstvennije moduli
* import urllib.request (v python3) i urllib (v python 2) – biblioteka prednaznachennaja dlja otkritija url (HTTP).
  + urllib.request.urlretrieve(“url\_address”,”file\_destination”) – metod dlja kopirovanija html koda internet stranici i zapisi ego v fail na lokaljnoj mashine. Metod prinimajet dva argumenta: url\_address – ssilka na internet stranicu iz kotoroj neobhodimo skopirovatj html kod i file\_destination – fail v kotorij neobhodimo zapisatj rezultat.  
    V python 2 etot metod vizivajetsja cherez urllib.urlretrieve(a,b).  
    V python 2 etot metod vizivajetsja cherez urllib.request.urlretrieve(a,b)
* import winsound – biblioteka dlja raboti s audio failami v srede WINDOWS.
  + winsound.Beep(freq, duration) – metod vosproizvedenija audio s chastotoj ravnoj znacheniju argument freq i protjazhennostju ravnoj znacheniju peremennoj duration. Protjazhennostj izmerjajetsja v (ms). Chastota lezhit v diapazone ot 37 do 32767 gerts. Esli sistema ne mozhet vosproizvesti zvuk cherez kolonk to budet vozbuzhdeno iskljuchenije RuntimeError.
  + winsound.Playsound(“filename”,flag) – vosproizvodit audio file filename
* import webbrowser – biblioteka rabotajushaja s web browserom
  + open(url,new\_tab) – otkrivajet url ssilku v browsere v novoj vkladke, esli new\_tab raven 1

[TOP](#TITLE)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

OOP

* \_\_new\_\_() – metod vizivajemij pri sozdanii objekta (on sozdajet shablon objekta bez inicializacii. Opredelen v klasse object)
* \_\_init\_\_() – metod vizivajetsjaa posle \_\_new\_\_() dlja inicializacii objekta i prisvojenije ukazateljnoj peremennoj address objekta, esli takovaja imejetsja
* class Animal(object):

is\_alive = true

def \_\_init\_\_(self,name,age):

self.name = name

self.age = age

def description(self):

print(self.name)

print(self.age)

hippo = Animal(“John”, 102)

hippo.description()

kljuchevoje slovo class, objavljajet sam klass. Animal imja klassa. V kruglih skobkah ukazivajetsja nasledujemij klass, v dannom sluchaje object. Is\_alive javljajetsja globaljnoj peremennoj v klasse. Def \_\_init\_\_ javljajetsja konstruktorom, kotorij objazateljno dolzhen prinimatj odin argument (self) SELF NE JAVLJAJETSJA KLJUCHEVIM SLOVOM, NO NE GLASNO PRINJATA, CHTO ETO PEREMENNAJA NAZIVAJETSJA SELF. Self eto analog this v java. Cherez nego sozdajutsja atrubiti objekta. V sozdannije soboj metodi kak def description, neobhodimo perenositj argument self, esli neobhodimo obratitsja k atributam objekta. Hippo = Animal(“John”,102) – sozdanije objekta Hippo i peredacha konstruktoru dva argument: name i age. ARGUMENT SELF ZA PREDELAMI KLASSA NE ISPOLZUJETSJA. ETO MOZHNO ZAMETITJ KOGDA HIPPO VIZIVAJET METOD DESCRIPTION

* Nasledovanije – class Class(NewClass,NewClass2,... ,NewClassN) – klass Class nasledujet NewClass i t.d.
* super() – vizov metoda superklassa, kotorij bil pereorpredelen podklassom. Super avtomaticheski vstavljajet self v kachestve argumenta
* Class object – vse klassi v Python javljajutsja nasledujushimi ot klassa object
* Pereopredelenije metodov – esli klass nasledoval atributi drugogo klassa i v naslednike estj metod imejushij odinakovoje imja i odinakovije argument chto i u roditelja, to takoj metod budet pereopredelen na tot, chto nahoditsja v naslednike
* Kazhdomu atributu k kotoromu obrashajutsja vnutri klassa, nuzhno pripisivatj self. Tak zhe pri realizacii ljubogo metoda vnutri klassa, nuzhno peredavatj hotja bi odin argument - self
* \_\_init\_\_ - esli v klasse ne opisan nikakoj konstruktor init, togda ispolzujetsja init po umolchaniju
* \_\_repr\_\_(self) – korotkoje ot representation. Opredeljajet v kakom formate vivoditj soderzhimoje objekta klassa. Primer:

Class A(object):

Def \_\_init\_\_(self, a,b,c):

Self.a = a

Self.b = b

Self.c = c

Def \_\_repr\_\_(self):

Return (%d,%d,%d) % (self.a,self.b,self.c)

My\_obj = A(1,2,3)

Print(My\_obj) => (1,2,3)

* super(Imja\_dannogo\_klassa, self).metod(argumenti) – super pozvoljajet obratitsja k metodu superklassa, kotorij bil pereopredelen naslednikom

[TOP](#TITLE)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Faili

* f = open(“text.txt”,”w”) – otkritj fail text v rezhime zapisi “w”. I prisvoitj rezultat failovomu objektu f
* f = open(“text.txt”,”r”) – zdesj proishodit vse to zhe, chto i opisannoje vishe, toljko rezhim otkritija faila izmenen na chtenije “r”
* f = open(“text.txt”,”r+”) – zdesj proishodit vse to zhe, chto i opisannoje vishe, toljko rezhim otkritija faila izmenen na chtenije in a zapisj “r+”
* f.write(“something” + “\n”) – zapisivajet v uzhe otkritij fail, znachenije something, kotoroje zapishet informaciju v fail text (stoit pomnitj, chto chaste neobhodimo konvertirovatj znachenije something v string)
* f.read() – chitajet iz faila
* f.readline() – chitajet iz faila po strochno, kazhdij raz perekidivaja pointer na novuju strochku
* f.close() – zakrivajet fail (eto neobhodimo, chto zapisj v fail bila proizvedena korrektna. )

V processe zapisi, vsja informacija hranitsja v bufere. I do teh por, poka mashina ne budet uverena, chto v fail bolshe ne budet postupatj informacija, ona ne zapishet informaciju v fail. Metod close() govorit mashini o tom, chto v fail bolshe ne budet postupatj informacija, poetomu buffer smelo otpravljajet vse dannije v sam fail

* with open(“text.txt”,”w”) as f:

f.write(“Something”)

with pozvoljajet avtomaticheski otkritj failovij objekt s imenem f i ispolzovatj ego dlja svoih celej. Zakritije faila proshodit avtomaticheski pri vihode iz bloka with (s pomoshju vnutrennego \_\_exit\_\_() metoda) i ne trebujet propisivanije close() metoda

* closed – javljajetsja failovim atributom, kotorij glasit o tom, zakrit li dannij fail ili net. Closed hranit znachenije True ili False (f.closed => True)
* Failovije objekti imejut sobstvennije iteratori. Takim obrazom k nim mozhno obrashatsja cherez vstrojennij metod next(f) na prjamuju, bez predvariteljno, sozdannogo otdeljno, iteratora

[TOP](#TITLE)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Procesi i potoki

[TOP](#TITLE)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Seti

[TOP](#TITLE)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Bazi dannih

[TOP](#TITLE)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Reguljarnije virazhenija

[TOP](#TITLE)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Grafika

[TOP](#TITLE)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*