Itasdi

Innovative Teaching Approaches in development of Software Designed Instrumentation and its application in real-time systems

Praktikum iz merno-akvizicionih sistema

Co-funded by the Erasmus+ Programme of the European Union

Itasdi

Innovative Teaching Approaches in development of Software Designed Instrumentation and its application in real-time systems

Faculty of Technical Sciences

Ss. Cyril and Methodius University
Faculty of Electrical Engineering and Information Technologies

Zagreb University of Applied Sciences

School of Electrical Engineering
University of Belgrade

Faculty of Physics
Warsaw University of Technology

Co-funded by the Erasmus+ Programme of the European Union

Lekcija 12

Uvod u *Python* programski jezik

Cilj

Cilj lekcije je da se studenti upoznaju sa:

- Python programskim jezikom i programskim okruženjem *Spyder*
- Tipovima podataka koji se koriste u *Python* programskom jeziku
- Petljama i strukturama
- Učitavanjem biblioteka
- Radom sa funkcijama
- Radom sa različitim tipovima datoteka.

12.1 Python i programsko okruženje Spyder

Python je programski jezik visokog nivoa i opšte namene. Dizajniran je tako da se povećava čitljivost koda, a njegova sintaksa omogućava korisnicima brzo i jednostavno rešavanje problema. Podržava različite stilove programiranja kao što su objektno-orijentisano, funkcionalno i proceduralno programiranje. Kreiran je tako da direktno interpretira komande u mašinske instrukcije. Tvorac Python programskog jezika je *Guido van Rossum* (1990).

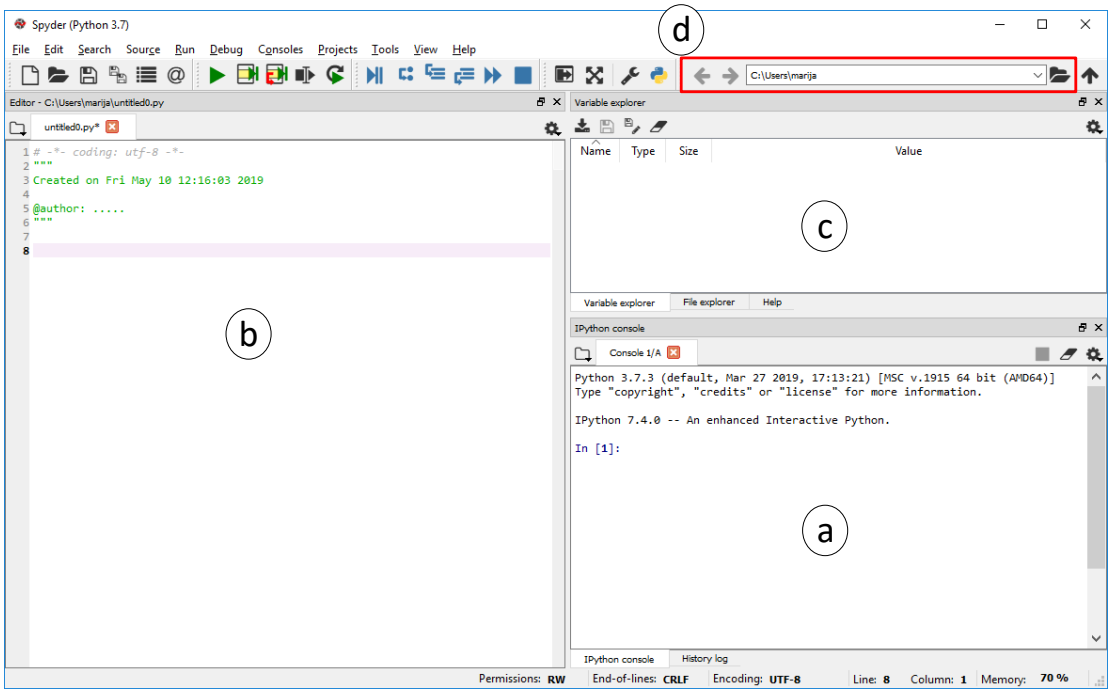
Postoje dve verzije Python programskog jezika (Python 2.x i 3.x). Između ove dve verzije postoji mali broj razlika (kao npr. funkcija *print*, deljenje celih brojeva…). Veliki broj funkcija koje se koriste za rad sa podacima u Python-u se nalazi u bibliotekama (modulima) koji ne dolaze uz osnovnu instalaciju. Postoje različita okruženja za Python programski jezik (*Spyder, PyCharm, Canopy…*).

Napomena: Na ovim časovima će se koristiti Spyder okruženje i verziju Python 3.7. Anaconda distribuciju zajedno sa Spyder okruženjem možete preuzeti na sledećem linku: https://www.anaconda.com/distribution/.

Pokrenuti *Spyder* programsko okruženje. Otvara se prozor kao na Sl. 12.1.1. Osnovna arhitektura ovog okruženja se sastoji od četiri glavna dela:

a) Konzola – služi za prikaz rezultata izvršavanja *.py* skripte, kao i izvršavanje komandi jednu po jednu. Kucanjem naredbi u konzoli se automatski dobija rezultat izvršavanja.
b) *Editor* – služi za pisanje *.py* skripti sa programskim kodom. Za razliku od pisanja koda u konzoli, pisanjem koda u skripti su lakše izmene koda, kao i dodavanje delova koda. Veza između korisnika i skripte se vrši preko konzole, o čemu će biti reči kasnije.
c) *Variable explorer* – sliži za pregled svih varijabli koje su se pojavile nakon izvršavanja koda; *File explorer* – služi za pregled svih datoteka koje se nalaze u trenutnom folderu; *Help* – služi za objašnjenja funkcija. Informacije o funkciji se dobijaju prilikom kucanja komande *help(funkcija)* u konzoli.
d) Putanja ka trenutnom folderu u kome se nalazimo. Ukoliko želimo da pozivamo neke spoljašnje datoteke (.txt, .xslx, .csv) u kodu, potrebno je da se putanja ka

toj datoteci slaže sa putanjom ka trenutnom folderu. U suportom će program prijaviti grešku kako ne može da nađе zadatu datoteku.



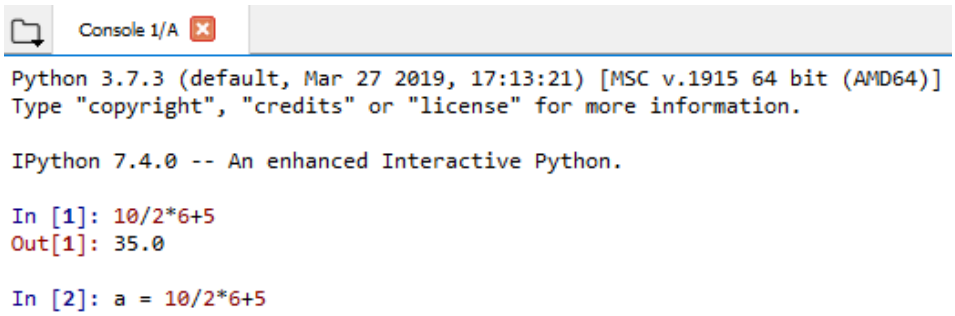
Sl. 12.1.1. *Spyder* programsko okruženje

12.2 Tipovi podataka u Python-u

U Python programskom jeziku nije potrebno navoditi tip promenljive, već on ima mogućnost da sam prepozna tip podatka. Osnovni tipovi podataka su:

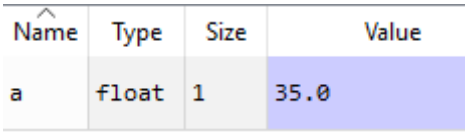
1. Logički tip promenljivih (*boolean*)
2. Brojevi – dele se u tri grupe: celi brojevi (*integer*), realni brojevi (*float*) i kompleksni brojevi (*complex*). Napomena: svi brojevi koji imaju u sebi decimalnu tačku se smatraju realnim brojevima, uključujući i slučaj da se nakon decimalne tačke nalazi 0 (npr. 15.0).
3. Sekvence – konačni uređeni skupovi koji mogu da se indeksiraju celim brojevima. U ovu grupu spadaju stringovi, liste i torke.

U Python konzoli se direktno može upisati neki matematički izraz ili izvršiti poziv neke funkcije. Rezultat izvršavanja komande se može, ali ne mora dodeliti nekoj promenljivoj. Ukoliko se rezultat dodeli promenljivoj, on se neće ispisati u konzoli (Sl. 12.2.1).



Sl. 12.2.1. Rezultat izvršavanja matematičkog izraza u konzoli programskog okruženja *Spyder*

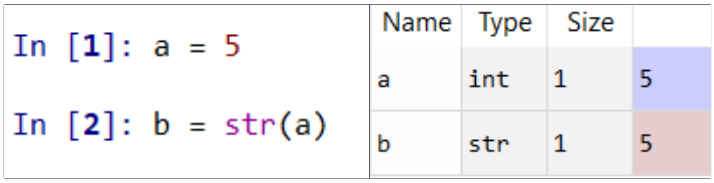
Otvoriti *Variable explorer*. Primetiti da se u njemu nalazi promenljiva ***a*** na spisku svih promenljivih (Sl. 12.2.2) i da joj je klasifikovana kao realni tip promenljive..



Name	Type	Size	Value
a	float	1	35.0

Sl. 12.2.2. Primer prikaza promenljive u *Variable explorer* prozoru

Kao rezultat izvršavanja funkcije *type(x)* se dobija tip promenljive *x*. Moguće je i menjanje tipa promenljiva. Funkcijom *cast(x)* promenljiva *x* prelazi u tip *cast* (Sl. 12.2.3).



```
In [1]: a = 5

In [2]: b = str(a)
```

Name	Type	Size	
a	int	1	5
b	str	1	5

Sl. 12.2.3. Primer promene tipa promenljive *a*

Zadatak 12.2.1. Učitati promenljive *x* i *y* u konzoli. Vrednost promenljive *x* neka bude *'zadatak'*, a promenljiva *y* je jednaka 17. U *Variable explorer-u* proveriti tip promenljivih *x* i *y*.

Logičke operacije:
Logičke operacije su definisane na operandima koji su tipa *boolean* (mogu da imaju vrednost *True* ili *False*). Rezultat izvršavanja je takođe *boolean*.

Tip operacije	**Simbol**
Logičko *i*	and
Logičko *ili*	or
Logička negacija	not

Operacije nad brojevima:
Nad brojevima je moguće primeniti klasične matematičke operacije:

Tip operacije	**Simbol**
Sabiranje	+
Oduzimanje	-
Množenje	*
Deljenje	/
Celobrojno deljenje	//
Ostatak pri deljenju	%
Stepenovanje	**

U Python verziji 2.x ukoliko su oba operanda celi brojevi prilikom deljenja dolazi do celobrojnog deljenja. U Python verziji 3.x to nije slučaj, već je potrebno eksplicitno definisati potrebu za celobrojnim deljenjem pomoću odgovarajućeg operatora.

Ostale matematičke funkcije (*sin, cos, sqrt…*) ne pripadaju osnovnom paketu, već je potrebno učitati dodatnu biblioteku.

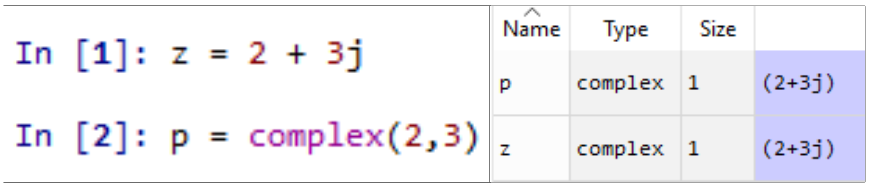
Zadatak 12.2.2. U konzoli učitati promenljive *b1 = 92*, a *b2 = 13*. Izračunati koliko puta se broj *b2* sadrži u broju *b1* i ostatak pri deljenju broja *b1* sa brojem *b2*.

Nad brojevima je moguće primeniti i operacije poređenja:

Tip operacije	Simbol
Jednako	==
Nije jednako	<>, !=
Veće od	>
Manje od	<
Veće ili jednako od	>=
Manje ili jednako od	<=

Kompleksni brojevi:
Kompleksni brojevi u Python programskom jeziku se mogu definisati na dva načina (Sl. 12.2.4):

i. Eksplicitno pomoću imaginarne jedinice *j*
ii. Pomoću funkcije *complex(realni_deo, imaginrni_deo)*



Sl. 12.2.4. Primeri definisanja kompleksnih brojeva

Osnovne funkcije koje se primenjuju nad kompleksnim brojevima su:

Tip operacije	Simbol
Realni deo	p.real
Imaginarni deo	p.imag
Moduo	abs(p)
Konjugovano-kompleksni broj	p.conjugate()

Zadatak 12.2.3. U konzoli definisati kompleksni broj *z = 11 + j7*. Izračunati moduo broja *z1* na dva načina: 1) pomoću funkcije *abs()* i 2) pomoću formule $\sqrt{real^2 + imag^2}$. Kvadratni koren izračunati kao stepenovanje sa brojem 0.5.

Liste:
Liste predstavljaju niz elemenata odvojenih zarezom koji se nalaze unutar uglastih zagrada (Sl. 12.2.5). Elementi liste mogu biti različitog tipa.

```
In [14]: lista = [2, 'etf', True, 3.5]

In [15]: type(lista)
Out[15]: list
```

Sl. 12.2.5. Primer kreiranja jedne liste

Elementima liste je moguće pristupiti indeksiranjem pomoću uglastih zagrada []. **Indeksiranje u Python-u počinje od 0!** Moguće je vršiti dodelu vrednosti nekom elementu u listi. Pristupanju elementu liste koji nije još definisan dovodi do greške.

Tip operacije	Simbol
Prvi element liste	lista[0]
i-ti element liste	lista[i]

Poslednji element liste	lista[-1]
Deo liste od *i*-tog do (*k-1)*-tog elementa	lista[i:k]
Deo liste od *i*-tog do poslednjeg elementa	Lista[i:]

Liste u Python-u se posmatraju kao objekti koje sadrže svoje podatke, metode i funkcije. Metodama i funkcijama se pristupa pomoći operatora **.**. Osnovne funkcije nad listama su:

Tip operacije	**Simbol**
Dužina liste	len(lista)
Dodavanje elementa na kraj liste	lista.append(element)
Uklanjanje elementa liste na određenom indeksu	del(lista[indeks])
Uklanja određeni element iz liste	lista.remove(element)
Sortiranje liste pri čemu se originalna lista menja	lista.sort()
Sortiranje liste pri čemu se originalna lista ne menja	sorted(lista)

Dve liste se mogu međusobno sabrati pomoću operatora +, pri čemu se tada jedna lista nadodaje na drugu. Takođe, jedna lista se može umnožiti nekoliko puta pomoću operatora * (Slika 12.2.6).

```
In [23]: lista1 = [1, 'etf', True, 3.5]

In [24]: lista2 = [False, 14.65]

In [25]: lista1 + lista2
Out[25]: [1, 'etf', True, 3.5, False, 14.65]

In [26]: lista2*3
Out[26]: [False, 14.65, False, 14.65, False, 14.65]
```

Sl. 12.2.6. Primena operacija sabiranja i množenja nad listama

Zadatak 12.2.4. U konzoli kreirati tri liste koje sadrže po dva elementa. Svaka lista predstavlja jedno teme trougla, a elementi liste su *x* i *y* koordinata temena. Izračunati stranice trougla pomoću formule $s = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. Izračunati površinu trougla pomoću formule $P = \sqrt{s(s-a)(s-b)(s-c)}$, gde je *s* poluobim. Kvadratni koren izračunati kao stepenovanje sa brojem 0.5.

U Python-u ime liste pokazuje na objekat te liste. Tako da ukoliko se kreira nova lista na osnovu već postojeće liste (*lista2 = lista1*) tada se samo kreira novi pokazivač ka istom objektu. To znači da sve pomene nad novom listom, utiču na promene stare liste. Korišćenjem naredbe *lista2 = lista1[:]* se kreira nov objekat, tj. vrši se kloniranje liste.

Zadatak 12.2.5. U konzoli kreirati listu *temperatura* koja sadrži vrednosti *hladno, toplo i vruce*. Kreirati listu *prognoza=temperatura.* Promeniti drugi element liste *prognoza* u *suncano*. Proveriti elemente liste *temperatra*. Kreirati listu *vreme=temperatura[:]*. Promeniti prvi element liste *vreme* u *oblacno*. Pogledati elemente liste *temperatura*.

Stringovi:
Stringovi predstavljaju niz simbola koji se nalazi između navodnika (“) ili apostrofa (’). Dva stringa se mogu međusobno spojiti pomoću operatora +. Takođe, string se može umnožiti nekoliko puta pomoću operatora * (Sl. 12.2.7).

```
In [2]: string = 'Zdravo svete!'

In [5]: s = 'ba' + 2*'na'      In [3]: pozdrav = string + ' :)'

In [6]: s                      In [4]: pozdrav
Out[6]: 'banana'               Out[4]: 'Zdravo svete! :)'
```

Sl. 12.2.7. Primer manipulacija sa stringovima

Pomoću funkcije *input(tekst_poruke)* je moguće iz konzole pročitati poruku korisnika. Najpre se korisniku ispisuje *tekst_poruke*, a zatim korisnik treba da upiše njegovu poruku. Korisnikova poruka se čuva u odgovarajućoj promenljivoj kao tip *string* (Sl. 12.2.8). Funkcija *input* blokira izvršavanja ostatka koda sve dok korisnik ne unese neku poruku. Moguće je ispisati rezultat izvršavanja koda korisniku pomoću funkcije *print(poruka)*.

```
In [32]: poruka = input('Unesite zeljeni tekst. ')

Unesite zeljeni tekst. praktikum

In [33]: print(poruka)
praktikum
```

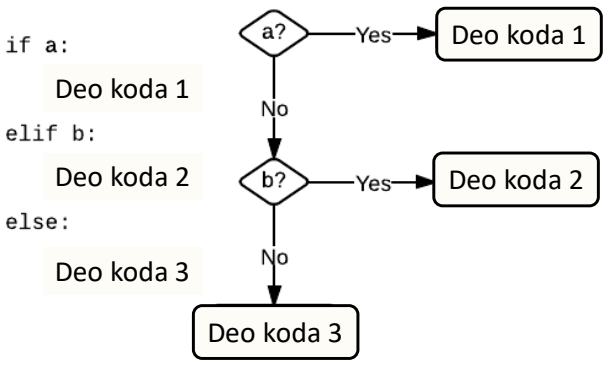
Sl. 12.2.8. Primer korišćenja *input* funkcije

Zadatak 12.2.6. Zatražiti korisniku da unese željenu vrednost površine trougla. Poveriti da li je željena površina veća od površine izračunate u zadatku 12.2.4. *Napomena*: obratiti pažnju da funkcija *input* vraća string i da je potrebno izvršiti konverziju tipa podatka.

12.3 Strukture i petlje

Python programski jezik razdvaja blokove pomoću uvlačenja teksta, za razliku od drugih programskih jezika koji koriste ključne reči ili vitičaste zagrade. Na taj način se povećava čitljivost samog koda. Uvlačenje tekste se primenjuje kod *while* i *for* petlje, *if* strukture, definisanja funkcija…

IF struktura:
IF struktura određuje koji deo koda se izvršava u zavisnosti od uslova (Sl. 12.3.1). Uslov je tipa *boolean*.



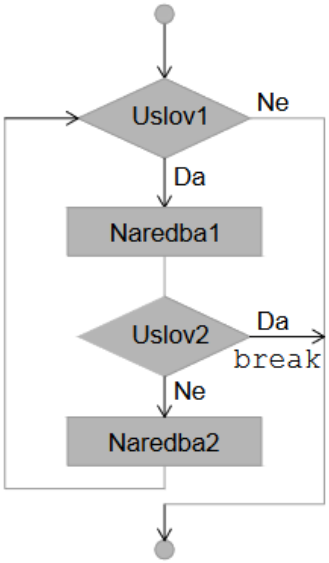
Sl. 12.3.1. Princip funkcionisanja *if* strukture

Zadatak 12.3.1. Potrebno je napisati program kojina osnovu kategorije avionske karte određuje da li korisnik ima prtljag koji je teži od dozvoljenog.

1. U editoru otvoriti novu *.py* skriptu (*File>>New File*).
2. Korisnik preko konzole treba da unese kategoriju svoje karte (funkcije *input*): 1) K1, dozvoljen prtljag do 8 kg, 2) K2, dozvoljen prtljag do 15 kg i 3) K3, dozvoljen prtljag do 20 kg.
3. Nakon unosa kategorije, od korisnika treba da se traži da unese i težinu njegovog kofera.
4. Na osnovu kategorije avionske karte, ispitati da li je težina kofera veća od dozvoljene. Ukoliko jeste ispisati korisniku poruku u konzoli (funkcija *print*).
5. Sačuvati skriptu pod imenom 12_03_01.py.
6. Pokrenuti i testirati program. Python skripta se pokreće iz padajućeg menija *Run>>Run* ili prečicom F5 na tastaturi. Moguće je pokretanje i klikom na  .

WHILE petlja:

WHILE petlja služi za ponavljanje dela koda sve dok je ispunjen uslov (Sl. 12.3.2). Uslov je tipa *boolean*. Petlja može da se prekine naredbom *break*, čak i ako je i dalje ispunjen uslov.



Sl. 12.3.2. Tok izvršavanja *while* petlje. Petlja će se završiti ili ukoliko je nije ispunjen *uslov1* ili ukoliko dođe do *break*-a.

Zadatak 12.3.2. Napisati program u kome korisnik preko tastature unosi niz pozitivnih brojeva. Korisnik treba da unosi broj po broj, sve dok ne unese karakter 'x'. Uneti brojevi se čuvaju u listi. Na kraju izvršavanja programa potrebno je ispisati najveći broj u listi.

1. U editoru otvoriti novu *.py* skriptu (*File>>New File*).
2. Dodati promenljive **broj** sa početnom vrednošću '0' i **lista** sa početnom vrednošću []. Pomoću praznih uglastih zagrada je kreirana prazna lista.
3. Kreirati *while* petlju koja će se ponavljati sve dok promenljiva *broj* ne postane 'x'. Unutar petlje treba konvertovati broj u tip *float*, a zatim ga treba dodati u listu (funkcija *append*) i pitati korisnika da unese novi broj (pomoću funkcije *input*).

4. Nakon završetka *while* petlje potrebno je pronaći najveći broj iz liste. Jedan od načina je pomoću funkcije *max()*.
5. Pomoću funkcije *print* ispisati korisniku koji je najveći broj uneo.
6. Sačuvati skriptu pod imenom 12_03_02.py.
7. Pokrenuti i testirati program.

FOR petlja:
FOR petlja se za razliku od *while* petlje ne izvršava sve dok je ispunjen zadati uslov, već dok god ima elemenata u nekoj datoj sekvenci (Sl. 12.3.3). *For* petlja se takođe može zaustaviti pomoću naredbe *break*.

```
for n in range(5):
    print(n)
```

Sl. 12.3.3. Primer korišćenja *for* petlje

For petlja može da vrši iteracije na više načina. Najčešće se koristi funkcija *range()* koja generiše niz brojeva u zadatom opsegu. Sekvenca može biti i u rastućem i u opadajućem poretku.

Poziv funkcije	**Generisana sekvenca**
range(n)	0, 1, 2,…, n-1
range(m, n)	m, m+1, m+2,…, n-1
range(m, n, step)	m, m+step, m+2*step,…, n-step

Zadatak 12.3.3. Napisati program koji ispisuje *n* parnih brojeva.
1. U editoru otvoriti novu *.py* skriptu (*File>>New File*).
2. Od korisnika tražiti da unese broj *n*.
3. Pomoću *for* petlje ispisati sve parne brojeve u konzoli (funkcija *print*). Vršiti iteracije kroz petlju pomoću funkcije *range()*.
4. Sačuvati program pod nazivom 12_03_03.py.
5. Pokrenuti i testirati program.

For petlja može vršiti i iteracije kroz elemente liste, stringa ili niza (Sl. 12.3.4).

```
L = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

total = 0
for i in L:
    total += i

print(total)
```

Sl. 12.3.4. Primer *for* petlje koja prolazi kroz sve elemente liste *L* i računa zbir elemenata liste.

12.4 Biblioteke

U *Python* programskom paketu se nalaze samo osnovne funkcije za rad sa podacima. Ukoliko je potrebno vršiti naprednije manipulacije nad promenljivima potrebno je učitati odgovarajuću biblioteku. *Python* poseduje veliki broj biblioteka koje su već oformljene za različite probleme. Najčešće se koriste biblioteke *numpy, matplotlib, scipy, pandas…*

Biblioteke se učitavaju pomoću ključne reči *import*. Postoje različiti načini učitavanja biblioteke:

Način uključivanja	Pozivanje funkcija iz biblioteke	Objašnjenje
import numpy	numpy.sin(numpy.pi)	Potrebno je uneti celo ime modula
import numpy as np	np.sin(np.pi)	Funkcije se pozivaju korišćenjem skraćenice np
from numpy import *	sin(pi)	Funkcije postaju deo okruženja pa nije potrebno koristiti ime modula

Dodavanje svih funkcija neke biblioteke kao deo podrazumevanog prostora se ne preporučuje, jer može doći do mešanja funkcija sa istim nazivom iz različitih biblioteka.

Ukoliko biblioteka nije instalirana u okruženju može se instalirati iz komandnog prozora. Iz Start menija otvoriti *Anaconda prompt*. Kucanjem komande *pip install ime_biblioteke* i pritiskom tastera *enter* će se instalirati potrebna biblioteka.

Zadatak 12.4.1. Napisati program koji ispisuje vrednost funkcije $y = e^x$ za $x \in [a, b]$.

1. U editoru otvoriti novu *.py* skriptu (*File>>New File*).
2. Učitati biblioteku *math* pomoću naredbe *import*
3. Pomoću *for* petlje proći kroz sve vrednosti između *a* i *b*. Samostalno izabrati *a*, *b* i korak (funkcija *range*).
4. Izračunati vrednost funkcije *y* za trenutno *x* pomoću funkcije *exp* iz biblioteke *math*. U svakoj iteraciji petlje prikazati rezultat korisniku.
5. Sačuvati program pod nazivom 12_04_01.py.
6. Pokrenuti i testirati program.

Numpy biblioteka:

Numpy biblioteka je kreirana za tzv. *Scientific computing*. Omogućava vrlo efikasan rad sa višedimenzionalnim nizovima. Nizovi u *Python*-u podsećaju na liste, ali svi elementi niza moraju biti istog tipa i indeksiranje je moguće samo nenegativnim brojevima.

Nizovi se mogu kreirati na više načina:

Tip operacije	Simbol
Kreiranje niza prvog reda	np.array([1, 2, 3])
Kreiranje niza ispunjenog nulama dimenzija *m*x*n*	np.zeros((m, n))
Kreiranje niza ispunjenog jedinicama dimenzija *m*x*n*	np.ones((m, n))
Kreiranje niza ispunjenog brojem *b* dimenzija *m*x*n*	np.full((m, n), b)
Kreira niz brojeva u opsegu *[a, b]*, sa korakom *k*	np.arange(a, b, k)
Kreira *n* brojeva u opsegu *[a, b]*	np.linspace(a, b, n)

Indeksiranje elemenata niza se vrši pomoću uglastih zagrada (Sl. 12.4.1). U primeru je kreirana matrica *a* kao dvodimenzionalni niz.

```
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

element = a[0, 1]    # Izdvajanje pojedniačnog elemnta
                     # iz prvog reda i druge kolone

# Izdvajanje podniza koji sadrži prva dva reda
# i drugu i treću kolonu; rezultat ima oblik (2, 2)
podniz = a[:2, 1:3]

red = a[1, :]        # Izdvaja samo drugi red matrice a
kolona = a[:, 2]     # Izdvaja samo treću kolonu matrice a
```

Sl. 12.4.1. Primer izdvajanja elemenata niza

Numpy biblioteka koristi osnovne matematičke operacije nad nizovima ili matricama *elementwise*. To znači da primenjuje neku matematičku operaciju na svaki član posebno.

Zadatak 12.4.2. Napisati program koji računa zbir svih elemenata matrice, kao i srednju vrednost svake kolone.

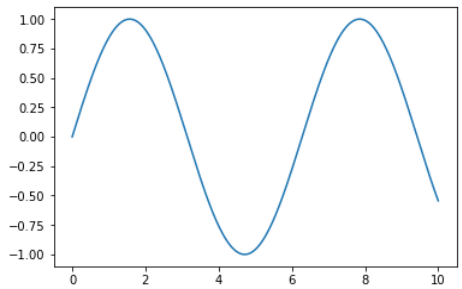
$$M = \begin{bmatrix} 10 & 5 & 18 & 2 \\ 9 & 17 & 3 & 6 \end{bmatrix}$$

1. U editoru otvoriti novu *.py* skriptu (*File>>New File*).
2. Učitati biblioteku *numpy* pomoću naredbe *import*
3. Kreirati matricu M kao na Sl. 12.4.1.
4. Pomoću funkcije *sum* iz biblioteke *numpy* izračunati sumu svih elemenata matrice M.
5. Pomoću funkcije *mean* iz biblioteke *numpy* izračunati srednju vrednost svake kolone. Funkcija *mean* ima obavezan argument (promenljivu čija se srednja vrednost računa) i opcioni argument (osu po kojoj računa, 0 – uzima svaku kolonu posebnu, 1 – uzima svaki red posebno).
6. Sačuvati program pod nazivom 12_04_02.py.
7. Pokrenuti i testirati program.

Matplotlib biblioteka:
Matplotlib biblioteka se najčešće koristi za grafički prikaz podataka. Da bi podaci mogli da se grafički prikažu potrebno je definisati vektor x-koordinata i vektor y-koordinata (za 2D slučaj). Prilikom prikaza grafika vektori *x* i *y* moraju biti istih dimenzija.

Naredba *plot* se koristi za crtanje kontinualnih signala. Prvi argument ove funkcije je raspodela *x* ose, a drugi predstavlja vrednosti na *y* osi. Prostor između dve poznate tačke se na grafiku aproksimira pravom linijom. To znači da što je „finija“ podela x-ose to će grafik izgledati „glađe“. Primer izgleda grafika dobijenog pomoću *plot* funkcije je prikazan na Sl. 12.4.2.

```
import matplotlib.pylab as plt
import numpy as np

t = np.linspace(0,10,2000)

plt.figure()
plt.plot(t, np.sin(t))
```



Sl. 12.4.2. Primer generisanja grafika pomoću naredbe *plot* (levo) i odgovarajući grafik (desno)

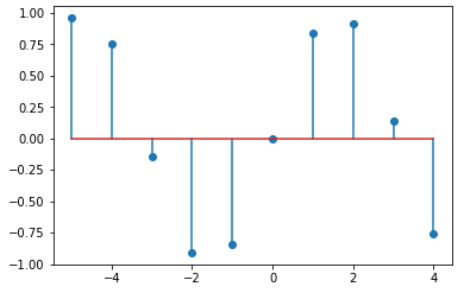
Naredba *figure* otvara novi prostor za crtanje grafika. Ukoliko se ne pozove ova naredba signali na grafiku će se prikazivati jedan preko drugog.

Zadatak 12.4.3. Nacrtati funkciju $x(t) = cos(2\pi f_1 t) + 5sin(2\pi f_2 t)$, ako je f_1=10 Hz i f_2=7 Hz, a t je u rasponu od 0 do 1 s.

1. U editoru otvoriti novu *.py* skriptu (*File>>New File*).
2. Učitati biblioteke *numpy* i *matplotlib.pyplot* pomoću naredbe *import*
3. Kreirati vremensku osu *t* u opsegu od 0 do 1 s sa odgovarajućem brojem tačaka (koristiti funkciju *linspace*).
4. Formirati signal *x(t)* i prikatati ga na grafiku pomoću funkcije *plot*.
5. Sačuvati program pod nazivom 12_04_03.py.
6. Pokrenuti i testirati program.

Naredba *stem* se koristi za crtanje diskretnih signala. Pozivanje funkcije je isto kao i kod naredbe *plot*, samo što se susedne tačke ne spajaju. Primer izgleda grafika dobijenog pomoću *stem* funkcije je prikazan na Sl. 12.4.3.

```
import matplotlib.pylab as plt
import numpy as np

n = np.arange(-5, 5, 1)

plt.figure()
plt.stem(n, np.sin(n))
```



Sl. 12.4.3. Primer generisanja grafika pomoću naredbe *stem* (levo) i odgovarajući grafik (desno)

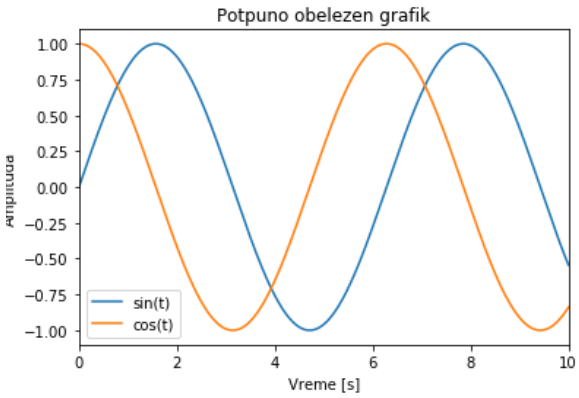
Zadatak 12.4.4. Nacrtati funkciju $y(n)$ pomoću naredbe *stem*. Promenljive n i y definisati kao *numpy* nizove.

$$y(n) = \begin{cases} 1, & n = 1 \\ 5, & n = 2 \\ 6.2, & n = 3 \\ -9.5 & n = 4 \\ 11, & n = 5 \\ -3, & n = 6 \\ 0, & ostalo \end{cases}$$

1. U skripti 12_04_03.py dodati formiranje signala *y* i *n*.
2. Otvoriti novu figuru. Pomoću naredbe *stem* prikazati signal *y* u zavisnosti od *n*.
3. Pokrenuti i testirati program.

Ova biblioteka omogućava obeležavanje grafika u vidu definisanja naziva osa (*plt.xlabel*, *plt.ylabel*), dodavanje naziva grafika (*plt.title*), definisanje granica *x* i *y* ose u kojima će se prikazivati signal (*plt.xlim*, *plt.ylim*), dodavanje mreže (*plt.grid*)…Ukoliko se na istom grafiku prikazuje više signala potrebno je dodati i legendu. Funkcija *legend* treba da ima onoliko argumenata koliko ima različitih signala na grafiku. Primer potpuno definisanog grafika je dat na Sl. 12.4.4.

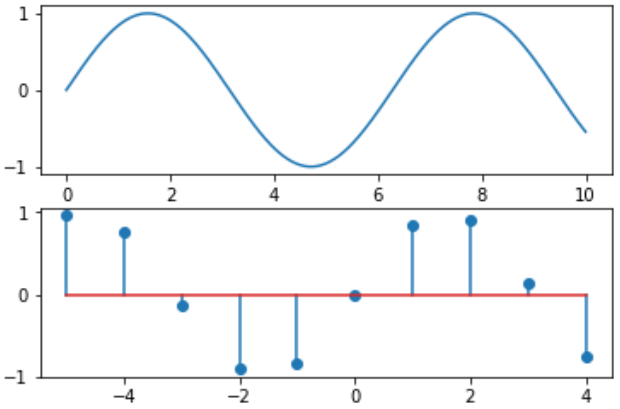
```
import matplotlib.pylab as plt
import numpy as np

t = np.linspace(0,10,2000)

plt.figure()
plt.plot(t, np.sin(t))
plt.plot(t, np.cos(t))
plt.legend(('sin(t)','cos(t)'))
plt.xlabel('Vreme [s]')
plt.ylabel('Amplituda')
plt.title('Potpuno obelezen grafik')
plt.xlim((0,10))
```



Sl. 12.4.4. Primer generisanja obeleženog grafika (levo) i odgovarajući grafik (desno)

Pomoću naredbe *subplot* moguće je prikazati više signala na posebnim graficima u okviru jedne figure. Naredba *plt.subplot(m, n, g)* generiše matricu grafika u jednoj figuri dimenzija *mxn* i vraća pokazivač na polje *g*. Primer pozivanja funkcije *subplot* je prikazan na Sl. 12.4.5. Korišćenjem funkcije *subplot* potrebno je za svaki grafik obeležiti grafik.

```
import matplotlib.pylab as plt
import numpy as np

t = np.linspace(0,10,2000)

plt.figure()
plt.subplot(2,1,1)
plt.plot(t, np.sin(t))

n = np.arange(-5, 5, 1)

plt.subplot(2,1,2)
plt.stem(n, np.sin(n))
```



Sl. 12.4.4. Primer generisanja pomoću funkcije *subplot* (levo) i odgovarajući grafik (desno)

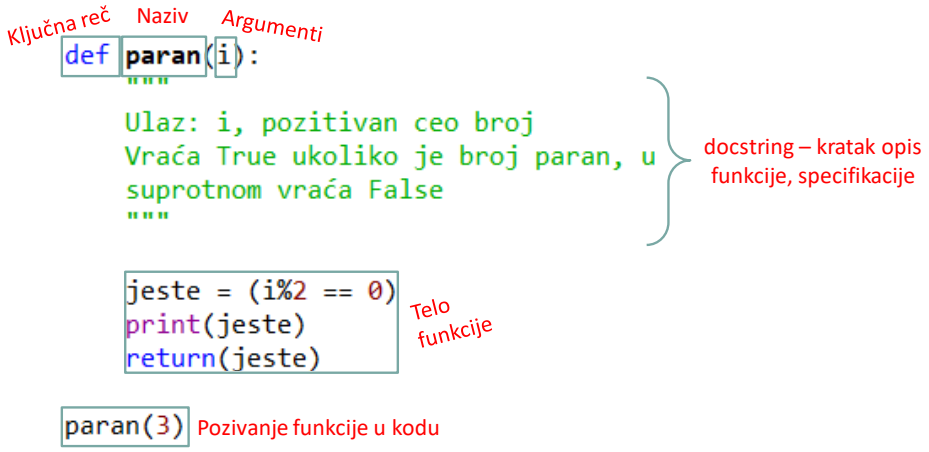
Zadatak 12.4.5. Nacrtati grafike funkcija *sin(t), cos(t), sin(3t)* i *cos(3t)*

4. U editoru otvoriti novu *.py* skriptu (*File>>New File*).
5. Učitati biblioteke *numpy* i *matplotlib.pyplot* pomoću naredbe *import*
6. Kreirati vremensku osu *t* u opsegu od 0 do 15 s sa odgovarajućem brojem tačaka (koristiti funkciju *linspace*).
7. Pomoću funkcije *subplot* u jednoj figuri prikazati signale *sin(t)* i *sin(3t)* na jednom grafiku, a signale *cos(t)* i *cos(3t)* na drugom grafiku.
8. Obeležiti ose, dodati legendu, dodati naslov grafika.
9. Sačuvati program pod nazivom 12_04_05.py.
10. Pokrenuti i testirati program.

12.5 Funkcije

Funkcije sadrže delove koda koji se više puta koriste u glavnom kodu. Funkcija započinje sa ključnom reči **def** nakon koje se definiše naziv funkcije. Svaka funkcija može, ali ne mora imati ulazne argumente. Zatim, može se definisati *docstring* koji

sadrži kratak opis funkcije. Na kraju, nalazi se telo funkcije. Prikaz definisanja funkcije je dat na Sl. 12.5.1.



Sl. 12.5.1. Definisanje funkcije *paran* koja proverava da li je dati broj paran.

Pozivanje funkcije u kodu se vrši navođenjem njenog imena i potrebnih argumenata.

Zadatak 12.5.1. Napisati funkciju koja računa elemente Fibonačijevog niza. Fibonačijev niz počinje brojevima 0 i 1, a svaki sledeći član se računa po formuli:

$$f_n = f_{n-1} + f_{n-2}, n = 3{,}4{,}5 \ldots$$

Korisnik treba da zada broj elemenata Fibonačijevog niza koji treba da se izračunaju.

1. U editoru otvoriti novu *.py* skriptu (*File>>New File*).
2. Definisati funkciju *fibonacci* sa jedanim ulaznim argumentom *n* koji označava broj elemenata. U telu funkcije je potrebno definisati promenljivu *niz* tipa liste sa početnom vrednošću [0, 1]. U svakoj iteracije *for* petlje izračunati novi član niza i dodati ga u listu *niz* (funkcija *append*). Nakon završetka *for* petlje vratiti listu *niz* u glavni program (ključna reč *return*).
3. U glavnom delu programa zatražiti od korisnika da unese broj elemenata niza (funkcija *input*), a zatim iskoristiti tu vrednost pri pozivu funkcije.
4. Sačuvati program pod nazivom 12_05_01.py.
5. Pokrenuti i testirati program

Argumenti funkcije se dele na obavezne i opcione argumente. Obavezne argumente je potrebno definisati prilikom poziva funkcije, tj. dodeliti im neku vrednost. Ukoliko se prilikom definisanja funkcije nekom argumentu dodeli podrazumevana vrednost tada on postaje opcioni argument. To znači da korisnik može da pozove funkciju bez dodele vrednosti tom argumentu, i u telu funkcije će se koristiti podrazumevana vrednost.

Posmatrajmo primer funkcije koja ima zadatak da izračuna PDV na zadatu cenu. Ulazni parametri funkcije su zadata cena *x* i stopa PDV-a *stopa*. Argumentu *stopa* je dodeljena podrazumevana vrednost 20 (Sl.12.5.2).

```
def pdv(x, stopa = 20):
    """
    x je obavezan argument
    stopa je opcioni argument
    """

    return x*(1 + stopa/100)
```

Sl. 12.5.1. Definisanje funkcije *pdv*

Prilikom pozivanja funkcije može se iskoristi ime argumenta. Na taj način ukoliko postoji veliki broj opcionih argumenata ne moraju se navoditi vrednosti za sve argumente, već samo za one čiju podrazumevanu vrednost želimo da promenimo. Ovo je prednosti u odnosu na druge programske jezike. Sam poziv funkcije se može izvršiti na više načina:

Poziv funkcije	**Objašnjenje**
pdv(200)	Početna cena *x* je 200, a *stopa* uzima podrazumevanu vrednost
pdv(200, 10)	Početna cena *x* je 200, a *stopa* je 10
pdv(x = 200, stopa = 10)	Funkcija se poziva preko imena argumenata, redosled navođenja argumenata ne mora biti kao pri definiciji funkcije
pdv(200, stopa = 10)	Ukoliko se neka vrednost argumenata navede bez imena, odmah se dodeljuje obaveznom argumentu. Ukoliko ima više obaveznih argumenata, vrednosti se dodeljuju redom, kao u definiciji funkcije
pdv(200, x = 5)	Dolazi do greške zato što se argumentu *x* dva puta dodeljuje neka vrednost

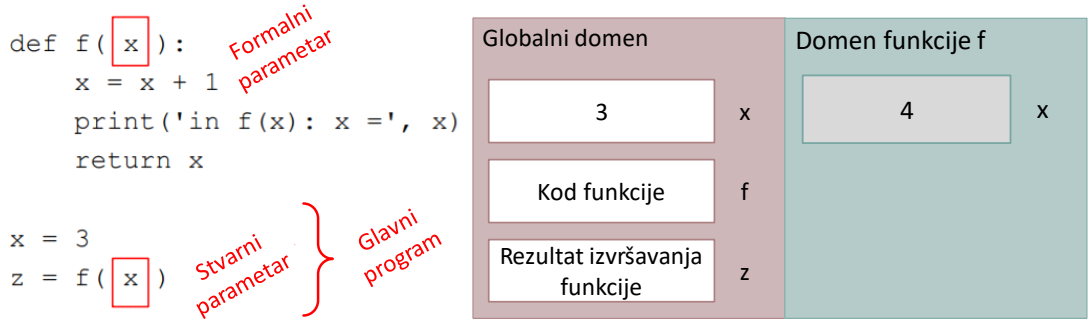
Funkcija može da vraća i više vrednosti tako što se one grupišu pomoću običnih zagrada: *return (a, b)*. Prilikom poziva funkcije potrebno je smestiti rezultat u onoliko promenljivih koliko se vraća iz funkcije (Sl. 12.5.2).

```
import numpy as np

def osobine(niz):
    srednja_vrednost = np.mean(niz)
    suma = np.sum(niz)

    return (srednja_vrednost, suma)

n = np.array([1,5,4,3,2,9,8])
srvr, s = osobine(n)
```

Sl. 12.5.2. Vraćanje više vrednosti iz funkcije

Prilikom pozivanja funkcije javljaju se stvarni i formalni parametri. Stvarni parametri pripadaju globalnom prostoru i oni se nalaze u glavnom programu. Formalni parametri se vezuju za vrednost stvarnih parametara prilikom poziva funkcije. Unutar funkcije se može pristupiti promenljivama koje su definisane u glavnom delu programa. Prilikom svakog pozivanja funkcije definiše se novi domen (Sl. 12.5.3). Obratiti pažnju da

promenljiva *x* postoji i u glavnom kodu i u funkciji, ali ima različite vrednosti. Ukoliko dođe do preklapanja imena, funkcija ima mogućnost da utiče samo na izmenu promenljive *x* koja je definisana unutar nje.



Sl. 12.5.3. Domen važenja varijabli

Zadatak 12.5.2. Napisati funkciju koja računa površinu geometrijskog oblika: kvadrat, pravougaonik ili trougao. Funkcija treba da ima tri argumenta: x (*float*), y (*float*) i tip oblika (*string*). Tip oblika treba da ima podrazumevanu vrednost 'k' za kvadrat, a y ima podrazumevanu vrednost 0. U zavisnosti od tipa predmeta treba da se primeni drugačija formula za površinu:

Kvadrat	Pravougaonik	Trougao
$P = x^2$	$P = x \cdot y$	$P = \frac{x \cdot y}{2}$

1. U editoru otvoriti novu *.py* skriptu (*File>>New File*).
2. Definisati funkciju *povrsina* sa tri ulazna argumenta kao što je opisao u postavci zadatka.
3. Telo funkcije treba da sadrži računanje površine u zavisnosti od tipa geometrijskog oblika. U glavni program se vraća izračunata površina.
4. U glavnom programu pozvati funkciju i ispisati rezultati izvršavanja korisniku.
5. Sačuvati program pod nazivom 12_05_02.py.
6. Pokrenuti i testirati program za različite kombinacije ulaznih parametara funkcije

12.6 Rad sa datotekama

U zavisnosti od tipa datoteke koju želimo da učitamo u naš program postoje različite biblioteke sa funkcijama za rad sa datotekama. Podaci za obradu su najčešće smešteni u tekstualnim datotekama ili *.csv* datotekama (*comma separated values*).

Bitna stvar na koju treba obratiti pažnju prilikom učitavanja datoteka jeste da trenutni folder u kome se *Spyder* nalazi mora da se poklopi sa folderom u kom se nalazi datoteka.

Prilikom definisanja imena datoteke iz koje čitamo podatke potrebno je definisati i ekstenziju, npr. *ime_datoteke = 'proba.txt'*.

Rad sa tekstualnim datotekama:
Funkcija za učitavanje tekstualne datoteke se nalazi osnovnom paketu.

Poziv funkcije	Objašnjenje
d = open('ime_datoteke', tip)	Otvaranje datoteke. Parametar *tip* može imati vrednosti 'r' (read), 'w' (write) ili 'a' (add). U zavisnosti od tipa, iz datoteke može da se čita sadržaj, da se upisuje sadržaj u nju ili da se doda sadržaj u nju. Podrazumevana vrednost je 'r'.
d.read()	Čitanje celog sadržaja datoteke. Kad se jednom pročita sadržaj, pri ponovnom pozivanju ove funkcije vraća se prazna datoteka. Rezultat ove funkcije je tipa *string*.
d.readline()	Čitanje samo jednog reda tekstualne datoteke. Kada se opet pozove ova funkcija vraća se sledeći red teksta, sve dok se ne pročitaju svi redovi. Rezultat koji vraća ova funkcija je tipa *string*.
d.readlines()	Čitanje svih redova tekstualne datoteke. Rezultat koji vraća ova funkcija je lista stringova, pri čemu svaki element liste odgovara jednom redu teksta.
d.write()	Upisivanje sadržaja u datoteku.
d.close()	Zatvarfanje datoteke. Nakon zatvaranja datoteke nije moguće vršiti nikakve manipulacije sa njom.

Kroz datoteku je moguće proći pomoću *for* petlje. U svakoj iteraciji *for* petlje se čita novi red u datoteci, sve dok se ne dođe do kraja datoteke (Sl.12.6.1).

```
In [45]: d = open('proba.txt', 'r')

In [46]: for line in d:
    ...:     print(line)
    ...:
What do you want from me?

Why don't you run from me?

What are you wondering?

What do you know?
```

Sl. 12.6.1. Čitanje teksta iz datoteke

Zadatak 12.6.1. Napisati program kopira sadržaj liniju po liniju iz jedne tekstualne datoteke u drugu.

1. U editoru otvoriti novu *.py* skriptu (*File>>New File*).
2. Podesiti putanju trenutnog foldera u *Spyder* okruženju tako da se slaže sa putanjom ka datotekama.
3. Otvoriti datoteku iz koje će se čitati sadržaj. Otvoriti datoteku u koju će se upisivati sadržaj.

4. Svaki red iz ulazne datoteke kopirati u jedan red u izlaznoj datoteci (koristiti *for* petlju).
5. Po završetku kopiranja, zatvoriti obe datoteke.
6. Sačuvati program pod nazivom 12_06_01.py.
7. Pokrenuti i testirati program za ulaznu tekstualnu datoteku koju samostalno kreirate.

Ukoliko .txt datoteka sadrži samo numeričke podatke može se učitati na lakši način pomoću funkcije *loadtxt* iz biblioteke *numpy* (Sl.12.6.2). Rezultat izvršavanja ove funkcije je tipa *float* ili *numpy* niz *float* vrednosti.

```
In [5]: import numpy as np

In [6]: a = np.loadtxt('datoteka.txt')

In [7]: a
Out[7]:
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  6.,  7.,  8.],
       [ 9., 10., 11., 12.]])

In [8]: type(a)
Out[8]: numpy.ndarray
```

Sl. 12.6.2. Primer čitanja podataka iz .txt datoteke pomoću funkcije *loadtxt*

Zadatak 12.6.2. Napisati program koji učitava signal iz datoteke EKG.txt i prikazuje ga na grafiku. U datoteci se nalaze odbirci EKG signala, koji su odabirani sa frekvencijom odabiranja od 100 Hz.

1. U editoru otvoriti novu *.py* skriptu (*File>>New File*).
2. Podesiti putanju trenutnog foldera u *Spyder* okruženju tako da se slaže sa putanjom ka datoteci.
3. Učitati biblioteke *numpy* i *matplotlib.pyplot*
4. Učitati sadržaj datoteke pomoću funkcije *loadtxt*. Kreirati promenljivu *fs* = 100. Kreirati vremensku osu pomoću funkcije *arange*. Vremenska osa treba da ima vrednosti od 0 do *len(signal)/fs* sa korakom 1/fs.
5. Na grafiku prikazati EKG signal u zavisnosti od vremena. Obeležiti ose grafika i dodati naslov grafika.
6. Sačuvati program pod nazivom 12_06_02.py.
7. Pokrenuti i testirati program.

Rad sa .csv datotekama:

Ponekad se podaci osim u .txt datoteka čuvaju u vidu *Excel* tabele sa ekstenzijom .csv. Funkcije za učitavanje .csv datoteka se nalaze u biblioteci *pandas*.

Datoteke sa ekstenzijom *.csv* u sebi sadrže podatke koji su najčešće razdvojene zarezom. Funkcija koja omogućava čitanje podataka se zove *read_csv*. Ova funkcija ima veliki broj argumenata. Najznačajniji argumenti ove funkcije su *filepath* (putanja do datoteke iz koje želimo da čitamo podatke) i *sep* (separator između kolona. Podrazumevana vrednost je ',').

Kao rezultat, ova funkcija vraća *data frame*. *Data frame* tip podataka je sličan kao matrice, samo što različite kolone mogu imati različite tipove promenljivih, ali unutar jedne kolone svi podaci moraju biti istog tipa. Takođe, kolone *data frame*-a mogu imati naziv preko koga je moguće pristupiti članovima te kolone pomoću operatora **.**

Primer korišćenja ove funkcije je dat na Sl. 15.6.3.

```
import pandas

df = pandas.read_csv('analiza.csv', sep=',')
print(df[['ID']])
pol = df.Pol
```

Sl. 12.6.2. Primer čitanja podataka iz .csv datoteke pomoću funkcije *read_csv*

Zadatak 12.6.3. Napisati program koji učitava signale iz datoteke GSR.csv. Izračunati srednju vrednost i standardnu devijaciju svakog od četiri GSR signala i prikazati podatke na grafiku u vidu *error bar*-a.

1. U editoru otvoriti novu *.py* skriptu (*File>>New File*).
2. Podesiti putanju trenutnog foldera u *Spyder* okruženju tako da se slaže sa putanjom ka datoteci.
3. Učitati biblioteke *numpy, matplotlib.pyplot* i *pandas*
4. Učitati sadržaj datoteke pomoću funkcije *read_csv*. Nakon učitavanja dobijeni *data frame* ima 4 kolone koje predstavljaju različite GSR signale.
5. Pomoću funkcije *mean* iz biblioteke *numpy* izračunati srednju vrednost svake kolone *data frame*-a. Smestiti rezultat u promenljivu *m*.
6. Pomoću funkcije *std* iz biblioteke *numpy* izračunati standardnu devijaciju svake kolone *data frame*-a. Smestiti rezultat u promenljivu *sd*.
7. Otvoriti novu figuru. Iz biblioteke *matplotlib.pyplot* iskoristiti funkciju *errorbar* za prikaz greške. *Error bar* je pogodan za vizualizaciju podataka i rezultata u smislu odstupanja od srednje vrednosti. Ova funkcija kao ulazne argumente prima podelu x-ose, vrednosti na y-osi i grešku za svaki odbirak po y-osi. Formirati x-osu tako da ima vrednosti od 1 do 4, za svaki od četiri GSR signala. Argument *y* treba da bude jednak *m*, a argument *yerr* je jednak *sd*. Poziv funkcije je plt.errorbar(x, m, yerr=sd). Moguće je dodati opcioni argument *fmt* sa vrednošću '.' kako se ne bi spojile tačke na grafiku.
8. Obeležiti ose na grafiku i dodati naslov.
9. Sačuvati program pod nazivom 12_06_03.py.
10. Pokrenuti i testirati program

Zadatak 12.6.4. Datoteka *temperatura.csv* sadrži srednju temperaturu u toku godine za vremenski period od 1880. do 2016. godine za dva izvora. Na istom grafiku prikazati zavisnost srednje temperature od godine za oba izvora. Obeležiti ose grafika na adekvatan način i dodati legendu. Izračunati srednju vrenost i standardnu devijaciju za oba izvora posebno. Prikazati rezultat pomoću *error bar*-a.