Skript jezici
V02 – Kolekcije

Milan Tomić

Sadržaj

- Formatiranje stringova
- Kolekcije: liste, n-torke, rečnici, skupovi
- Funkcije: raspakivanje argumenata
- Moduli

Format protokol

- Protokol za formatiranu serijalizaciju objekata
  - Nalik na printf ali sa više mogućnosti
- f-stringovi

% operator

- Videli smo da stringovi podržavaju % operator, koji služi za formatiranje u stilu jezika C
  - %s – stringovi, %d – celi brojevi, %f – realni brojevi
- Ovo formatiranje je uglavnom brže, ali i ima dosta ograničenja u pogledu toga kako se stvari ispisuju
  - Naročito važi za složene objekte

Primer korišćenja % operatora za formatiranje stringa

```
#%% Formatiranje pomoću %
a = 20
b = 13.4278135
c = 'FIN'
s = 'Ceo broj: %d, Realni broj: %-7.3f, String: %s' % (a, b, c)
print(s)  # Ceo broj: 20, Realni broj: 13.428 , String: FIN
```

str.format

- Stringovi imaju format metodu koja služi za komplikovanije formatiranje u serijalizaciji
- Klasa Formatter u string modulu omogućuje korisniku da definiše svoja ponašanja (format stringove) za formatiranje svojih klasa
  - Time se nećemo baviti, ali ako vam zatreba možete pronaći ovde: https://docs.python.org/3/library/string.html#custom-string-formatting
- Metoda traži mesta unutar stringa označena vitičastim zagradama { } i na tim mestima upisuje odgovarajuće argumente u zadatom formatu
- Svako takvo mesto (polje) može da sadrži numerički indeks ili naziv argumenta koji je prosleđen format metodi
  - O raspakivanju argumenata više na narednim časovima

Formatiranje u format poljima

- Okvirno: {naziv argumenta ! konverzija : format}
- Naziv argumenta predstavlja literal ili izraz kako se inače pristupa vrednosti argumenta funkcije format koje se ispisuje – redni broj argumenta ili njegovo ime
  - x
  - x.polje
  - x[indeks] – samo numerički indeks, ne prihvata stringove
  - kombinacije gorenavedenih
- Od verzije 3.1. za pozicione argumente nije neophodno navoditi indeks (ako idu redom), dovoljno je ostaviti prazna polja
- Kompletnu specifikaciju možete pronaći ovde: https://docs.python.org/3/library/string.html#formatstrings

Formatiranje u format poljima – primer

- Sintaksa je za većinu slučajeva slična kao za % operator
  - Razlike možete pronaći u dokumentaciji
  - Klase koje podržavaju specijalno formatiranje imaju to u svojoj dokumentaciji (primer – datetime klasa)

```
#%% Formatiranje pomoću str.format
a = 20
b = 13.4278135
c = 'FIN'
s = 'Ceo broj: {0}, Realni broj: {1:<7.3f}, String: {2}'.format(a, b, c)
print(s)  # Ceo broj: 20, Realni broj: 13.428 , String: FIN
```

f-stringovi

- f-stringovi su specijalna vrsta string literala, koji imaju sličnu funkcionalnost kao str.format
  - Oni postoje od verzije 3.6
- f-stringovi počinju slovom f ili F, posle čega ide uobičajeni string literal
- Ovi literali se evaluiraju u trenutku kada se kreiraju
- Unutar vitičastih zagrada umesto indeksa ili naziva argumenta funkcije primaju standardne python izraze
  - Izrazi se evaluiraju kao da su u zagradama, osim u slučaju lambda funkcije, koja se mora eksplicitno ograničiti zagradama
  - Izraz može da sadrži nove redove, ali ne sme da sadrži komentare
- Nakon izraza ide konverzija (!s – str, !r – repr, !a – ascii) pa format :...
- Kompletnu specifikaciju f-stringova možete pronaći ovde: https://docs.python.org/3/reference/lexical_analysis.html#formatted-string-literals

Primer korišćenja f-stringova

```
#%% f-stringovi
a = 20
b = 13.4278135
c = 'FIN'
s = f'Ceo broj: {a}, Realni broj: {b:<7.3f}, String: {c}'
print(s)  # Ceo broj: 20, Realni broj: 13.428 , String: FIN
```

Liste (list)

- Liste u Pythonu su mutabilne kolekcije proizvoljne veličine elemenata proizvoljnog tipa u kojima je bitan poredak elemenata po poziciji
- Poredak po poziciji: liste se ponašaju kao nizovi u drugim jezicima, pristup elementu na proizvoljnoj poziciji je u O(1)
- Mutabilne: na svakoj poziciji element može da se zameni drugim elementom; u listu se mogu dodavati elementi (pri čemu se lista produžuje), iz liste se mogu brisati elementi (pri čemu se lista skraćuje)
- Proizvoljne veličine: od prazne liste do bilo koje dužine koju memorija može da podnese
- Elementi proizvoljnog tipa: u listi se istovremeno mogu naći brojevi, stringovi i bilo koji drugi objekti (nije određena tipom elemenata koje prihvata)

Operacije nad listama

- Liste podržavaju:
  - Indeksiranje i slajsovanje (uglaste zagrade)
    - Pomoću indeksiranja se pristupa jednom elementu, koji može da se pročita, setuje ili izbriše iz liste
    - Pomoću slajsovanja se pristupa podlisti, koja takođe može da se pročita, setuje ili izbriše iz liste
  - Proveru dužine (funkcija len)
  - Proveru postojanja elementa (binarni operator in)
  - Pošto su mutabilne, ako bilo koja funkcija promeni sadržaj liste koja je prosleđena kao argument, to se reflektuje i van funkcije
- Konstruktor liste: list(iterable) – pravi listu od elemenata iz kolekcije
- Može se zadati i u obliku [1, 2, 3, 4, 5]

Metode listi

- append(obj) – dodaje obj na kraj liste
- clear() – prazni listu
- copy() – plitka (*shallow*) kopija liste – vraća novu listu sa istim elementima
- count(obj) – vraća broj pojavljivanja obj u listi
- extend(lst) – dodaje sve elemente iz kolekcije lst na kraj liste
- index(obj,?start,?end) – vraća indeks prvog pojavljivanja elementa obj u listi
- insert(i, obj) – ubacuje objekat obj na indeks i
- pop(?i) – izbacuje i vraća element sa indeksa i (tj. sa kraja liste)
- remove(obj) – izbacuje prvo pojavljivanje objekta obj iz liste
- reverse() – obrće listu (mutira je!)
- sort(key=None, reverse=False) – sortira listu stabilnim algoritmom (mutira je!), po zadatom ključu (ili u prirodnom poretku), u normalnom ili obrnutom poretku

Liste – primeri – konstruktori

```
#%% Liste primeri (1)

# konstrukcija liste
l = [1, 2, 3, 4, 5, 6, 7, 8]
print(l)  # [1, 2, 3, 4, 5, 6, 7, 8]

# konstruktor preko iterable, npr. range
rl = list(range(10))
print(rl)  # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# NIJE ISTO KAO PRVO:
rl2 = [range(10)]
print(rl2)  # [range(0, 10)]
```

Liste – primeri – indeks i slajs

```
#%% Liste primeri (2)

# indeksiranje
print(l[3])  # 4
l[4] = 9
print(l)  # [1, 2, 3, 4, 9, 6, 7, 8]

# slajsovanje
print(l[2:6])  # [3, 4, 9, 6]

# dodela slajsa (ako je proširen korakom onda mora biti iste veličine)
l[2:6] = [4, 7]
print(l)  # [1, 2, 4, 7, 7, 8]
```

Liste – primeri – operacije

```
#%% Liste primeri (3)

# provera postojanja
print(2 in l)  # True
# pronalaženje indeksa
print(l.index(7))  # 3
# prebrojavanje
print(l.count(7))  # 2
# dodavanje
l.append(4)
# obrtanje
l.reverse()
print(l)  # [4, 8, 7, 7, 4, 2, 1]
# sortiranje
l.sort()
print(l)  # [1, 2, 4, 4, 7, 7, 8]
```

n-torke (tuple)

- n-torke u Pythonu su nemutabilne kolekcije proizvoljne veličine elemenata proizvoljnog tipa u kojima je bitan poredak elemenata po poziciji
- Mogu se posmatrati kao konstantne liste – čuvaju poredak, mogu da se indeksiraju, ali ne mogu im se menjati članovi, niti se mogu proširivati ili sužavati
- Upotreba n-torki je uglavnom u situacijama kada je potrebna upotreba liste, ali nije neophodno da se sadržaj i oblik liste menja
- Upotreba n-torke je memorijski i vremenski efikasnija nego upotreba liste
- Konstruktor: tuple(iterable)
- Može se zadati i u obliku: (1, 2, 3, 4, 5)

Operacije nad n-torkama

- Podržavaju indeksiranje i slajsovanje ali samo za čitanje
  - Slajs n-torke je takođe n-torka
- Podržavaju metode index i count
- Za razliku od liste, n-torke su hešabilne* jer su nepromenljive
  - Pod uslovom da su svi elementi koje n-torka sadrži takođe hešabilni
  - Funkcija hash() može se iskoristiti za izračunavanje heša n-torke (i bilo kog drugog objekta kome je heširanje implementirano)

n-torke - primeri

```
#%% n-torke primeri

t = (1, 2, 3, 4, 5, 6)
print(t)  # (1, 2, 3, 4, 5, 6)
rt = tuple(range(1, 10))
print(rt[3])  # 4
print(rt[2:8:2])  # (3, 5, 7)
print(rt.index(5))  # 4
```

Rečnici (dict)

- Rečnici u Pythonu su tip koji odgovara hešmapama
- Hešmapa uparuje ključeve i vrednosti i to tako što za svaki ključ izračunava heš i onda na osnovu heša skladišti u svoju strukturu par ključa i vrednosti tako da se taj par može pronaći u O(1)
- Odatle je lako zaključiti da ključevi u dictu mogu biti samo hešabilni objekti, dok vrednosti mogu biti bilo kog tipa
  - Brojevi, stringovi i n-torke (uslovno) mogu biti ključevi u dictu, dok liste ne mogu
- Konstruktori: dict(mapping), dict(iterable) – rečnici koji iz mapiranja ili kolekcije uzimaju parove elemenata i postavljaju prvi element kao ključ a drugi kao vrednost
- Može se zadati i u obliku: {’a’: 1, ’b’: 2, ’c’: 3, ’d’: 4, ’e’: 5}

Operacije nad rečnicima

- Rečnici podržavaju indeksiranje, ali ne slajsovanje
  - Umesto klasičnog pozicionog indeksiranja, kao indeksi se koriste ključevi, za get i za set operacije
- Binarni operator in proverava da li se ključ nalazi u rečniku
- Iteracija kroz rečnik je po ključevima
  - Postoje metode za iteraciju po vrednostima i po parovima (ključ, vrednost)
- Rečnici nisu hešabilni

Metode rečnika

- clear() – prazni rečnik
- copy() – vraća *shallow* kopiju rečnika
- fromkeys(iterable, value=None) – pravi novi rečnik od ključeva zadatih u iterable objektu i svakom ključu pridružuje datu vrednost (default None)
- get(key, default=None) – vraća element koji je pridružen ključu key ili default ako se ključ ne nalazi u rečniku (default None)
- items() – svi parovi (k,v) iz rečnika u skupu; keys() – svi ključevi iz rečnika u skupu
- pop(key, default) – izbacuje element sa ključa key i vraća ga; ako default nije dat a key nije u rečniku, baca KeyError
- popitem() – izbacuje i vraća jedan par (key, value) iz rečnika; ako je rečnik prazan baca KeyError
- setdefault(key, value) – vraća d.get(key, value) i setuje d[key]=value ako key nije u rečniku
- values() – sve vrednosti iz rečnika u skupu

Rečnici primeri - konstruktori

```
#%% Rečnici primeri (1)

# konstrukcija rečnika
d = {'A': 2, 3: 4, (5, 6): 8}
print(d)  # {'A': 2, 3: 4, (5, 6): 8}

# konstruktor preko iterable
dl = dict(((1, 10), (2, 20), (3, 30)))
print(dl)  # {1: 10, 2: 20, 3: 30}
```

Rečnici – primeri – osnovne operacije

```
#%% Rečnici primeri (2)

# dodavanje vrednosti
dl[4] = 40
print(dl)  # {1: 10, 2: 20, 3: 30, 4: 40}
# pristup
print(dl[2])  # 20
# iteracija
for k in dl:
    print(k, dl[k])  # 1 10
                     # 2 20
                     # 3 30
                     # 4 40
# popitem
print(dl.popitem())  # (4, 40)
# update
dl.update({4: 35})
print(dl)  # {1: 10, 2: 20, 3: 30, 4: 35}
# values
print(list(dl.values()))  # [10, 20, 30, 35]
```

Skupovi (set)

- Skupovi u Pythonu su neuređene mutabilne kolekcije proizvoljne veličine jedinstvenih elemenata proizvoljnog hešabilnog tipa
  - Jedinstvenih elemenata – ako je element već u skupu, operacija dodavanja ne utiče na sastav skupa (tj. element se ne dodaje dva puta)
  - Neuređene kolekcije – ne čuva se obavezno redosled dodavanja elemenata
- Skupovi su zgodni za simulaciju matematičkih operacija nad skupovima kao što su presek, unija, razlika, simetrična razlika itd.
- Provera da li je neki element u skupu je složenosti O(1)
- Konstruktor skupa: set(iterable) – pravi novi skup od elemenata u kolekciji iterable
- Može se zadati i u obliku: {1, 2, 3, 4, 5}

Operacije nad skupovima

- Skupovi ne podržavaju indeksiranje i slajsing
- Binarni operator in može se koristiti za proveru da li je element u skupu
- Iteracija kroz skup je u proizvoljnom redosledu elemenata
- Skupovi nisu hešabilni

Metode skupova

- add(x) – dodaje x u skup (ako već nije u njemu)
- clear() – prazni skup
- copy() – vraća *shallow* kopiju skupa
- difference(...) – vraća razliku između tog i jednog ili više datih skupova
- difference_update(...) – ostavlja u skupu samo elemente razlike
- discard(x) – uklanja x iz skupa ako je u njemu, inače ne radi ništa
- intersection(s) – vraća presek sa skupom s
- intersection_update(s) – ostavlja u skupu samo elemente preseka

Metode skupova

- isdisjoint(s) – proverava i vraća da li su skupovi disjunktni
- issubset(s) – da li je skup podskup skupa s
- issuperset(s) – da li je s podskup skupa
- pop() – uklanja i vraća element skupa; KeyError ako je skup prazan
- remove(x) – uklanja x koji je element skupa; KeyError ako x nije u skupu
- symmetric_difference(s) – vraća simetričnu razliku sa skupom s
- symmetric_difference_update(s) – ostavlja samo elemente simetrične razlike u skupu
- union(s) – vraća uniju sa skupom s
- update(s) – dodaje elemente skupa s u skup

Skupovi – primeri – konstruktori

```
#%% Skupovi primeri (1)

# konstrukcija skupa
s = {1, 2, 3, 4, 5}
print(s)  # {1, 2, 3, 4, 5}

# konstruktor preko iterable
sl = set(l)
print(sl)  # {1, 2, 4, 7, 8}

# provera pripadnosti
print(7 in s)  # False

# dodavanje u skup
s.add(6)
print(s)  # {1, 2, 3, 4, 5, 6}
```

Skupovi - primeri - metode

```
#%% Skupovi primeri (2)

# presek
print(s.intersection(sl))  # {1, 2, 4}

# razlika
print(s.difference(sl))  # {3, 5, 6}

# symdiff update
s.symmetric_difference_update(sl)
print(s)  # {3, 5, 6, 7, 8}
```

Funkcije – nastavak

- Videli smo osnovni način definisanja funkcije
- Funkcije u Pythonu podržavaju još i:
  - Definisanje default vrednosti argumenata
  - Prosleđivanje proizvoljnog broja argumenata
    - Po poziciji
    - Po nazivu
  - Prosleđivanje argumenata raspakivanjem iz kolekcija

Definisanje podrazumevanih vrednosti argumenata funkcije

- Argumenti funkcije mogu imati podrazumevane vrednosti
- Argumenti koji imaju podrazumevane vrednosti ne moraju biti prosleđeni prilikom poziva funkcije
- Argumenti koji nemaju podrazumevane vrednosti moraju biti prosleđeni prilikom poziva funkcije
- Argumenti koji imaju podrazumevane vrednosti moraju se navoditi posle argumenata koji nemaju podrazumevane vrednosti (ako takvi argumenti postoje)
- Definicija podrazumevane vrednosti za argument funkcije navodi se u deklaraciji funkcije, dodavanjem =vrednost iza naziva argumenta

Primer definisanja podrazumevane vrednosti argumenta

```
#%% Funkcije - podrazumevane vrednosti argumenata

def kvadrat_ili_stepen(x, n=2):
    return x ** n

print(kvadrat_ili_stepen(10, 3))  # 1000
print(kvadrat_ili_stepen(10))  # 100
```

Poziv funkcije sa eksplicitnim navođenjem imena argumenata

- Funkcija u Pythonu može se pozvati i sa eksplicitnim navođenjem naziva jednog ili više argumenata
- Pravilo je slično kao kod navođenja podrazumevanih vrednosti – prvo se navode svi argumenti koji se prosleđuju po poziciji, a zatim po imenu
  - Svi argumenti mogu se proslediti po imenu, tj. nije neophodno navoditi uopšte pozicione argumente
- Kada se argumenti navode po imenu, mogu se zadati po proizvoljnom redosledu
- Način navođenja je isti – u pozivu funkcije navodi se ime argumenta, znak dodele (=) i vrednost

Primer poziva funkcije sa eksplicitnim navođenjem imena argumenata

```
#%% Funkcije - poziv sa eksplicitnim navođenjem argumenata

def ispis(a, b, c):
    print(f'a = {a}, b = {b}, c = {c}')

ispis(1, 2, 3)  # a = 1, b = 2, c = 3
ispis(3, c=5, b=2)  # a = 3, b = 2, c = 5
```

Pozicioni argumenti – *args

- Funkcije u Pythonu pored navedenih argumenata mogu da primaju i proizvoljni broj pozicionih argumenata, ako je tako definisano u funkciji
- Definicija funkcije koja može da primi proizvoljne pozicione argumente sadrži posle svih poimenično navedenih pozicionih argumenata jedan izraz, koji se obično zapisuje *args
  - Bitno je da ima * ispred sebe, a koji je naziv nije bitno
  - Argumenti koji imaju podrazumevane vrednosti mogu da se stave i posle *args, u kom slučaju će moći da se dodeljuju samo eksplicitnim imenovanjem
- Zahvaljujući tome, formira se n-torka sa datim nazivom (args) koja sadrži sve argumente prosleđene funkciji posle onih koji se podrazumevaju
- U ovoj n-torci argumenti su poređani po redosledu kako su prosleđeni

Primer funkcije sa prosleđenim proizvoljnim pozicionim argumentima

```
#%% Funkcije - pozicioni argumenti [1]

def x_i_ostali(x, *args):
    if len(args) > 0:
        print(f'Drugi argument: {args[0]}')
    else:
        print('Samo jedan argument')
    return f'{x}, {args}'

print(x_i_ostali(10))  # Samo jedan argument
                       # 10, ()
print(x_i_ostali(10, 20, 30, 40))  # Drugi argument: 20
                                   # 10, (20, 30, 40)
```

Imenovani argumenti – **kwargs

- Funkcije u Pythonu pored navedenih argumenata mogu da primaju i proizvoljni broj imenovanih argumenata, ako je tako definisano u funkciji
- Definicija funkcije koja može da primi proizvoljne imenovane argumente sadrži posle svih poimenično navedenih argumenata jedan izraz, koji se obično zapisuje **kwargs
  - Bitno je da ima ** ispred sebe, a koji je naziv nije bitno
- Zahvaljujući tome, formira se rečnik sa datim nazivom (kwargs) koji sadrži sve argumente prosleđene funkciji po imenu osim onih koji se podrazumevaju
- Svakom argumentu iz rečnika može se pristupiti po nazivu po kome je prosleđen funkciji

Primer funkcije sa prosleđenim proizvoljnim imenovanim argumentima

```
#%% Funkcije - imenovani argumenti

def x_i_ostali(x, **kw):
    s = f'x = {x}, kw = {kw}'
    if len(kw) > 0:
        i = kw.popitem()
        print(f'Jedan argument: {i[0]} = {i[1]}')
    return s

print(x_i_ostali(10, a=20, b=30))  # Jedan argument: b = 30
                                   # x = 10, kw = {'a': 20, 'b': 30}
print(x_i_ostali(a=10, x=20, c=30))  # Random argument: c = 30
                                     # x = 20, kw = {'a': 10, 'c': 30}
```

Kombinacija svega navedenog

- U deklaraciji funkcije mogu se navesti redom:
  - Pozicioni argumenti sa imenima i bez podrazumevanih vrednosti
  - Pozicioni argumenti sa imenima i sa podrazumevanim vrednostima
  - Pozicioni argumenti bez imena (*args)
  - Imenovani argumenti sa podrazumevanim vrednostima
  - Imenovani argumenti bez podrazumevanih vrednosti (**kwargs)
- Još jednom pojašnjenje terminologije:
  - Pozicioni – argumenti koji mogu da se prosleđuju bez eksplicitnog navođenja imena
  - Imenovani – argumenti koji mogu da se prosleđuju isključivo sa eksplicitnim navođenjem imena

Raspakivanje argumenata

- Ako pozicione argumente imamo u listi (ili n-torci), a želimo da ih prosledimo nekoj funkciji koja ih prima u tom redosledu, možemo to učiniti korišćenjem operatora * u pozivu
  - Python će argumente prosleđene po redosledu raspakovati kao vrednosti pozicionih argumenata
- Ako imenovane argumente imamo u rečniku, a želimo da ih prosledimo nekoj funkciji koja ih prima po imenu, možemo to učini korišenjem operatora ** u pozivu
  - Python će argumente prosleđene na ovaj način dodeliti argumentima sa imenima koji se nalaze u ključevima rečnika i vrednostima koje odgovaraju tim ključevima

Primer raspakivanja argumenata

```
#%% Funkcije - raspakivanje argumenata

def zbir(x, y):
    return x + y

l = [(1, 2), (3, 4), (5, 6)]
for par in l:
    print(f'{par[0]} + {par[1]} = {zbir(*par)}')
# 1 + 2 = 3
# 3 + 4 = 7
# 5 + 6 = 11

d = {'x': 12, 'y': 13}
print('12 + 13 =', zbir(**d))  # 12 + 13 = 25
```

Modularnost

- Rekli smo da su u Pythonu sve promenljive objekti, tj. instance nekih klasa
- Na stringovima i kolekcijama uvideli smo da Python ima prilično dobru podršku za osnovne operacije nad tim klasama
- Python takođe nudi veliki izbor standardnih biblioteka sa dodatnim klasama za najrazličitije potrebe
  - Pored standardnih, Python community ima i veliki repozitorijum autorskih nestandardnih biblioteka za specifične zadatke
- Biblioteke su spakovane u module, a moduli u pakete, što sve zajedno nije ništa drugo nego klasična direktorijumska struktura u kojoj su paketi i potpaketi folderi, a moduli fajlovi

Naredba import

- Naredba import koristi se za uvoženje modula
- Svi paketi instalirani su u nekom od foldera koji se nalaze na putanji %PYTHONPATH% tj. na spisku foldera u fajl sistemu koje Python interpreter pretražuje
  - Kreće od lokalne strukture – trenutnog foldera u kome radi, a onda obilazi redom foldere zabeležene u PYTHONPATH sistemskoj promenljivoj
- Importovani moduli se ponašaju kao objekti koji u svojim svojstvima imaju sve definisane klase, funkcije i promenljive tog modula
- Navešćemo neke od standardnih modula i primer načina njihovog uvoženja i korišćenja pre nego što pređemo na objektno-orijentisano programiranje

Neki od ~230 modula iz standardne biblioteke (https://docs.python.org/3/library/index.html)

- string – česte operacije sa stringovima
- re – operacije sa regularnim izrazima
- codecs – podrška za rad sa različitim kodovanjima (za tekst i bajtove)
- datetime – rad sa datumima i vremenom
- collections – dodatne alternative uobičajenim Python kolekcijama
- math – osnovne matematičke funkcije
- os – rad sa interfejsima operativnog sistema
- io – za rad sa streamovima
- random – generatori slučajnih događaja
- threading – paralelizam sa nitima
- multiprocessing – paralelizam sa procesima
- http – HTTP moduli
- tkinter – interfejs za rad sa Tcl/Tk grafičkim interfejsom
- sys – sistemski parametri i funkcije

Primeri korišćenja bibliotečkih funkcija

```
#%% Uvoženje modula

# ceo modul
import os
print(os.getcwd())  # trenutna putanja

# deo modula
from string import ascii_uppercase
from random import choice
for i in range(5):
    print(choice(ascii_uppercase), end=' ')  # pet slucajnih velikih slova
print()  # prelazak u novi red
```

Zadaci za samostalan rad

- Vezbe02_zadaci.pdf