[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)

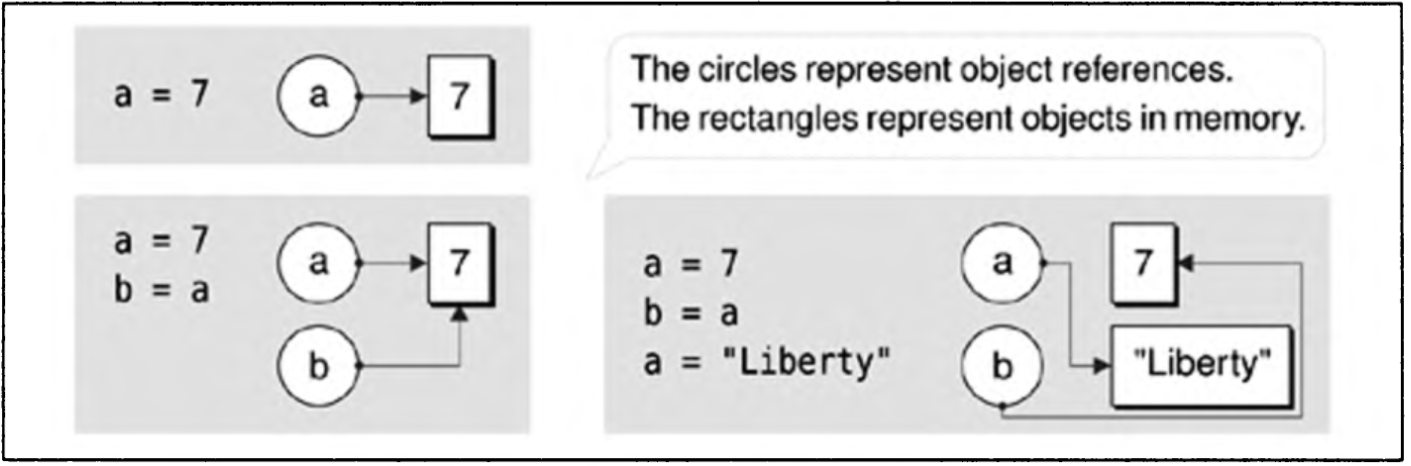
[Bazi dannih](#Bazi_Dannih)

[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

[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

* ord(Var) – otobrazhajet Unicode simvola
* chr(var) – otobrazhajet simvol dlja sootvetstvujushego koda
* bool – boolevi znachenija
* hex - shestnadcatirichnaja
* bin - dvoichnaja
* oct - vosjmerichnaja
* 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(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

* 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)

* 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)

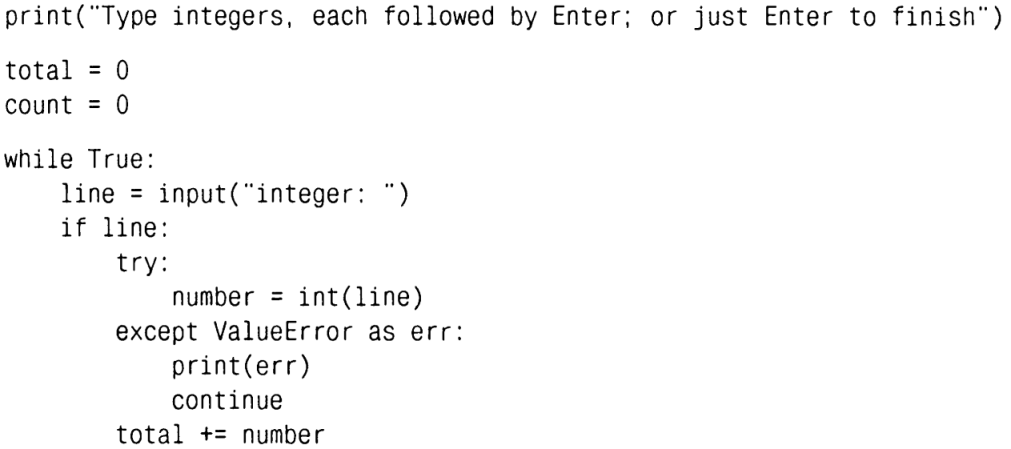
Funkcija zip pozvoljajet objedinjatj spiski i zakanchivajet cikl, kogda zakonchatsja elementi

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. Zdesj budet rezultat s plavajushej tochkoj
* 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
* \*\* - 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
* String\_a.upper() – prevrashajem stroke v bolshije simvoli
* 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

* String.strip() – udaljajet opredelennije simovli iz stroke (po umolchaniju probeli)
* 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
* help() – vizivajet help dlja konkretnogo argumenta

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

[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
* list\_a.append(var) - dobavljajet v list list\_a ssilku na objekt var (dopolnjajet spisok)
* 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.insert(3,”Some value”) – vstavljajet v opredelennuju poziciju spiska list\_a znachenije “Some value”
* 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
* sum(list\_a) – summirujet vse znachenije spiska
* min(list\_a) – nahodit minimaljnoje znachenije spiska
* max(list\_a) – nahodit maksimaljnoje znachenije spiska
* sorted(list\_a) – vozvrashajet sortirovannij list
* 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. 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
* 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
* dict1.items() – vozvrashajet vesj spisok key/value pair dictionary. Kazhdaja jacheika etogo spiska, sodzerzhit tuple znachenije key/value => [(key1, 1),(key2,2),(key3,3)]
* dict1.keys() – vivodit vse kljuchi slovarja
* dict1.values() – vivodit vse znachenija values slovarja
* print(dict[“key”]) – vivedet znachenije jacheiki s kljuchom key
* for a in dictionary:

print(a) - vivedet vse kljuchi dicitonary

* + 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
* (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**

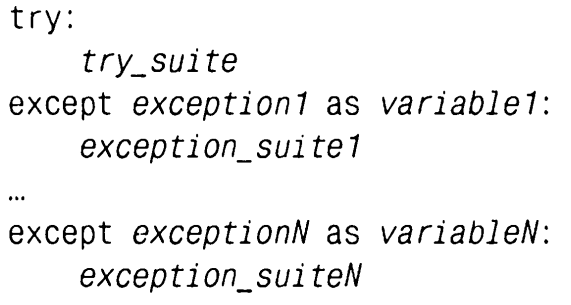
**na samom korotkom spiske. zip mozhet prinimatj i bolshe chem dva spiska. (a rinimajet list\_a, b beret list\_b)**

[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

* 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
* 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}

* 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

[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
* 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 sravnenija obichnih virazhenij

[TOP](#TITLE)

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

OOP

* 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) – klass Class nasledujet NewClass
* 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)

[TOP](#TITLE)

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

Procesi i potoki

[TOP](#TITLE)

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

Seti

[TOP](#TITLE)

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

Bazi dannih

[TOP](#TITLE)

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

Reguljarnije virazhenija

[TOP](#TITLE)

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

Grafika

[TOP](#TITLE)

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