Python 3 - Uvod u programski jezik Python

autor: Milenko Letic - https://programiranje.ba

e-mail: milenko.letic@programiranje.ba

verzija: 0.5

- dodane igre vjesala i potapanje brojeva

verzija: 0.4

- dodani primjeri za vjezbe (lecije 1 i 2)

O kursu

Kurs **Python 3 - Uvod u programski jezik Python** je dizajniran za sto jednostavnije ucenje. Namijenjen je pocetnicima i kao takav pokusava da zadrzi vasu paznju, postepenim dodavanjem manjih detalja kako bi se kreirala jasna slika sta je to Python programski jezik, gdje ga sve mozemo koristiti, kako nam moze biti od pomoci na dnevnoj bazi ali ono sto je najvaznije kako vam moze obezbijediti buducnost u polju informatike.

Nije tajna da je programer danas, kao i u proslosti, veoma cijenjeno zanimanje a da je potraznja na trzistu rada veoma velika za ovim kadrovima sto ce i ostati u buducem periodu.

Ono sto cete postici na kraju ovog kursa, i sto bi trebala biti mjera da li ste uspjesno usvojili znanje, jeste da cete biti u mogucnosti samostalno da kreirate osnovne programe, koji vam mogu biti od velike koristi prilikom automatizacije, ali ono najvaznije razumjecete principe programiranja i moci cete u potpunosti da se oslonite na svoje steceno znanje. Takodje, lakse cete moci da naucite druge programske jezike i da se upustite u ozbiljnije programarinje.

Citavo vrijeme cete biti vodjeni kroz kurs na vama je samo, da u par sati koliko kurs traje, pomno pratitie sve sto instruktor radi, odradjujete vjezbe nakon svake lekcije (rjesenja su uvijek data na pocetku sledece lekcije ili na kraju knjige koja dolazi sa kursom, takodje besplatno).

Primijeticete da nazivi Python fajlova imaju malo cudnu konvenciju, ali na nacin kako su fajlovi nazvani autor je olaksao organizaciju izvornog koda kao i referisanje studenata na odredjeni kod.

Prilikom kreiranja kursa i primjera mahom su koristeni karakteri iz poznate anime serije Zmajeva Kugla (Dragon Ball). Ponekad su dijelovi teksta uzeti iz pjesama Miladina Sobica, Tome Zdravkovica, Dubioze Kolektiv i Ramba Amadeusa. Takodje, postoje dijelovi teksta iz pjesmica za djecu, sve u cilju da se programiranje usvoji sto je lakse i prirodnije moguce ali i da bude zanimljivo tokom cijelog kursa.

tbd. Igrice vjesala i potapanje brodova. tbd. python mozete koristiti na svim poljima, automatizaciju, obradu ogromne kolicine podataka, web, igrice ...

Sva pitanja vezana za kurs mozete postaviti preko e-mail adrese *pitanja-python@programiranje.ba* ili na YouTube kanalu https://www.youtube.com/channel/UCSYrkPyht9PAXMhAbkGTbsQ (https://youtube.com/c/channel_name kada bude spreman tbd.)

Par rijeci o Python programskom jeziku

Python je programski jezik opste namjeni, dinamicki typed i interpretiran, objektno orjentisan programski jezik kreiran u kasnim 80-tim od strane Guido van Rossum.

Dizajn filozofija Python-a se svodi na jednostavnu citljivost, dakle u prvom planu ima za cilj sto lakse citanje i pisanje koda. Ovo se postize koristenjem white-space to deliniate code blocks umjesto vec dobro poznatog i ustaljenog nacina koristenja uglastih zagrada *{}* i tacka zareza *;*.

Kako pokrenuti Python

Generalno sav python kod se pokrece koristenjem interpretera. Najpopularniji i orginalni interpreter je CPython, zato sto je implementiran u C programskom jeziku. Takodje, postoji i par drugih interpretera, a mnogi od njih su implementirani u razlicitim jezicima od C-a, kao sto su Java ili C# (C sharp).

Najcesce koristen interpreter CPython, koristi automatski garabage koletor (sakupljac smeca 😃, kako bi obezbijedio nesmetano i efikasno upravljenje memorijom kompjutera. Python je siroko poznat po usvajanju ne tradicionalne, minimalne sintakse, bazirane na white space, i dizajnu koju tezi cistom i citljivom kodu.

Verzije Python-a

Prije samo par mjeseci (pisano Feb. 2020), ako biste htjeli instalirati Python na vasem racunaru, dosli biste u konfuznu situaciju, jer Python, za razliku od mnogih drugih programskih jezika, ima dvije glavne (major), ne kompatibilne verzije koje su podjednako u sirokoj upotrebi

Python verzije 2.7.3, released u 2012, je zadnja verzija popularnog Python-a 2 koji je released. Ova verzija je uglavnom u potpunosti kompatibilna unazad sa svim prethodnim verzijama.

Godine 2008, kreator, Guido van Rossum odlucio je da ocisti Python bazu (codebase) i overhall dosta drugih stvari u Python 2 koje mu se nisu svidjale, s toga je kreirao Python 3.

Python 3 je prihvatan ali veoma oprezno i polako na pocetku, najvise iz razloga sto nije kompatibilan unazad sa prethodnom verzijom Python 2, i zato je postojao ogroman eko-sistem biblioteka napisanih za Python 2 koje nece raditi sa novom verzijom Python-a 3.

Ovih dana Python 3 eco-sistem je uveliko pohvatao i izjednacio se sa prethodnom verzijom, sto nas dovodi do zakljucka da je Python 3 logicni izbor za sve nove developere koji planiraju uciti ovaj programski jezik. Python 3 je verzija koju cemo ujedno obraditi u ovom kursu.

Priprema radnog okruzenja

Izbor editora teksta i **Integrisanog razvojnog okruzenja** IDE (Integrated Development Environment)

Izbor tekst editora

Za pocetnike, se preporucuje koristenje nekog jednostavnog tekst editora kao Notpad++, Sublime, VisualStudio Code ...

Izbor Interisanog razvojnog okruzenja

Vecina programera odabere pisanje Python koda, koristenjem specijalnog integrisanog razvojnog okruzenja. Trenutno tri najistaknutija za Python su Eclipse, PyCharm i Netbeans.

Windows

- PyCharm installation (Win, Linux)
- Sublime installation, notpad ++
    - mi cemo koristiti PyCharm - IDE (Integrated Development Environment)
- python 2 (legacy), python 3 (future)
    - razlika u sintaksi
- podesavanje PyCharm-a i nas prvi program
    - promjena teme, odredisnog direktorija, velicine fonta i sl.
    - New -> Python File ...

Zdravo Svijete

Izvorni kod: kod-10_zdravo-svijete.py

```
print("Zdravo Svijete!")
```

Imamo dva nacina za pokretanje Python programa, ovo je preporuka kada koji koristiti:

- cmd, terminal - (ako nesto zelimo da provjerimo brzo i trenutno)
- direktno iz IDE-a (precice) - (kada pisemo vise linija koda)

Programiranje predstavlja davanje instrukcija kompjuteru (kroz programski jezik) i na osnovu ovih instrukcija kompjuter donosi odluke.

Izvorni kod: kod-11_crtanje-oblika.py

```
print("*")
print("**")
print("***")
print("****")
print("*****")
print("******")
```

- u ovom slucaju python ide liniju po liniju i izvrsava kod
- sta se desava u slucaju da zamijenimo prvu i zadnju liniju?

Komentarisanje koda

Komentare koristimo kada zelimo da zapisemo neki podsjetinik unutar koda, komentarisemo kod, objasnimo drugima i sebi sta odredjena linija koda radi. Praksa i preporuka je da se koristi simbol taraba (hash tag) #. Komentari su po default-u ignorisani u Python-u, preciznije ignorisani od strane Python interpretera.

Izvorni kod: kod-12_demonstracija-komentara.py

```
'''
Viselinijski komentar
'''
"""
Viselinijski komentar
"""
print("Komentari su korisni")
# print("Ova linija koda nece biti ispisana")
```

Zadaci za samostalni rad!

1. Napisati program koji ispisuje vase ime i prezime

```
Ime i Prezime
```

2. Napisati program koji crta pravougaonik oblika

```
__________
|          |
|          |
|__________|
```

3. Napisati program koji crta trougao oblika

```
    /\
   /  \
  /    \
 /      \
/________\
```

Promjenjive i tipovi podataka

Promjenjive ili varijable su osnovni objekti podataka kojima se manipulise u programu. Recimo da zelimo imati promjenjivu **ime_korisnika** koju mozemo koristiti kroz citav program i koja sadrzi vrijednost **Goku**. To bi smo mogli napisati kao:

```
ime_karaktera = "Goku"
```

Ovo citamo kao: Deklarisana je promjenjiva, ciji je naziv (identifikator) **ime_karaktera** a cija je inicijalna (pocetna) vrijednost **Goku**.

Takodje iz navedenog primjera mozemo zakljuciti da je promjenjiva, ciji je naziv **ime_karaktera**, tekstualnog tipa, niz karaktera ["G","o","k","u"] ili na engleskom tipa **string**.

Promjenjive, varijable (variables)

Programiranje se uglavnom svodi na obradu podataka, stim u vezi je potrebno pohraniti podatke i organizovati ih na najbolji moguci nacin. Varijabla ili promjenjiva predstavlja lokaciju u memoriji vaseg kompjutera i sluzi da pokaze na odredjenu vrijednost koju ta memorijska lokacija predstavlja.

Tri glavna faktora koji cine promjenjivu/varijablu jesu:

- naziv,
- operator i
- vrijednost varijable

```
naziv   |    operator     | vrijednost
          pridruzivanja

ime             =           "Goku"

godine          =             23
```

Gradjenje varijable

- naziv varijable ne smije poceti sa brojem
- naziv varijable moze poceti, malim, velikim slovima ili donjom crticom (_) nakon cega moze ici broj
- mala i velika slova se razlikuju (a != A)

Izvorni kod: kod-155_naziv-varijabli.py

```
ime = "Goku"
godine = 16
_nove_godine = 18
25_godine = 25 # ovde cemo dobiti gresku
a = 3
A = 4
print (ime)
print (godine)
print (_nove_godine)
print (25_godine) # ovde cemo dobiti gresku
print (a)
print (A)
```

Izvorni kod: kod-156_prakticna-primjena-varijabli.py

```
karakter_1 = "Son Goku"
karakter_2 = "Krilin"
godine = "17"

print("U dalekoj proslosti zivio je djecak po imenu Goku.")
print("Goku je imao 15 godina.")
print("Volio je upoznavati nove karaktere ")
print("i imao je najboljeg druga po imenu Krilin!")
print (" ")
print("U dalekoj proslosti zivio je djecak po imenu " + karakter_1)
print(karakter_1 + " je imao " + godine + " godina.")
print("Volio je upoznavati nove karaktere ")
print("i imao je najboljeg druga po imenu " + karakter_2)
# Veoma korisna funkcija type() sluzi nam da odredimo kog tipa je neka
promjenjiva tokom razvoja programa.
print(type(karakter_ime))
#godine = 17
print(type(godine))
```

Tipovi podataka

```
            Tipovi podataka                   |    python sintksa    |
objasnjenje
______________________________________________|______________________|______
_____________________________
Tekstualni (string - predstavlja niz znakova)|       string()       |
operacije nad znakovnim tipovima
                                              |                      |
podataka
______________________________________________|______________________|______
_____________________________
Brojevi cijeli, realni (integer, float)       |         int()        |int()
- pretvara u cijeli broj
                                              |                      |
(npr. 1,10,33)
                                              |        float()
|float() - pretvara u realni broj
                                              |                      |
(npr. 1.0, 3.14, 33.3333)
______________________________________________|______________________|______
_____________________________
Logicki tacno, netacno (boolean True/False)   |         bool()       |bool()
- operacije nad logickim
                                              |                      |
tipovima podataka
                                              |                      |
(True i False)
```

```
_____|_____|_____
_____
```

Mijenjanje tipova promjenjive (kastovanje)

Primjenjiva moze mijenjati tip kroz izvrsavanje programa, sto se jos naziva i kastovanje (casting).

Izvorni kod: kod-157_tipovi-podataka.py

```
karakter_ime = "Goku"
karakter_godine = 15
karakter_visina = 168.5
karakter_osobina_dobar = True
karakter_osobina_los = False

print(karakter_ime + " ima " + karakter_godine)
print(karakter_ime + " ce za godinu dana imati " + karakter_godine + 1)
print(karakter_ime + " je visok " + karakter_visina + "cm.")
print(karakter_ime + " ce za godinu dana biti visok " + (karakter_visina +
5) + " cm.")
print(karakter_ime + " je dobar karakter" + karakter_osobina_dobar)
print(karakter_ime + " je los karakter" + karakter_osobina_los)

karakter_ime = "Goku"
karakter_godine = 15
karakter_visina = 164.5
karakter_osobina_dobar = True
karakter_osobina_los = False

print(karakter_ime + " ima " + str(karakter_godine))
print(karakter_ime + " ce za godinu dana imati " + str(karakter_godine +
1))
print(karakter_ime + " je visok " + str(karakter_visina) + " cm.")
print(karakter_ime + " ce za godinu dana biti visok " + str(karakter_visina
+ 5) + " cm.")
print(karakter_ime + " je dobar karakter " + str(karakter_osobina_dobar))
print(karakter_ime + " je los karakter " + str(karakter_osobina_los))
```

Rad sa stringovima

Stringovi predstavljaju niz karaktera (velika/mala slova, brojevi, znakovi interpunkcije, specijalni znakovi, ...), koji sluzi za skladistenje i rad sa tekstualni podacima. Mozemo reci da su stringovi uredjeni i smisleni niz karaktera.

Jednostavan primjer kreiranja stringa

Izvorni kod: kod-165_rad_sa_stringovima.py

```
print("programiranje.ba besplatni online kursevi")

# escape karakter \
print("programiranje.ba \n besplatni online kursevi")

sajt_naziv = "programiranje.ba"
sajt_slogan = " besplatni online kursevi"

# jos neki primjeri kreiranja string-a (kao sto smo imali slucaj sa
viselinijskim komentarom)
sajt_naziv = 'https://programiranje.ba'
sajt_slogan = """besplatni online kursevi
                  za sve """
```

Funkcije nad stringovima

Izvorni kod: kod-166_rad_sa_stringovima.py

```
print("programiranje.ba besplatni online kursevi")

# escape karakter \
print("programiranje.ba \n besplatni online kursevi")
# takodje se koristi kada trebamo ispisati specijalne karaktere
#print("\")
print("\\")

sajt_naziv = "programiranje.ba"
sajt_slogan = " besplatni online kursevi"
print(sajt_naziv)
print(sajt_slogan)
print(sajt_naziv + sajt_slogan)
print(sajt_naziv.upper() + sajt_slogan.upper())
print(sajt_naziv.isupper())
print(len(sajt_naziv))
print(sajt_naziv * 3)

# index stringa pocinje na poziciji 0
print(sajt_naziv[4])
print(sajt_naziv[-4])
print(sajt_naziv[1:4])
# index funkcija i proslijedjivanje parametara
print(sajt_naziv.index('g'))
print(sajt_naziv.index('mira'))
#print(sajt_naziv.index('h'))
# find funkcija i proslijedjivanje parametara
print(sajt_naziv.find('g'))
print(sajt_naziv.find('mira'))
# razlika izmedju find i index, u slucaju da ne postoji trazeni patern
# index() vraca gresku, dok find vraca -1

# replace funkija
```

```
print(sajt_slogan.replace("kursevi","tutoriali").upper())

print("{1}, {0}".format(sajt_naziv, sajt_slogan))
print(f"{sajt_naziv} {sajt_naziv}")

# jos neki primjeri funkcija nad stringovima
sajt_naziv = "programiranje.ba"
sajt_godina = "2020"
sajt_kratki_slogan = "kursevi"

print(sajt_naziv.isalpha())
print(sajt_godina.isdigit())
print(sajt_kratki_slogan.isalpha())
### funkcije chr() i ord() kasnije potrebni za cezarovu sifru
# funkcija ord() daje cjelobrojnu vrijednost karaktera prema ASCII tabli
print(ord('A'))
print(ord('a'))
print(ord('b'))
print(ord('z'))

# funkcija chr() konvertuje cjelobrojnu vrijednost u odgovarajuci karakter
print(chr(64))
print(chr(33))
print(chr(97))
print(chr(100))

# kombinacija chr() i ord()
print(ord('b') + 3)
print(chr(ord('b') + 3))
```

Rad sa brojevima

brojevi, rad sa brojevima, funkcije nad brojevima. Osnovne aritmeticke operacije koje mozemo vrsiti u Python-u su

Operacija	Naziv operacije	Primjer	Rezultat
+	Sabiranje	print(1 + 1)	2
-	Oduzimanje	print(6 - 5)	1
*	Mnozenje	print(4 * 3)	12
/	Dijeljenje	print(4 / 2)	2
%	Ostatak pri dijeljenju (moduo)	print(5 % 2)	1
**	Potenciranje	print(2 ** 3)	8

Izvorni kod: kod-185_rad-sa-brojevima.py

```
print(3)
print(-4)
print(3.333)
print(7+3)
print(7+3.333)
print(8/4)
print(2*4)
print (2*(4+5))
# modulo , ostatak 9 mod 4
print(9%4)

moj_broj = 13
print(moj_broj)
print("Moj omiljeni broj" + str(moj_broj))

# math funkcije, funkcije su vec pripremljen kod koji odradjuje posao za
nas
broj = -2
print(abs(broj))
print(pow(4,2))
print(max(5,10))
print(min(3,6))
print(round(3.3333))
print(round(3.6))
print(round(3.5))

# import math modul (import math funkcije vise kada budemo pricali o
modulima)
from math import *
print(floor (3.6))
print(ceil(3.6))
print(sqrt(9))
```

Ulaz/upis podataka, prihvatanje podataka od korisnika ili interakcija sa programom

Slozicete se da je programiranje dosadno, ako nemamo interakciju, nekakav vid komunikacije sa nasim programom. Kako bi omogucili interakciju sa programom, Python nam na raspolaganje nudi funkciju input().

Izvorni kod: kod-190_interakcija-sa-korisnikom.py

```
input()
# hej ti, cekam da uneses neku informaciju podatak, naravno korisnik nije
siguran sta se desava

input("Unesite vase ime: ")
# aha ovo sad vec ima smisla
```

```
# <naziv varijable> <tip podatka>
korisnik_ime = input("Unesite vase ime: ")
# naravno, posto nam je korisnicki unos vazan mi zelimo sacuvati isti taj
unos u neku varijablu
# kako bi smo kasnije mogli koristiti
print("Zdravo, " + korisnik_ime + " dobrodosli.")

korisnik_ime = input("Unesite vase ime: ")
korisnik_godine = input("Unesite vase godine: ")
print("Zdravo, " + korisnik_ime + ". Vi imate " + korisnik_godine + "
godina.")

# vjezba ispisati preghodni program koristeci funkciju format
```

Izvorni kod: kod-191_interaktivni_karakter_program.py

```
print("U dalekoj proslosti zivio je djecak po imenu Goku.")
print("Goku je imao 15 godina.")
print("Volio je upoznavati nove karaktere ")
print("i imao je najboljeg druga po imenu Krilin!")

karakter_ime = input("Unesite ime karaktera: ")
karakter_godine = input("Unesite godine karaktera: ")
karakter_prijatelj = input("Unesite ime najboljeg prijatelja: ")

print(f"U dalekoj proslosti zivio je djecak po imenu {karakter_ime}.")
print(f"{karakter_ime} je imao {karakter_godine} godina.")
print(f"Volio je upoznavati nove karaktere ")
print(f"i imao je najboljeg druga po imenu {karakter_prijatelj}!")
```

Izvorni kod: kod-192_osnovni_kalkulator.py

```
broj_1 = input("Unesite prvi broj: ")
broj_2 = input("Unesite drugi broj: ")
rezultat = broj1 + broj2

print(rezultat)

broj_1 = input("Unesite prvi broj: ")
broj_2 = input("Unesite drugi broj: ")
rezultat = float(broj_1) + float(broj_2)

print(rezultat)
```

Zadaci za samostalni rad!

1. Napisati program koji trazi da upisete vase ime, a on automatski ispisuje vase ime pet puta, sa razmakom izmedju imena.

```
Unesite vase ime: Goku
Goku Goku Goku Goku Goku
```

2. Napisati program koji trazi da unesete dvije rijeci, a zatim ih ispise u istom redu sa razmakom od tri space karaktera izmedju izmedju.

```
Unesite prvu rijec: Dobar
Unesite drugu rijec: Dan
Dobar   Dan
```

3. Napisati program koji racuna aritmeticku sredinu za tri unesena broja. Pomoc: Aritmeticka sredina za a,b,c se izracunava po formuli (a + b + c)/3

```
Unesite prvi broj: 1
Unesite drugi broj: 2
Unesite treci broj: 3
Aritmeticka sredina je: 2.0
```

4. Napisati program koji za uneseni karakter sa tastaure ispisuje vrijednost iz ASCII tabele

```
Unesite znak sa tastature: }
Vrijednost znaka '}' u ASCII tabeli je 125
```

Liste - []

Liste predstavljaju niz objekata, gdje svaki clan liste ima svoj indeks. Ovi clanovi se nazivaju elementima lista. Slicne su stringovima, s tim da svaki elemnent liste moze biti razlicitog tipa. Elementi liste su smjesteni u uglaste zagrade *[]* i razdvojeni zarezom ,.

```
lista   |  [ 1, 4, 6.33, 10, "Goku" ]
indeks  |    0  1     2   3    4
```

Rad sa listama, nam pomaze pri organizaciji i boljem pracenju toka podataka.

Izvorni kod: kod-210_rad-sa-listama.py

```
prazna_lista = []
print(prazna_lista)
```

```
pritn(type(prazna_lista))

karakteri = ["Goku", "Krilin", "Bulma", "Chi-Chi", "Yamcha", "Picolo"]
print(karakteri)

##### mozete smjestiti stringove , brojeve, boolean u liste
##### referenciranje po indexu, ako zelimo pristupiti elementu unutar liste
print(karakteri[0])
print(karakteri[4])

print(karakteri[-1])
print(karakteri[-2])

print(karakteri[1:])
print(karakteri[2:4])
print(karakteri[2:-2])

##### izmjena elemenata u listi
karakteri[4] = "Master Roshi"
print(karakteri[4])
print(karakteri)
```

funkcije nad listama

Izvorni kod: kod-211_rad-sa_listama.py

```
karakteri = ["Goku", "Krilin", "Bulma", "Chi-Chi", "Yamcha", "Picolo"]
print(karakteri)
loto_brojevi = [4, 7, 3, 10, 32]
print(loto_brojevi)

karakteri.extends(loto_brojevi)
print(karakteri)

karakteri.append("Majin Buu")

print(karakteri)

karakteri.insert(2, "Vegeta")
print(karakteri)

karakteri.remove("Majin Buu")
print(karakteri)

del(karakteri[3]) # funkcija del
print(karakteri)

izbrisan_karakter = karakteri.pop(4) # metoda pop, u slucaju da hocemo da
sacuvamo izbrisani element
print(karakteri)
```

```
karakteri.clear("")
print(karakteri)

##### provjeri da li je odredjeni element u listi

karakteri = ["Goku", "Krilin", "Bulma", "Chi-Chi", "Yamcha", "Picolo",
"Bulma"]
print(karakteri)

print(len(karakteri)) # funkcija len

print(karakteri.index("Bulma")) # metoda index

print(karakteri.count("Bulma")) # metoda count, prebrojavanje koliko se
trazeni element pojavljuje u listi

karakteri.sort() # metoda sort
print(karakteri)

karakteri.reverse() # metoda reverse
print(karakteri)

karakteri_novi = karakteri.copy() # metoda copy

print(karakteri_novi)

karakteri_novi.sort()
print(karakteri_novi)

print list(string_karakter)
# funkcije min i max, vracaju najmanji ili najveci element liste
respektivno
karakteri = ["Goku", "Krilin", "Bulma", "Chi-Chi", "Yamcha", "Picolo",
"Bulma"]
print(karakteri)
print min(karakteri) # funkcija min
print max(karakteri) # funkcija max

# konverzija stringa u listu
string_karakter = "Goku"
type(string_karakter)
print(string_karakter)
```

Tuples - torke (tip podaktovne strukture, veoma slican listama) - ()

Tuples predstavlja niz nepromjenjivih clanova. Clanovi unutar tuple-a mogu biti istih ili razlicitih tipova. Tuple definisemo nabrajanjem objekata odvojenih zarezom, cak i ako je u pitanju jedan jedini clan moramo imati zaraz, u suprotnom se gubi osobina tuple-a.

Razlikuju se od liste po tome sto su **nepromjenjive (immutable -ne mogu se mijenjati).**

Tuple mozemo prevosti kao torke, a izraz dolazi iz matematike od pojma *n-torka* (eng.tuple) koji predstavlja konacni niz (poznat kao uredjena lista) od n objekata, od kojih je svaki specifincnog tipa.

Clanovi torke su smjesteni u obicne zagrade *()* i razdvojeni zarezom ,. Clanovi torke mogu biti same torke.

Izvorni kod: kod-220_rad-sa-torkama.py

```
karakteri = (1,2,3,4,"a","d","-")
print(karakteri)
type(karakteri)

karakteri = (1,)
print(karakteri)
type(karakteri)

karakteri = (1)
print(karakteri)
type(karakteri)

### list of tuples
koordinate = [(4,5), (6,3), (7.4)]
print(type(koordinate))
print(len(koordinate))

## tuple su nepromjenjive strukture
koordinate = (4, 5)
print(type(koordinate)) # funkijca type()
print (len(koordinate)) # funkcija len()
print(koordinate.index(5)) # metod index
print(koordinate[0])
print(koordinate[1])

koordinate[1] = 10 # dobicemo gresku

### pristupanje elementima torki slicno je kao i kad pristupamo listama
karakteri = (1,2,3,4,"a","d","-") # pristupanje elementima tuple-a
print(karakteri[1])
print(karakteri[2:4])
print(karakteri[-2:]
print(karakteri[::2]) # pristupanje svakom drugom elementu

# konverzija drugih tipova u tuple
karakter_ime = "Goku"
type(karakter_ime)
karakter_godine = 15
type(karakter_godine)

karakter_ime = tuple(karakter_ime)
type(karakter_ime)
print(karakter_ime)
karakter_godine = tuple(karakter_godine)
karakter_godine = tuple(str(karakter_godine))
type(karakter_godine)
```

```
print(karakter_godine)

karakter = karakter_ime + karakter_godine
type(karakter)
print(karakter)

# brisanje tuple-a
del(karakter[3]) # brisanje elemenata tuple-a nije moguce
del(karakter)
```

Rijecnici - Dictionaries - { }

Rijecnici su tipovi podataka, opet slicni listama, ali za razliku od listi indeksiranje se obavlja kljucevima.

Za lakse razumijevanje ih mozemo uporediti sa klasicnim rijecnikom za prevodjenje rijeci sa jednog jezika na drugi, gdje imamo strukturu strana rijec na lijevoj strani i detaljno objasnjenje rijeci na desnoj strani. Ako navedenu analogiju primijenimo rijecnicima, kao tipovima podataka u Python-u, onda rijec predstavlja kljuc (key), dok detaljno objasnjenj predstavlja vrijednost (value).

Elementi rijecnika su smjesteni u viticaste zagrade **{ }** a parovi elemenata su razdvojeni zarezom **,**.

`text{kljuc:vrijednost} ({key:value})`

Bitno je napomenti da kljuc (key), mora biti jedinstven, ne mozemo imati dva ista kljuca.

```
{"kljuc_1:vrijednost_1", "kljuc_2:vrijednost_2", "kljuc_3:vrijednost_1"} -
ispravno
{"kljuc_1:vrijednost_1", "kljuc_1:vrijednost_2", "kljuc_3:vrijednost_1"} -
nije ispravno
```

Izvorni kod: kod-230_rad-sa-rijecnicima

```
karakteri={} # prazan rijecnik
print(karakteri)

karakteri_osobine={"Goku":"Vegeta", "Picolo":"Namek", "Krilin":"Zemlja"}
print(karakteri_planete)
print(type(karakteri_planete))

karakteri_planete["Bulma"]="Zemlja"
print(karakteri_planete)
karakteri_planete["Goku"]="Namek" # prepisace trenutnu vrijednost ako
postoji

del(karakteri_planete["Goku"]) # brisanje elementa

print(len(karakteri_planete)) # primijetimo da se broje parovi

print(karakteri_planete.keys()) # metoda keys() nad rijecnicima, ispisuje
```

```
sve kljuceve (keys), nema argumente
print(karakteri_planete.values()) # metoda values() ispisuje vrijednosti
elementa, nema argumente
print(karakteri_planete.items()) # metoda items() ispisuje kljuc:
vrijednost elementa, nema argumente

# metode get i setdefault
karakteri_planete={'Goku':'Vegeta', 'Krilin':'Zemlja', 'Bulma':''}
print(karakteri_planete.get('Goku','Karakter ne postoji u bazi'))
print(karakteri_planete.get('Pikolo','Karakter ne postoji u bazi')) #
metoda get() nad rijecnikom vrsi pretragu po zadatom kljucu, u slucaju da
kljuc ne postoji vraca default-nu vrijednost, vrijednost koja je
proslijedjena kao drugi parametar

print(karakteri_planete.setdefault('Pikolo','Karakter nema definisanu
planetu')) # kljuc ce biti kreiran u slucaju da ne postoji, a vrijednost ce
biti podesena na vrijdnost drugog proslijednjenog parametra
print(karakteri_planete.setdefault('Bulma','Zemlja')) # obzirom da kljuc
postoji, nece doci do promjena

# metode pop i update
karakteri_planete={'Goku':'Vegeta', 'Krilin':'Zemlja', 'Bulma':'Zemlja'}
obrisan_karakter=(karakteri_planete.pop('Goku')) # pop() metoda prilikom
brisanja key:value, zadrzava vrijednost (value)
print (karakteri_planete)
print (obrisan_karakter)

# spajanje rijecnika mozemo izvesti upotrebom metode update()
karakteri_1_planete={'Goku':'Vegeta', 'Krilin':'Zemlja', 'Bulma':'Zemlja'}
karakteri_2_planete={'Chi-Chi':'Zemlja', 'Vegeta':'Vegeta',
'Bulma':'Namek'}
karakteri_1_planete.update(karakteri_2_planete) # update() metod nad
rijecnicima prosiruje prvi rijecnik vrijednostima iz drugog, u slucaju da
imamo dva ista kljuca, kljuc iz prvog rijecnika bice zamijenjena kljucem iz
drugog rijecnika
karakteri_1_planete={'Goku':'Vegeta', 'Krilin':'Zemlja', 'Bulma':'Zemlja'}
karakteri_2_planete={'Chi-Chi':'Zemlja', 'Vegeta':'Vegeta',
'Bulma':'Namek'}
karakteri_2_planete.update(karakteri_1_planete)
```

Izvorni kod: kod-231_konverzija_mjeseci.py

```
# recimo da zelimo konvertovati kratke nazive mjeseca u standardne
# Jan -> Januar
# Mar -> Mart

konverzijaMjeseci = {
    "Jan": "Januar",
    "Feb": "Februar",
    "Mar": "Mart",
    "Apr": "April",
    "Maj": "Maj",
```

```
    "Jun": "Juni",
    "Jul": "Juli",
    "Avg": "Avgust",
    "Sep": "Septermbar",
    "Okt": "Oktobar",
    "Nov": "Novembar",
    "Dec": "Decembar"
}

print(konverzijaMjeseci["Jan"])
print(konverzijaMjeseci["Feb"])
print(konverzijaMjeseci["Mar"])

print(konverzijaMjeseci.get("Jan"))
print(konverzijaMjeseci.get("Dec"))
print(konverzijaMjeseci.get("Dese","Nije validan kljuc")) # ako koristimo
get necemo dobit gresku vec empty
```

Funkcije

Skup naredbi koje po pozivu izvrsavaju odredjene zadatke. Funkcije sluze takodje za bolju organizaciju koda.

Funkcije se definisu pomocu kljucne rijeci **def**, kad god se pojavi def na pocetku linije, python zna da korisnik zeli kreirati funkciju i stim u vezi se i ponasa.

Sintaksa funkcije:

```
def naziv_funkcije(parametri): # parametri su opcioni, ali ako postoje
moraju biti definisani/proslijedjeni
  blok naredbi # argumenti (poznat jos kao tijelo funkcije)

naziv_funkcije(argumenti) # poziv funkcije
```

Iz sintakse mozemo zakljuciti da je naziv_funkcije identifikator kojim pozivamo funkciju, dok parametri sluze da bi se definisale vrijednosti koje se mogu proslijediti kao parametri naredbama unutar tijela funkcije.

Izvorni kod: kod-255_rad-sa-funkcijama.py

```
def zdravo_svijete(): # funkcija bez parametara
  print("Zdravo Svijete.")

# moramo pozvati funkciju ako zelimo da je izvrsimo
zdravo_svijete()

def zdravo_svijete():
  print("Zdravo Svijete.")

print("Prije funkcije")
zdravo_svijete()
```

```
print("Nakon funkcije")

# prosledjivanje parametara funkciji

def zdravo_svijete(ime):
  print("Zdravo " + ime)

# moramo pozvati funkciju ako zelimo da je izvrsimo
zdravo_svijete("Goku")
zdravo_svijete("Krilin")

def zdravo_svijete(ime,godine):
  print("Zdravo " + ime + " vi imate " + str(godine))

# moramo pozvati funkciju ako zelimo da je izvrsimo
zdravo_svijete("Goku", "15")
zdravo_svijete("Krilin", "16")

# primjer funkcije sa korisnickim unosom
korisnik_ime = input("Unesite ime : ")
def pozdrav(ime):
    print ("Zdravo " + ime)

pozdrav(korisnik_ime)
```

Naredba return (return direktiva)

Kada zelimo dobiti povratnu informaciju iz funkcije koristimo naredbu **return**. Sa ovom informacijom mozemo nastaviti manipulaciju kroz daljni dio koda. Naredba return se moze pojaviti samo unutar tijela funkcije. Takodje kada zelimo da funkcije mogu medjusobno komunicirati, razmjenjivati informacije koristimo return naredbu.

Izvorni kod: kod-260_kub-broja.py

```
def kub(broj):
  broj * broj * broj

cub(3)

print(kub(3))

####
def kub(broj):
  return broj * broj * broj # aha zelim vratiti informaciju ko god da je
pozvao funkciju

print(kub(3))

####
```

```
def kub(broj):
  return broj * broj * broj

rezultat = kub(3) # sacuvaj vrijednost koju si dobio od funkcije, ne i samu funkciju
print(rezultat)

####
def kub(broj):
  print("stampaj prije return direktive")
  return broj * broj * broj
  print("stampaj nakon return direktive")

print(cub(3))
```

Funkcija ne mijenja sadrzaj promjenjive

Izvorni kod: kod-261_rad-sa-funkcijama.py

```
def brojac(broj):
  broj = broj + 1
  return broj

broj=3
brojac(broj) # vrijednost koju promjenjiva pokazuje, ali ne i samu promjenjivu, sto obezbjedjje da funkcija ne moze mijenjati promjenjivu, samo kopiju vrijednosti koja je proslijedjena
print(broj) #

# funkcije unutar sebe mogu imati lokalne (local) i globalne (global) promjenjive
# ako lokalna i globalna promjenjiva unutar funkcije nose isti naziv, Pyhon ce koristiti lokalnu
# lokalne promjenjive su definisane po default-u ako se ne navede drugacije
def brojac():
    global broj
    broj = broj + 1
    return broj

broj = 3
brojac()
print(broj)

# nepoznati broj argumenta
# u slucaju da nismo sigurni koji je tacno broj argumenate koje zelimo proslijediti funkciji
# parametru funkcije dodamo * i time postizemo n broj elemenata koji mozemo proslijediti funkciji

def srecni_brojevi(*brojevi):
    print("Loto brojevi za ovu sedmicu: " + str(brojevi))
```

```
srecni_brojevi(8,13,22,12,54)

# poziv funkcije iz druge funkcije
def kub(broj):
  return broj * broj * broj

def ispis_kubnog_broja(broj):
  kubni_broj = kub(broj)
  print("Kub broja " + str(broj) + " je " + str(kubni_broj))

ispis_kubnog_broja(2)

# docstring - dokumentacijski string,  predstavlja dokumentaciju same
funkcije koja moze da se pozove funkcijom
# help()
def kub(broj):
  '''Funkcije izracunava kub unesenog broja, po formuli broj * broj * broj.
             Primjer koristenja funkcije: print(kub(2))
                                          daje vrijednost 8
  '''
  return broj * broj * broj

print(kub(2))
help(kub)

# neke od standardnih, ugradjenih, funkcija koje dolaze sa Python-om
print(abs(-1))
print(len("Goku"))
print(max(2,3))
print(min(2,3))
print(str(2))
print(type(kub))
print(type(print))
print(type(type))
```

Naredbe za kontrolu toka (if, elif, else)

Ako zelimo da donosenje odluke prepustimo nasem programu, na osnovu uslova koji se moraju ispuniti, a samim tim krairamo nas program pametnijim, uvescemo novi uslov if (naredbu if), koja se jos zove i naredba kontrole toka. Ukoliko je uslov ispunjen (Tacan - True), izvrsava se naredba ili blok naredbi pod tim uslovom, u suprotnom izvrsava se drugi blok naredbi ili se nastavlja ispitivanje.

Primjer iz realnog zivota

```
Probudio sam se i oprao zube
ako sam gladan
  trebam doruckovati

Trebam ici u vani
ako je oblacno
```

```
  ponijecu kisobran
u suprotnom
  ponijecu suncane naocare

U restoranu
ako zelim meso
  narucicu stejk
ako zelim pastu
  narucicu spagete
u suprotnom
  narucicu salatu
```

Relacijski operatori, operatori poredjenja (>,<, >=, <=, ==, !=)

Operacija	Naziv operacije	Primjer ako su : a=3 b=2 i c=3	Rezultat
>	Vece	print(a > b)	Tacno (True)
<	Manje	print(a < b)	Nije Tacno (False)
>=	Vece ili jednako	print(a >= b) print(a >= c)	Tacno (True) Tacno (True)
<=	Manje ili jednako	print(a <= b) print(a <= c)	Nije Tacno (False) Tacno (True)
==	Jednako	print(a == b) print(a == c)	Nije Tacno (False) Tacno (True)
!=	Nije jednako Razlicito	print(a != b) print(a != c)	Tacno (True) Nije Tacno (False)

Logicki operatori (and, or i not) ili bitski operatori

Operacija	Naziv operacije	Primjer ako su : a=3 b=2 i dobar=True	Rezultat

and	i	if(a > 4 and b < 3):
		  print("Zdravo")
		else:
		  print("Dovidjenja")
			Dovidjenja

or	ili	if(a > 4 or b < 3):
		  print("Zdravo")
		else:
			Zdravo

```
            |                        |  print("Dovidjenja") |
____________|________________________|_____________________|_______________
     not    |      ne, nije          |if(not(dobar)):      |
            |                        |  print("Nije Dobro") |
            |                        |else:                |  Dobro je
            |                        |  print("Dobro je")  |
____________|________________________|_____________________|_______________
```

```
dobar = True

if dobar:
  print("Goku je dobar karakter")

####
dobar = True
zabavan = False

if dobar or smijesan:
  print("Goku je dobar karakter") # koliko god koda mozete smjestiti ovde
  print("Freza nije je zabavan")
else:
  print("Freza je los karakter")

###
dobar = True
zabavan = True

if dobar and smijesan:
  print("Goku je dobar karakter") # koliko god koda mozete smjestiti ovde
  print("Krilin je zabavan")
else:
  print("Freza je los karakter")

####
dobar = True
zabavan = False

if dobar or smijesan:
  print("Goku je dobar karakter") # koliko god koda mozete smjestiti ovde
elseif dobar and not(zabavan):
  print("Freza nije je zabavan")
else:
  print("Freza je los karakter")
```

maksimalan_broj.py

```
def maksimalan_broj(broj_1, broj_2, broj_3):
  if broj_1  >= broj_2 and broj_1 >= broj_3:
    return broj_1
```

```
    elif broj_2 >= broj_1 and broj_2 >=broj_3:
      return broj_2
    else:
      return broj_3

  print(maksimalan_broj(7, 8, 9))
```

kalkulator_nadogradjena_verzija.py

Ovo je bio osnovni_kalkulator.py

```
broj_1 = input("Unesite prvi broj: ")
broj_2 = input("Unesite drugi broj: ")
rezultat = broj1 + broj2

print(rezultat)

broj_1 = input("Unesite prvi broj: ")
broj_2 = input("Unesite drugi broj: ")
rezultat = float(broj_1) + float(broj_2)

print(rezultat)
```

kalkulator_nadogradjena_verzija.py

```
broj_1 = float(input("Unesite prvi broj: "))
broj_2 = float(input("Unesite drugi broj: "))
operator = input("Unesite operator: [+, -, /, *] ")

if operator == "+":
  print(broj_1 + broj_2)
elif operator == "-":
  print(broj_1 - broj_2)
elif operator == "/":
  print(broj_1 / broj_2)
elif operator == "*":
  print(broj_1 * broj_2)
else:
  print("unijeli ste pogresan operator")
```

While petlja - (Izvrsavaj blok koda sve dok je ispunjen uslov ...)

While petlja predstavlja strukturu u Python-u koja nam omogucava da prolazimo kroz isti blok koda vise puta, onoliko puta koliko smo to zadali inicijalnim uslovom, odnosno sve dok uslov ima vrijednost Tacno (True) ili dok nasilno ne prekinemo uslov naredbom prekida (**break**).

Dakle svakom iteracijom kroz blok koda, while petlja ce da izvrsi sve sto se nalazi u tijelu petlje. Naravno, uz while petlju mozemo kombinovati i uslove cime dobijamo na brzini koda i vecoj efikasnosti.

Ono sto je bitno napomenti kod while petlje, ona se koristi uglavnom kada unaprijed nemamo definisan broj iteracija.

Izvorni kod: kod-310_while_brojac.py

```
i = 1
while i <= 10:
  print("Vrijednost i je : " + i)
  #i = i + 1
  i += 1

print("Kraj brojaca")
```

Izvorni kod: kod-311_igra_pogadjanja.py

```
# primjenimo do sad nauceno
tajna_rijec = "python"

pokusaj = ""

while guess != tajna_rijec:
  pokusaj = input("Pokusajte pogoditi tajnu rijec: ")

print("Cestitamo, pogodili ste")

# limitiraj broj pogresnih pokusaja
tajna_rijec = "python"
pokusaj = ""
pokusaj_broj = 0
pokusaj_limit = 4
kraj_igre=False

while pokusaj != tajna_rijec and not(kraj_igre):
if pokusaj_broj < pokusaj_limit:
    pokusaj = input("Pokusajte pogoditi tajnu rijec: ")
    pokusaj_broj += 1
else:
    kraj_igre = True

if kraj_igre:
    print("Iskoristili ste sve pokusaje. Kraj igre")
else:
    print("Cestitamo, pogodili ste")
```

Naredba prekida (**break**)

Izvorni kod: kod-312_demonstracija-naredbe-break.py

```
karakter_opis = {}

brojac=0
limit=10

while brojac <= limit:
    karakter_ime = input("Unesi ime karaktera: ")
    karakter_godine = input("Unesi godine karaktera :")

    if int(karakter_godine) <= 0:
        print("Godine ne mogu biti manje od 1!")
        break
    else:
        karakter_opis[(karakter_ime)] = (karakter_godine)
        brojac+=1

print(karakter_opis)
```

else kod While petlje

Kao sto smo vidjeli sa uslovom **if**, takodje mozemo koristiti granu else prilikom konstrukcije while petlje, ali trebamo imati na umu da se else izvrsava samo jednom, ako i samo ako je glavni uslov while petlje netacan (False). Naravno ukoliko unutar while petlje imamo naredbu **break** koja je izvrsena, else naredba ce biti preskocena.

Izvorni kod: kod-313_demonstracija-grane-else.py

```
karakter_opis = {}

brojac=0
limit=2

while brojac <= limit:
    karakter_ime = input("Unesi ime karaktera: ")
    karakter_godine = input("Unesi godine karaktera :")

    if int(karakter_godine) <= 0:
        print("Godine ne mogu biti manje od 1!")
        break
    else:
        karakter_opis[(karakter_ime)] = (karakter_godine)
        brojac+=1
else:
    # sadrzaj rijecnika ce biti ispisan samo ako se
    # kompletan program izvrsi bez okidanja/trigerovanja naredbe break
    print(karakter_opis)
```

For petlja

For petlju mozemo nazvati specijalni tip petlje u Python-u, a za razliku od while petlje, for petlju koristimo kada zelimo da vrsimo iteraciju kroz tijelo petlje ako unaprijed znamo koliko puta je to potrebno.

Vrijednosti se uglavnom zadaju kao predefinisane ali mozemo koristiti izvore poput lista, stringova, rijecnika.

Izvorni kod: kod-320_rad-sa-for-petljom.py

```
# operator in, za iteraciju nad listama, torkama, rijecnicima mozemo
koristit kljucnu rijec in
for slovo in "programiranje.ba":
  print(slovo)

for karakter in karakteri:
  print(karakter)

karakteri = ["Goku", "Kirlin", "Yamcha"]
for indeks in range(len(karakteri)):
  print (karakteri[indeks])

loto_brojevi = [1,33,13,43,56]
for broj in loto_brojevi:
  print (broj)

for broj in range(20):
  print(broj)

for broj in range(14, 20):
  print (broj)

for indeks in range(len(karakteri)):
  print (karakter[indeks])

for broj in range(5):
    if broj == 0:
        print("prvi pokusaj")
    else:
        print("ostali")

for i in range(10):
    print (i)
    i+=1

# break naredba unutar for petlje
karakteri = ["Goku", "Kirlin", "Yamcha", "Goku", "Bulma"]
for i in range(len(karakteri)):
  if karakteri[i] == "Yamcha":
    print("Prekini izvrsenje for petlje")
    break
  print (karakteri[i])

# enumerate() funkcija - enumeracija
```

```
# ukoliko zelimo da zajedno sa vrijednostima iz liste, stringa ili
rijecnika ispisujemo i njihove indekse
# koristicemo funkciju enmureate()
karakteri = ["Goku", "Kirlin", "Yamcha", "Goku", "Bulma"]
for i,ime in enumerate(karakteri):
  print(str(i) + " " + ime)

### eksponencijaln funkcija - kada ne znamo koliki je eksponent
#print(2**3)

def eksponent_broja(baza, eksponent):
  rezultat = 1
  for i in range(eksponent):
    rezultat = rezultat * baza
  return rezultat

print(eksponent_broja(2,3))
```

Primjer algoritma sortiranje mjehuricima (bubble sorting)

Prije nego napisemo kod potrebno je kratko objasnjenje algoritma. Algoritam sortiranje mjehuricima, ima za cilj da nad zadatim nizom nasumicnih/slucajnih brojeva izvrsi sortiranje od najmanjeg ka najvecem. Ovakvi tipovi zadataka predstavljaju osnovne koncepte teorije algoritma, a mozemo ih naci na kao zadaci na intervjuima u velikim firmama poput Google-a, Amazon-a, Facebook-a, Microsoft-a ...

Predpostavimo da imamo niz brojeva:

```
[4,2,1,5,3]
```

Primjenom algoritma sortiranja mjehuricima, svakom novom iteracijom, svaki element niza ce se uporedjivati sledecim, u slucaju da je prvi element veci od sledeceg, zaminijece mijesta, u suprotnom prvi element ostaje na svom mjestu. Ovaj proces se nastavlja sve dok se svi elementi konacno ne sortiraju od najmanjeg ka najvecem. Dakle, procedura sortiranje ce se obaviti sledecim redosledom:

```
tbd.
[2, 4, 1, 5, 3]
[2, 1, 4, 5, 3]
[2, 1, 4, 3, 5]
[1, 2, 4, 3, 5]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]

Pocetno stanje          |   [4,2,1,5,3]   |   Objasnjnje
________________________|_________________|_____________________________________
_________________
Nakon prve iteracije    |   [(2,4),1,5,3] | 4 i 2 mijenjaju mjesta jer je 4
vece od 2
                        |   [2,(1,4),5,3] | 4 i 1 mijenjaju mjesta jer je 4
```

```
vece od 1
                        |   [2,1,(4,5),3] | 4 i 5 ostaju na svom mjestu jer
je 4 manje od 5
                        |   [2,1,4,(3,5)] | 5 i 3 mijenjaju mjesta jer je 5
vece od 3
________________________|_________________|_____________________________________
________________
Nakon druge iteracije   |   [(1,2),4,3,5] | 2 i 1 mijenjaju mjesta jer je 2
vece od 1
                        |   [1,(2,4),3,5] | 2 i 4 ostaju na svom mjestu jer
je 2 manje od 4
                        |   [1,2,(3,4),5] | 4 i 3 mijenjaju mjesta jer je 4
vece od 3
                        |   [1,2,3,(4,5)] | 4 i 5 ostaju na svom mjestu jer
je 4 manje od 5
```

Izvorni kod: kod-321_sortiranje-mjehuricima.py

```
niz_brojeva = [4,2,1,5,3]

# inicijalno stanje varijable, koja nam govori da li je bilo
# zamjene brojeva prilikom iteracije kroz niz
zamjena_izvrsena = True

zadnji_element_niza = (len(niz_brojeva) - 1)

while zamjena_izvrsena:
  # predpostavimo da je niz sortiran
  zamjena_izvrsena = False

  # isijecamo poslednji element niza, jer unutar petlje provjeravamo
naredni preko indeks + 1
  for indeks,broj in enumerate(niz_brojeva[0:zadnji_element_niza]):
    if niz_brojeva[indeks] > niz_brojeva[indeks+1]:
      # mijenjamo mjesta elemenata niza

niz_brojeva[indeks],niz_brojeva[indeks+1]=niz_brojeva[indeks+1],niz_brojeva
[indeks]
      zamjena_izvrsena = True
else:
```

dvodimenzionalne liste i ugnijezdene petlje (nested)

```
resetka = [
    [1, 2, 3],
    [4, 5, 6],
    [7 ,8 ,9],
    [0]
```

```
]

print(resetka[0][2])
print(resetka[2][1])
print(resetka[3][0])

# nested for loop

# resetka = [
#     [1, 2, 3],
#     [4, 5, 6],
#     [7 ,8 ,9],
#     [0]
# ]

for row in resetka:
  for col in row:
    print(col)

for col in resetka:
    for row in col:
        print(row)

# cezarova sifra u python
def enkripcija(tekst, pomak):
  rezultat = ""

  for broj in range(len(tekst)):
    slovo = tekst[broj]

  if(slovo.isupper()):
    rezultat += chr((ord(slovo) + pomak - 65) % 26 + 65)
  else:
    rezultat += chr((ord(slovo) + pomak - 97) % 26 + 65)
  return rezultat

tekst = input("Unesite tekst: ")
pomak = 2

print("Unijeli ste: " + tekst)
print("Pomak: " + str(pomak))
print("Sifrovan tekst: " + enkripcija(tekst, pomak))
```

try / except (catch) - hvatanje greski

- kada ne zelimo da nas program puca
- ipak zelimo da nastavimo i da damo informaciju korisniku

```
try:
  broj = int(input("Unesite broj: "))
  print(broj)
```

```
except:
  print("Pogresan unos")

###


try:
  vrijednost = 10 / 0
  broj = int(input("Unesite broj: "))
  print(broj)
except ZeroDivisionErron:
  print("Dijeljenje sa nulom")
except ValueError:
  print("Pogresan unos")

###


try:
  vrijednost = 10 / 0
  broj = int(input("Unesite broj: "))
  print(broj)
except ZeroDivisionErron as err:
  print(err)
except ValueError:
  print("Pogresan unos")
```

citanje iz eksternog fajla

- dosta puta imamo potrebu za citanjem sadrzaja iz drugih fajlova
- parsiranje teksta ...
- apsolutni, relativna lokacija

karakteri_porijeklo.txt

```
Goku - Vegeta
Krilin - Zemlja
Piccolo - Namek
Frieza - Universe 7

# r - read; w - write; a - append to end of file; r+ - read and write
# otvoren fajl
karakteri_fajl = open("karakteri_porijeklo.txt", "r")

# provjeri da li je fajl citljiv
print(karakteri_fajl.readable())

# citanje informacija iz fajla
print(karakteri_fajl.read())

# citanje linije u fajlu
```

```
print(karakteri_fajl.readline())
print(karakteri_fajl.readline())
print(karakteri_fajl.readline())
print(karakteri_fajl.readline())

# citanje linije po liniju, citaj svaku liniju i pohrani u niz
print(karakteri_fajl.readlines())

# koristenjem for petlje
for karakter in karakteri_fajl.readlines():
  print(karakter)

karakteri_fajl.close()
```

upisivanje u eksterni fajl

```
karakteri_fajl = open("karakteri_porijeklo.txt", "a")

karakteri_fajl.write("Bulma - Zemlja")
karakteri_fajl.write("\n Chi-Chi - Zemlja")

karakteri_fajl.close()

### upisivanje u eksterni fajl

karakteri_fajl = open("karakteri_porijeklo_novi.txt", "w")

karakteri_fajl.write("Bulma - Zemlja")
karakteri_fajl.write("\n Chi-Chi - Zemlja")

karakteri_fajl.close()
```

moduli i pip alat

- python fajl koji mozete importovati unutar vaseg python koda
- kako kreirati svoj modul
- kako instalirati module (list of python modules on google) pip paket manager
- build-in moduli (ugradjeni) i eksterni moduli

korisni_alati.py

```
#
import random

def srecan_broj(broj):
    return random.randint(1, broj)

def pozdrav(tekst):
```

```
    return ("Pozdrav " + tekst)

##
import korisni_alati

print(korisni_alati.srecan_broj(3))
print(korisni_alati.pozdrav("Goku"))

##
from korisni_alati import *

print(srecan_broj(3))
print(pozdrav("Goku"))

##
import korisni_alati as ka

print(ka.srecan_broj(3))
print(ka.pozdrav("Goku"))

## how to install with pip
## how to import and use modules
```

Klase i objekti

Znamo od pocetka da je Python objektno orijentisani programski jezik, sto znaci da su svi podaci predstavljeni kao objekti. Prihvatanjem objektno orijentisanog nacina programiranja, nasi programi, ali sam kod, postaju ekstremno korisni, organizovaniji i mocniji. Kada programiramo susrecemo se sa razlicitim tipovima podataka, takodje susrecemo se sa razlicitim strukturama podataka, ali sta u slucaju kad ne mozemo predstaviti neku pojavu iz prirode sa vec postojecim tipovima ili strukturama podataka? Upravo su nam za to korisne klase. U Python-u mozemo krairati klase (definise vas licni tip podatka, ponasa se kao template, patern kako nesto treba da izgleda). Objekat je podatak u memoriji, pravi podatak kreiran iz klase.

Posto ne postoji student tip podaka, kreiracemo klasu Student

Student.py

```
class Student:
  # inicijalizacija klase (inicijalna funkcija)
  def __init__(self, ime, smjer, ocjena, brucos):
    self.ime = ime
    self.smjer = smjer
    self.ocjena = ocjena
    self.brucos = brucos
```

main.py

```
from Student import Student

# kreiranje instance Student student_prvi objetka
student_prvi = Student("Goku", "Programiranje", 8, False)
print(student_prvi)
print(student_prvi.ime)
print(student_prvi.ocjena)

student_drugi = Student("Krilin", "Ekonomija", 8.3, True)
print(student_drugi.brucos)
```

funkcije unutar klase (funkcije objekta)

```
class Student:
  def __init__(self, ime, smjer, ocjena, brucos):
    self.ime = ime
    self.smjer = smjer
    self.ocjena = ocjena
    self.brucos = brucos

  def dobar(self):
    if self.ocjena >= 7:
      return True
    else:
      return False
```

naslijedjivanje

- u slucaju da imamo klasu, kada kreiramo novu klasu mozemo naslijediti staru klasu
- nova klasa ce imati sve osobine stare klase sa novim opcijama

Kuvar.py

```
class Kuvar:
    def priprema_mesa(self):
        print ("Priprema pileceg mesa.")

    def priprema_salate(self):
        print("Pirprema Cezar salata")

    def priprema_specijalnog_jela(self):
        print("Priprema rebarcadi")
```

KineskiKuvar.py

```
from Kuvar import Kuvar

class KineskiKuvar(Kuvar):
    def priprema_rize(self):
        print("Priprema rize na kineski nacin")

    def priprema_specijalnog_jela(self):
        print ("Pekinska patka")
```

main.py

```
from Kuvar import Kuvar
from KineskiKuvar import KineskiKuvar

novi_kuvar = Kuvar()

novi_kuvar.priprema_mesa()
novi_kuvar.priprema_specijalnog_jela()

novi_kineski_kuvar = KineskiKuvar()

novi_kineski_kuvar.priprema_rize()
novi_kineski_kuvar.priprema_mesa()
novi_kineski_kuvar.priprema_specijalnog_jela()
```

devet_depresivaca.py - (Rambo Amadeus - Devet depresivaca)

```
print("9 depresivaca gajili su bostan")
print("Puko lastik od bandzija, ostalo ih 8")

print("8 depresivaca, k'o u dlan ih gledam")
print("u krivini hladnjaca, ostalo ih 7")

print("7 depresivaca, opet losa vijest")
print("Neuzemljen bojler, ostalo ih 6")

print("6 depresivaca, turbulentan let")
print("Dnevno kilo vinjaka, ostalo ih 5")

print("5 depresivaca bez mrlje na jetri")
print("Moca od pecenja, ostalo ih 4")

print("4 depresivca, veseli su svi")
print("Droga jeftinija od viskija, ostalo ih 3")

print("3 depresivca, svaki od njih vrijedan")
print("Dvojici crko facebook, ostao je 1")
```

```
print("1 depresivac, oprezan je bio")
print("Onda se ozenio")
```

```
#potapanje brodova
```

```
# Vjesala

import random

fajl = "rijecnik.txt"

def ucitaj_rijeci():
    """
    Vraca listu validnih rijeci. Rijeci su tipa string, napisane malim
slovima

    U zavisnosti od duzine liste rijeci, ova funkcija moze potrajati.
    """
    print("Ucitavanje rijeci iz fajla 'rijecnik.txt'...")
    # otvori_fajl: fajl
    otvori_fajl = open(fajl, 'r')
    # linija: string (procitaj liniju u fajlu - citav fajl je napisan kao
jedna linija)
    linija = otvori_fajl.readline()
    # lista_rijeci: lista rijeci (rijec po rijec)
    lista_rijeci = linija.split()
    print("  ", len(lista_rijeci), "rijeci ucitano.")
    return lista_rijeci

# listarijeci_pomoc = ucitaj_rijeci.__doc__
# print(listarijeci_pomoc)
# print(listarijeci)
# lista_rijeci = ucitaj_rijeci()
# print(lista_rijeci)

def izbor_rijeci(lista_rijeci):
    """
    fajl (lista): lista rijeci (string)

    Funkcija vraca slucajnu rijec iz liste lista_rijeci
    """
    return random.choice(lista_rijeci)

# izabrana_rijec = izbor_rijeci(lista_rijeci)
# print(izabrana_rijec)

# kraj pomocnog koda
# ---------------------------------
```

```
# Ucitaj listu rijeci u varijablu lista_rijeci
# kako bi smo mogli pristupiti listi bilo gdje iz programa
lista_rijeci = ucitaj_rijeci()

def da_li_je_rijec_pogodjena(tajna_rijec, pogodjena_slova):
    '''
    tajna_rijec: string, rijec koju igrac pogadja
    pogodjena_slova: lista, koja slova su pogodjena
    Funkcija vraca: boolean, Tacno (True) ako su sva koja se nalaze u
tajna_rijec rijeci takodje
    u pogodjena_slova, u suprotnom Netacno (False)
    '''
    brojac=0
    for slovo in tajna_rijec:
        if slovo in pogodjena_slova:
            brojac+=1
    if brojac==len(tajna_rijec):
        return True
    else:
        return False

def dohvati_pogodjenu_rijec(tajna_rijec, pogodjena_slova):
    '''
    tajna_rijec: string, rijec koju igrac pogadja
    pogodjena_slova: lista, koja slova su pogodjena
    Funkcija vraca: string, kombinaciju slova i donjih crtica koji
predstavljaju pogodjena slova i
      slova koja jos nisu pogodjena, respektivno.
    '''
    lista=[]
    rijec=""
    for key in tajna_rijec:
        if key in pogodjena_slova:
            rijec+=key
        else:
            rijec+="_ "
    return rijec

# tajna_rijec="asa"
# pogodjena_slova=["a"]
# rijecica=dohvati_pogodjenu_rijec(tajna_rijec,pogodjena_slova)
# print(rijecica)

def dohvati_raspoloziva_slova(pogodjena_slova):
    '''
    pogodjena_slova: lista, koja slova su pogodjena
    Funkcija vraca: string, listu slova koji sacinjavaju slova koja jos
trebaju biti pogodjena.
    '''
    rijec=""
    brojac=0
    slova='abcdefghijklmnopqrstuvwxyz'
    for slovo in slova:
        if slovo in pogodjena_slova:
```

```
            brojac+=1
        else:
            rijec+=slovo
    return rijec

# pogodjena_slova=["a"]
# rijecica=dohvati_raspoloziva_slova(pogodjena_slova)
# print(rijecica)


def vjesala(tajna_rijec):
    '''
    tajna_rijec: string, rijec koju igrac pogadja
    pogodjena_slova: lista, koja slova su pogodjena
    Funkcija vraca: string, kombinaciju slova i donjih crtica koji
predstavljaju pogodjena slova i
      slova koja jos nisu pogodjena, respektivno.

    Startuje interaktivnu igricu vjesala.

    * Na pocetku igre, daje informaciju igracu koliko
      slova ima tajnoj rijeci.

    * Pitaj igraca da proslijedi jedno slovo po pokusaju.

    * Igrac bi trebao dobiti informaciju odmah nakon pokusaja
      gdje se u rijeci nalazi slovo u slucaju da je pogodio.

    * Nakon svakog pokusaja, igrac bi trebao vidjeti djelimicno
      pogodjenu rijec, ali i slova koja nedostaju u rijeci.

    '''
    # main funkcija
    duzina_tajne_rijeci=len(tajna_rijec)
    print("Dobrodosli u igricu, Vjesala!")
    print("Razmisljam o rijeci duzine " + str(duzina_tajne_rijeci) + "
karaktera.")
    dozvoljen_broj_pokusaja=2*len(tajna_rijec)
    i=0
    pogodjena_slova=[]
    while (dozvoljen_broj_pokusaja != 0):
        print("------------")
        if tajna_rijec != dohvati_pogodjenu_rijec(tajna_rijec,
pogodjena_slova):
            print("Imate jos " + str(dozvoljen_broj_pokusaja) + "
pokusaja.")
            print("Raspoloziva slova:
",dohvati_raspoloziva_slova(pogodjena_slova))
            pokusaj=input("Molimo pokusajte slovo: ")
            pokusaj_mala_slova = pokusaj.lower()

            if pokusaj_mala_slova  in pogodjena_slova:
                print("Ups! Vec ste pokusali to slovo:
",dohvati_pogodjenu_rijec(tajna_rijec, pogodjena_slova))
```

```
            elif pokusaj_mala_slova not in tajna_rijec:
                print("Ups! Slovo se ne nalazi u rijeci koju sam
zamislio:",dohvati_pogodjenu_rijec(tajna_rijec, pogodjena_slova))
                dozvoljen_broj_pokusaja-=1
            else:
                pogodjena_slova.append(pokusaj_mala_slova)
                print("Pogodili ste: ",dohvati_pogodjenu_rijec(tajna_rijec,
pogodjena_slova))
                #chances+=1
            pogodjena_slova.append(pokusaj_mala_slova)
        elif tajna_rijec==dohvati_pogodjenu_rijec(tajna_rijec,
pogodjena_slova):
            print("Cestitamo!")
            break
    else:
        print("----------")
        print("Nazalost, nemate vise pokusaja. Tajna rijec je bila "+
tajna_rijec +".")




# Kada zavrsite funkciju vjesala, mozete testirati vas kod definisanjem
# tajne rijeci tajna_rijec="tajna"

#tajna_rijec= izbor_rijeci(lista_rijeci).lower()
#tajna_rijec="tajna"
tajna_rijec=izbor_rijeci(lista_rijeci)

vjesala(tajna_rijec)
``````python
from random import randint
mreza=[]

for i in range(0,7):
    mreza.append(["#"] * 7)
def stampaj_mrezu (board):
    for red in mreza:
        print (" ".join(red))

print ('Igra Potapanje brodova moze da pocne!')
stampaj_mrezu(mreza)

#Slučajnim odabirom u mrežu ubacujemo oba broda
def nasumicni_red(mreza):
    return randint(0, len(mreza)-1)

def nasumicna_kolona(mreza):
    return randint(0, len(mreza[0])-1)

#brod 1
red_1 = nasumicni_red(mreza)
kol_1 = nasumicna_kolona(mreza)
```
```

```
#brod 2
red_2 = nasumicni_red(mreza)
kol_2 = nasumicna_kolona(mreza)
#Da se brodovi ne bi preklapali, potrebno je da nemaju zajednička polja
#To se obezbeđuje funkcijom razliciti()
def razliciti(r,c):
    while r == red_1 and c == kol_1:
        r = nasumicni_red(mreza)
        c = nasumicna_kolona(mreza)
        red_2 = r
        kol_2 = c
razliciti(red_2,kol_2)
#Kada izaberete jedno polje, preostala dva mogu biti horizontalno(levo ili
desno) ili vertikalno(gore ili dole)
#Zato se definišu sledeći pravci
def nasumicni_pravac():
    n = randint(1,4)
    if n == 1:
        return "gore"
    elif n == 2:
        return "desno"
    elif n == 3:
        return "dole"
    elif n == 4:
        return "levo"
#Nasumično se odredi pravac, i na osnovu njega sledeća dva polja
while True:
    d = nasumicni_pravac()
    if d == "gore":
        if red_1 >= 2:

            red_1_2 = red_1 - 1
            kol_1_2 = kol_1
            red_1_3 = red_1 - 2
            kol_1_3 = kol_1
            break
    if d == "desno":
        if kol_1 <= len(mreza[0])-3:

            red_1_2 = red_1
            kol_1_2 = kol_1 + 1
            red_1_3 = red_1
            kol_1_3 = kol_1 + 2
            break
    if d == "dole":
        if red_1 <= len(mreza)-3:

            red_1_2 = red_1 + 1
            kol_1_2 = kol_1
            red_1_3 = red_1 + 2
            kol_1_3 = kol_1
            break
    if d == "levo":
        if kol_1 >= 2:
```

```
            red_1_2 = red_1
            kol_1_2 = kol_1 - 1
            red_1_3 = red_1
            kol_1_3 = kol_1 - 2
            break
brod_1 = [(red_1 ,kol_1 ),(red_1_2 ,kol_1_2 ),(red_1_3 ,kol_1_3 )]



#drugi brod:
while True:
#Nasumično se odredi pravac, i na osnovu njega sledeća dva polja
#Uslov je da se ne preklapaju sa poljima prvog broda
    d = nasumicni_pravac()
    if d == "gore":
        if red_2 >= 2:
            if (red_2 - 1,kol_2) not in brod_1 and (red_2 - 2,kol_2) not in
brod_1:

                red_2_2 = red_2 - 1
                kol_2_2 = kol_2
                red_2_3 = red_2 - 2
                kol_2_3 = kol_2
                break
    if d == "desno":
        if kol_2 <= len(mreza[0])-3:
             if (red_2 ,kol_2 + 1) not in brod_1 and (red_2,kol_2 + 2) not
in brod_1:

                red_2_2 = red_2
                kol_2_2 = kol_2 + 1
                red_2_3 = red_2
                kol_2_3 = kol_2 + 2
                break
    if d == "dole":
        if red_2 <= len(mreza)-3:
            if (red_2 + 1 ,kol_2) not in brod_1 and (red_2 + 2,kol_2) not
in brod_1:

                red_2_2 = red_2 + 1
                kol_2_2 = kol_2
                red_2_3 = red_2 + 2
                kol_2_3 = kol_2
                break
    if d == "levo":
        if kol_2 >= 2:
            if (red_2 ,kol_2 - 1) not in brod_1 and (red_2,kol_2 - 2) not
in brod_1:

                red_2_2 = red_2
                kol_2_2 = kol_2 - 1
                red_2_3 = red_2
                kol_2_3 = kol_2 - 2
```

```
                break



tacan = 0 #U ovoj promenljivoj smešta se ukupan broj pogođenih polja oba
broda
prvi_brod = 0 #U ovoj promenljivoj smešta se broj pogođenih polja prvog
broda
drugi_brod = 0 #U ovoj promenljivoj smešta se broj pogođenih polja drugog
broda
#Na početku nemamo nijedno pogođeno polje, pa sve promenljive postavljamo
na 0
#U ovoj promenljivoj smešta se ukupan broj pogođenih polja oba broda
tacan = 0
#U ovoj promenljivoj smešta se broj pogođenih polja prvog broda
prvi_brod = 0
#U ovoj promenljivoj smešta se broj pogođenih polja drugog broda
drugi_brod = 0

#Dozvoljeno je 15 pokušaja da se potope oba broda
for pokusaj in range(1,16):
    print (str(pokusaj ) + '. pokusaj:')

    nagadjanje_reda  = int(input('Pogodite red:'))
    nagadjanje_kolone  = int(input('Pogodite kolonu:'))
#Ispituje se da li je korisnik pogodio neko polje prvog broda
#Ako jeste, broj pogođenih polja se povećava za jedan
    if ((nagadjanje_reda -1  == red_1 ) and (nagadjanje_kolone -1  ==
kol_1)) or ((nagadjanje_reda -1 == red_1_2 ) and (nagadjanje_kolone -1 ==
kol_1_2)) or((nagadjanje_reda -1 == red_1_3 ) and (nagadjanje_kolone -1 ==
kol_1_3)) and mreza[nagadjanje_reda -1 ][nagadjanje_kolone -1] != 'X' :


        tacan = tacan+1
        prvi_brod = prvi_brod + 1

        if (tacan != 6) and (prvi_brod != 3) :


            print ('Bravo, pogodak!')
            mreza[nagadjanje_reda -1 ][nagadjanje_kolone -1] = 'X'
#Ako je pogođeno polje treće polje prvog broda, korisnik se obaveštava da
je potopio ceo brod
        elif (tacan != 6) and (prvi_brod == 3):

            mreza[nagadjanje_reda -1 ][nagadjanje_kolone -1] = 'X'

            print ('Bravo, potopili ste ceo brod! Ostao vam je jos jedan!')
#Ako je reč o šestom pogođenom polju, korisnik se obaveštava da je potopio
oba broda
        if (tacan == 6):
            mreza[nagadjanje_reda -1][nagadjanje_kolone -1] = 'X'
            print ('Svaka cast, potopili ste oba broda!')
```

```
            break
#Ispituje se da li je korisnik pogodio neko polje drugog broda
#Ako jeste, broj pogođenih polja se povećava za jedan

    elif   ((nagadjanje_reda -1  == red_2 ) and (nagadjanje_kolone -1  ==
kol_2)) or ((nagadjanje_reda -1  == red_2_2 ) and (nagadjanje_kolone -1  ==
kol_2_2)) or ((nagadjanje_reda -1  == red_2_3 ) and (nagadjanje_kolone -1
== kol_2_3))     and mreza[nagadjanje_reda -1 ][nagadjanje_kolone -1] !=
'Y'  :
        tacan = tacan+1
        drugi_brod = drugi_brod + 1
        if (tacan != 6) and (drugi_brod != 3):

            print ('Bravo, pogodak!')
            mreza[nagadjanje_reda -1 ][nagadjanje_kolone -1] = 'Y'
        elif (tacan != 6) and (drugi_brod ==3):

            mreza[nagadjanje_reda -1 ][nagadjanje_kolone -1] = 'Y'
#Ako je pogođeno polje, treće polje prvog broda, korisnik se obaveštava da
je potopio ceo brod

            print ('Bravo, potopili ste ceo brod! Ostao vam je jos jedan!')
#Ako je reč o šestom pogođenom polju, korisnik se obaveštava da je potopio
oba broda

        if (tacan == 6):
            mreza[nagadjanje_reda -1][nagadjanje_kolone -1] = 'Y'
            print ('Svaka cast, potopili ste oba broda!')
            break
    else:
        if (nagadjanje_reda < 1 or nagadjanje_reda > 7) or
(nagadjanje_kolone < 1 or nagadjanje_kolone > 7):
            print ('Ups, izvan opsega ste!')
        elif (mreza[nagadjanje_reda -1 ][nagadjanje_kolone -1]=='X'):
            print ('Vec ste pronasli ovaj deo broda!')
        elif (mreza[nagadjanje_reda -1 ][nagadjanje_kolone -1]=='O'):
            print ('Vec ste pogadjali isto polje!')
        else:
            print ('Promasili ste!')
            mreza[nagadjanje_reda -1 ][nagadjanje_kolone -1] = 'O'


    stampaj_mrezu(mreza)
    if (pokusaj == 15):
            print ('Igra je zavrsena!')

if (mreza[red_1 ][kol_1 ] != "X" or mreza[red_1_2 ][kol_1_2 ] != "X" or
mreza[red_1_3 ][kol_1_3 ] != "X") or (mreza[red_2 ][kol_2 ] != "Y" or
mreza[red_2_2 ][kol_2_2 ] != "Y" or
mreza[red_2_3 ][kol_2_3 ] != "Y"):
    print ('Brodovi su se nalazili na ovim pozicijama! Vise srece drugi
put!')
    mreza[red_1 ][kol_1 ] = "X"
    mreza[red_1_2 ][kol_1_2 ] = "X"
```

```
    mreza[red_1_3][kol_1_3] = "X"
    mreza[red_2][kol_2 ] = "Y"
    mreza[red_2_2 ][kol_2_2] = "Y"
    mreza[red_2_3][kol_2_3 ] = "Y"


stampaj_mrezu(mreza)
```

```
ime = input("Unesite vase ime: ")
pet_puta=(ime+" ")*5
print(pet_puta)
```

```
a=input("Unesite znak sa tastature: ")
b=ord(a)

print("Vrijednost znaka '" + a + "' u ASCII tabeli je " + str(b))
```

```
a=input("Unesite prvi broj: ")
b=input("Unesite drugi broj: ")
c=input("Unesite treci broj: ")
asredina=(int(a)+int(b)+int(c))/3
print("Aritmeticka sredina je: " + str(asredina))
```

```
prva_rijec  = input("Unesite prvu rijec: ")
druga_rijec = input("Unesite drugu rijec: ")

print(prva_rijec + '    ' + druga_rijec)
```