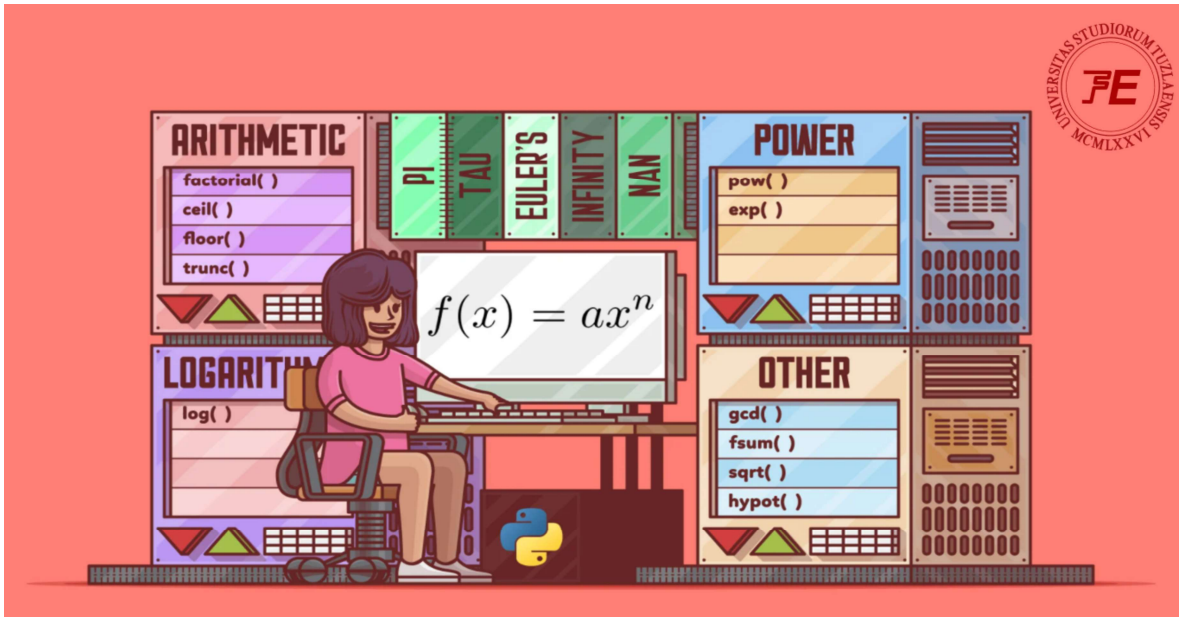
UNIVERZITET U TUZLI

FAKULTET ELEKTROTEHNIKE

Python tutoriali

Signali i sistemi

Emina Bajrektarević, Amera Sinanović



Tuzla,Septembar 2020. godine

Python Tutorijal 5

- **Moduli i paketi**
- **Math modul**
- **Liste**
- **Matrice**
- **numpy modul**

Moduli i paketi

Moduli i paketi omogućavaju lakši i brži razvoj programskog koda, a u konačnici i programa. Napisano je puno modula tj. paketa za Python koji ubrzavaju razvoj programskog koda jednostavnim pozivanjem unaprijed napisanih funkcija, razreda, konstanti. Moduli tj. paketi su organizirani u smislene cjeline radi što lakšeg snalaženja, ne samo prilikom korištenja već i radi što lakšeg traženja po dokumentaciji. Module i pakete koji dolaze sa standardnom instalacijom Pythona nazivamo standardnom bibliotekom. Ponekad su programerima potrebne funkcionalnosti koje ne dolaze s modulima tj. paketima iz standardne biblioteke. Za takve funkcionalnosti potrebno je poslužiti se Internetom i potražiti odgovarajuće pakete te ih instalirati na računaru na kojem će se program izvršavati.

Potrebno je razlikovati module i pakete. Moduli su elementi programskog koda unutar kojih je implementirana neka specifična funkcionalnost. Uzmimo za primjer nekoliko modula koji dolaze sa standardnom bibliotekom: math, cmath, random, datetime. Paketi su cjeline koje uključuju druge module i oni omogućavaju njihovo hijerarhijsko grupiranje.

Funkcije, konstante i razrede koji se nalaze unutar nekog modula, moguće je koristiti na dva načina:

- najavljivanjem korištenja i
- uključivanjem u program.

Najavljivanje korištenja modula ostvaruje se pomoću ključne riječi import. U Sljedećem kodu bit će pokazan način upotrebe naredbe import.

```
1 import math #importovanje math modula
2
3
4 print(math.sin(55)) #koristenje funkcije sinus iz math modula
5 print(math.tan(55)) #koristenje funkcije tangens iz math modula
```

U gornjem primjeru prikazan je način korištenja naredbe import. Kod ovakvog načina korištenja naredbe import najavljuje se korištenje svih funkcija, razreda i konstanti iz nekog modula. Ako se

korištenje funkcija samo najavi kao u gornjem primjeru, tada u svakom dijelu programa u kojem želimo iskoristiti funkciju iz najavljenog modula mora pisati ime modula i ime funkcije odvojene tačkom. Sintaksa ovakvoga načina korištenja najavljenih funkcija jest:

modul.funkcija()

Ako se želi izbjeći gornja sintaksa i koristiti samo ime funkcije kao poziv, a ne da se ispred imena funkcije piše ime modula u kojem se funkcija nalazi, tada je potrebno željene funkcije uključiti u program. Uključivanje funkcija u program radi se korištenjem ključnih riječi from i import. Sintaksa takvoga načina korištenja funkcija jest:

$$from < modul > import < funkcija >, < funkcija >, ...$$

Ključna riječ from govori koji modul će se koristiti, dok ključna riječ import govori koje sve funkcije iz modula će se uključiti u naš program. Kod ovakvog načina uključivanja funkcija u program, prilikom korištenja uključenih funkcija, za razliku od prijašnjeg primjera, više nije potrebno pisati u kojem modulu se nalazi korištena funkcija.

```

from math import sin,tan

print(sin(55))
print(tan(55))
```

U modulima se mogu nalaziti i razne konstante. Tako se u modulu math nalazi nekoliko matematičkih konstanti: pi, e, tau, inf, nan. U nastavku sklijedi Python kod u kojem je prikazana upotreba nekih funkcija moje se nalaze unutar math modula.

```
# -*- coding: utf-8 -*-
import math;
from math import sqrt, log10, log2,exp

alfa=90;

print(math.cos(alfa)) #cosinus ugla
print(math.sin(alfa)) #sinus ugla
print(math.tan(alfa)) #tangens ugla

pi=math.pi
pi_stepeni= math.degrees(pi) #konvertuje ugao iz radijana u stepene
print(pi_stepeni)

alfa_radijani = math.radians(alfa) # konvertuje ugao iz stepena u radijane
print(alfa_radijani)


print(math.factorial(5)) #faktorijel cijelog broja

print(sqrt(64)); #drugi korijen
print(log10(1000)); #logaritam po bazi 10
print(log2(256)); #logaritam po bazi 2

print(exp(2)); #e^2


# ================================================================================
# Konstante
# ================================================================================
print(math.pi)
print(math.e)
```

```
print(math.nan)
print(math.inf)


# ============================================================================
# Zaokru ivanje
# ============================================================================

from math import ceil, floor
a=1.3

print(ceil(a)) #naredni ve i cijeli broj
print(floor(a)) # prvi manji cijeli broj
print(round(a)) #bli i(po apsolutnoj vrijednosti) cijeli broj
```

Ukoliko postoji pottreba za radom sa kompleksnim brojevima korisit se cmath modul. U nastavku sklijedi Python kod u kojem je prikazana upotreba nekih funkcija moje se nalaze unutar cmath modula.

```
# -*- coding: utf-8 -*-
# ============================================================================
# CMATH modul
# ============================================================================

import cmath

a= 3-2j
z = complex(2, -3)

print(a.real) #realni dio kompleksnog broja
print(a.imag) #imaginarni dio kompleksnog broja

print(cmath.phase(a)) #faza kompleksnog broja


print(z.real) #realni dio kompleksnog broja
print(z.imag) #imaginarni dio kompleksnog broja
print(abs(z)) #realni dio kompleksnog broja
```

Liste

Liste možemo smatrati skupom (kolekcijom) varijabli koje su pohranjene u jednom objektu. Kao što je već prije opisano, u varijablu možemo spremiti samo jednu vrijednost, no to ponekad nije dovoljno. Uzmimo za primjer da imamo petlju koja učitava brojeve tako dugo dok se s tastature ne unese broj 0. U tom slučaju prilikom pisanja programa nije nam unaprijed poznato koliko će brojeva korisnik unijeti te zbog toga nismo u mogućnosti unesene brojeve pohranjivati u varijable (jer ne znamo koliko varijabli različitih imena moramo kreirati). Dodatno, otežavajuća okolnost je to što ako koristimo klasične varijable, tada je potrebno kontrolirati u koju od varijabli će se učitani broj upisati s obzirom na to koji se broj po redu trenutačno učitava. To je potrebno raditi kako upisana vrijednost ne bi pregazila vrijednost koju smo spremili u neku varijablu u prethodnom koraku petlje. Prednost liste je u tome da se kreira jedan objekt, tj. lista te se u tu listu jednostavno unose vrijednosti redoslijedom koji god da nam je potreban. Listu možemo zamisliti kao tabelu s jednim redom u kojem se nalaze kolone. U kolone te tabele stavljamo elemente jedan iza drugoga. Ispod se nalazi grafički prikaz kako možemo zamisliti da izgleda lista u koju stavljamo elemente. Svaka od donjih ćelija može sadržavati neku vrijednost.

Prilikom kreiranja liste, potrebno je posvetiti pažnju tomu što će biti spremljeno u svaku pojedinu listu prema nekoj logici, tj. potrebno je obratiti pažnju na organizaciju podataka.

Svaka Unesena vrijednost u listu dobiva svoj indeks te se dodavanjem svakoga novog elementa ta vrijednost povećava za 1. Indeksi uvijek kreću uvijek od 0, a ne od 1 kako bi se moglo očekivati.

Za razliku od nekih drugih programskih jezika, na primjer programskog jezika C ili C++, kod Pythona nije potrebno definisati koji tip vrijednosti će biti pohranjen u elemente liste. Sintaksa definiranja liste je:

$$ime_liste = [element1, element2, ...]$$

Ako želimo kreirati praznu listu, to radimo na sljedeći način:

$$ime_liste = []$$

U nastavku se nalazi primjer kreiranja i ispisa elemenata liste.

```
# -*- coding: utf-8 -*-

lista=["Amera", 26, 1.76]
print(lista)
print(type(lista))
```

Iz gore navedenoga primjera vidimo da se u prvoj liniji kreira lista od 3 elementa. Kreirana lista sastoji se od elemenata koji su različitoga tipa podataka (niz znakova, cijeli broj i decimalni broj). Elementi liste stavljaju se unutar uglatih zagrada. Ako se želi kreirati prazna lista, tada se može napisati samo lista = []. Pozivom funkcije print(), kojoj se kao jedini argument šalje prethodno kreirana lista, ispisuju se svi elementi liste u jednostavnom i preglednom obliku.

Ako u listi imamo pohranjene vrijednosti, pretpostavka je da im želimo i pristupiti, jer čemu imati listu ako po listi nije moguće iterirati i dohvaćati tačno određene elemente. U prethodnom dijelu prikazano je kreiranje liste i ispisivanje svih elemenata liste pomoću funkcije print(). U većini slučajeva samo ispis svih elemenata liste logici programa i ne donosi dodanu vrijednost. Zato pomoću indeksa elemenata liste možemo dohvatit element po element. U primjeru ispod je pokazano nekoliko načina dohavatanja elemenata iz liste pomoću indeksiranja. U nastavku se nalazi primjer kreiranja i ispisa elemenata liste.

```
lista=["Nula", "Jedan", "Dva", "Tri", "Cetiri", "Pet"]

print(lista) #svi elemeneti liste
print(lista[0]) #prvi element liste
print(lista[1:3]) # drugi i treci element liste
print(lista[:2]) #prvi i drugi element liste
print(lista[2:]) #elementi liste od treceg do kraja liste
print(lista[3:5]) #cetvrti i peti lement liste

# upisati dir(lista) u konzoli
```

Osim često korištenih akcija nad listom, ponekad su potrebne i akcije koje programer ne koristi često i jednostavno ih smetne s uma. Ako se na jednostavan način želi dobiti popis svih akcija koje je moguće napraviti nad listom, to je moguće dobiti tako da se pozove funkcija dir() koja kao argument prima listu. Funkcija dir() ispisuje akcije koje je moguće napraviti nad predanim objektom.

Ako se funkcija dir() pozove unutar samog programskog koda (skripte). Na zaslon nam se neće ispisati ništa. Funkcija dir() mora se pozvati unutar naredbenoga retka.

Nakon poziva funkcije dir() unutar konzole, vidimo da su metodi koje možemo koristiti na listama:

- append() - dodaje novi element na kraj liste
- clear() - uklanja sve elemente iz liste

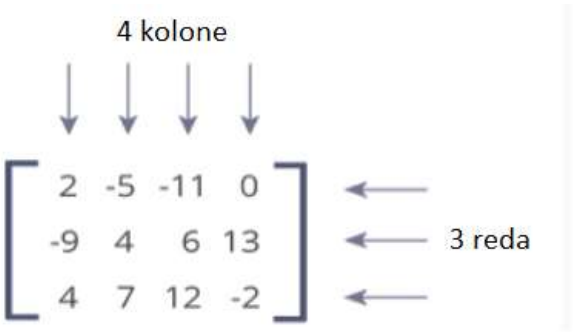
- copy() - kopira sve elemente liste u drugu listu
- count() - vraća koliko ima elemenata u listi neke određenevrijednosti
- extend() - proširuje postojeću listu s elementima neke druge liste
- index() - pronalazi i vraća indeks traženog elementa (ako ima više vrijednosti, vraća indeks najmanje vrijednosti)
- insert()- stavlja vrijednost u listu na tačno određenu poziciju
- pop() - iz liste uklanja vrijednost koja se nalazi na kraju liste
- remove() - iz liste uklanja element određene vrijednosti
- reverse()- ova metoda mijenja pozicije elemenata u suprotni raspored (prvi element postaje zadnji, drugi element postaje predzadnji itd.)
- sort()- metoda koja sortira elemente liste.

Najjednostavniji način koji omogućava da se prođe kroz sve elemente liste (ili barem dio njih) je korištenjem petlje for. Petlji predamo cijelu listu te ona dohvata element po element iz liste.

```
1
2  lista=[1,2,3,4,5,5, 5, 6,7,8]
3
4  #itetiranje kroz elemente liste
5
6  for x in lista:
7      print(x + 5)
8
9  #metodi za liste
10
11 print(lista)
12 lista_kopija=lista.copy() #kreirane nove liste koja je kopija liste
13 print(lista_kopija)
14 a=lista.count(5) #broj elemenata 5 u listi
15 print(a)
16
17 lista2= ['a','d', 'b','c'] #nova lista
18
19 lista.extend(lista2) #dodaj na kraj liste lemenete liste2
20 print(lista)
21
22 indeks=lista.index(4) #pronalazi indeks elementa 4 u listi
23 print(indeks)
24 indeks1=lista.index(5)
25 print(indeks1)
26
27 lista.insert(2,'test') #dodaje test na mjesto u listi sa indeksom 2
28 print(lista)
29
30 lista.pop(2) #izbacuje iz liste element sa indeksom 2
31 print(lista)
32
33 lista.remove(5) #brise iz liste element koji ima vrijednost 5
34 print(lista)
35
36 lista.reverse() #obrnuti redoslijed elemeenata liste
37 print(lista)
38
39 lista2.sort()
40 print(lista2)
41
42 lista.clear() #brisanje svih elemenata liste
43 print(lista)
```

Matrice

Matrica je dvodimenzionalna struktura podataka gdje se brojevi raspoređuju u redove i kolone, kao što je prikazano na slici. Matrice su vrlo važne strukture podataka za mnoge matematičke proračune.



Bitno je napomenuti da u programskom jeziku Python za manipulaciju nad matricama koristimo **NumPy** paket o kojem ćemo nešto više u ovom poglavlju. Indeksi N-dimenzionalne matrice/niza u Pythonu počinju od vrijednosti 0.

U prvom primjeru prikazat ćemo osnove sa matricama:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

A = [[1, 4, 5],  #matrica 2x3 kreirana kao "lista u listi"
    [-5, 8, 9]]

print(A)

A = [[1, 4, 5, 12],   #matrica 3x4
    [-5, 8, 9, 0],
    [-6, 7, 11, 19]]

print("A =", A)
print("A[1] =", A[1])       # drugi red
print("A[1][2] =", A[1][2])   # tre i element drugog reda
print("A[0][-1] =", A[0][-1])   # zadnji element prvog reda

column = [];         # prazna lista/kolona
```

NumPy

Za razliku od Matlab-a, koji ima ugrađenih osnovnih funkcija, u Python-u je to podijeljeno po paketima. Jedan od njih koji je u ovom poglavlju bitan, za lakše korištenje matrica, je paket NumPy. NumPy se koristi za osnovno numeričko računanje. Više možete naći na stranici NumPy. Za kreiranje matrica u Pythonu najčešće koristimo nizove(array()), ali Python može koristiti i naredbu matrix().

Prednost korištenja Anaconda/Spyder, je u tome što potrebne pakete ne moramo intalirati dodatno, nego je bitno samo uključiti ih prije korištenja neke od nama potrebnih naredbi koje se nalaze u određenom paketu.

Neke od osnovnih funkcija NumPy paketa, koje ćemo koristiti na matricama, su:

- matrix()
- zeros()

- ones()
- eye()
- array(): kreira niz
- dot(): proizvod matrica različitih dimenzija
- add(): sabiranje matrica
- substract():oduzimanje matrica
- divide(): dijeljenje matrica
- multiply(): množenje matrica istih dimenzija,...

Krenuti ćemo sa osnovnim funkcijama nad matricama, kao što je prikazano u primjeru ispod:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


import numpy as np #koristi paket numpy sa skra enim zapisom np

A = np.array([1, 2, 3]) #kreiranje niza
B = np.matrix([[1, 2, 3], [3, 4, 5]]) #kreiranje matrice funkcijom matrix()
print(A)
print(B)

A = np.array([[1, 2, 3], [3, 4, 5]]) #kreiranje niza kao matrice 2x3
print(A)

A = np.array([[1.1, 2, 3], [3, 4, 5]]) #kreiranje matrice, sa float tipom podataka
print(A)

A = np.array([[1, 2, 3], [3, 4, 5]], dtype = complex) # kreiranje matrice gdje  e  svi
                                                      # elementi biti kompleksni
    brojevi
print(A)

print("A=",np.zeros((3, 2),dtype=int)) #kreiranje matrice nula sa n kolona i m redova

print("A=",np.ones((4,3))) #kreiranje matrice jedinica sa n kolona i m redova

print("A=",np.eye(3, dtype=int)) #kreiranje jedini ne matrice tipa int

B=np.eye(3, dtype=int)
print(B[1][1]) #element matrice
```

Operacije nad matricama

Sabiranje matrica

```
# -*- coding: utf-8 -*-

import numpy as np

A = np.array([[2, 4], [5, -6]]) #matrica A 2x2
B = np.array([[9, -3], [3, 6]]) #matrica B 2x2
C = A + B      # sabiranje dvije matrice
print(C)
C = np.add(A,B)
print(C)
```

Množenje matrica

NumPy nizovi(array()) podržavaju množenje po elementima, a ne množenje redova. Za takvu vrstu množenja se koriste NumPy matrix().

```
# -*- coding: utf-8 -*-
import numpy as np

A = np.matrix([[3, 6, 7], [5, -3, 0]])
B = np.array([[1, 1], [2, 1], [3, -3]])
C = A.dot(B) #kori tenje funkcije dot() za mno enje matrica razli itih dimenzija
print(C)

A = np.array([[1, 2], [4, 5]])
B = np.array([[7, 8], [9, 10]])
C = np.multiply(A,B) #mno enje matrica istih dimenzija
print(C)
```

Oduzimanje matrica

```
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 11 12:55:21 2020

@author: Minja
"""

import numpy as np

A = np.array([[2, -4, 5], [-6, 2, 0]])
B = np.array([[0, -7, 5], [5, -2, 9]])

print ("Matrica A: ", A)
print ("Matrica B : ", B)


C = np.subtract(A, B)
print ("Rezultat: ", C)
```

Dijeljenje matrica

```
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 11 13:05:41 2020

@author: Minja
"""
import numpy #bez kori tenja skra enice
A = numpy.array([[1, 2], [4, 5]])
B = numpy.array([[7, 8], [9, 10]])
C = numpy.divide(A,B)
print(C)

# ta  bi bilo da imamo razli ite dimenzije matrica?

#dijeljenje konstantom
C = B/2
print(C)
```

Transponovanje matrice

```
# -*- coding: utf-8 -*-
import numpy as np

A = np.array([[1, 1], [2, 1], [3, -3]])
print(A.transpose()) #transponovana matrica
```

Pristupanje elementima matrice, kolonama i redovima

Elementima matrice se pristupa slično kao i listama, što je pokazano na početku, samo ovaj put korištenjem NumPy paketa koji nam to olakšava.

Pristupanje elementima niza

```
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 11 12:40:13 2020

@author: Minja
"""

import numpy as np
A = np.array([2, 4, 6, 8, 10])

print("A[0] =", A[0])      # Prvi element
print("A[2] =", A[2])      # Tre i element
print("A[-1] =", A[-1])    # Posljedni element
```

Pristupanje redovima

```
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 11 12:40:14 2020

@author: Minja
"""
#-------------------------------------------------------------------------
#                         Pristupanje elementima reda
#-------------------------------------------------------------------------
import numpy as np
A = np.array([[1, 4, 5, 12], #matrica 3x4
    [-5, 8, 9, 0],
    [-6, 7, 11, 19]])

#  Prvi element prvog reda
print("A[0][0] =", A[0][0])

# Tre i element drugog reda
print("A[1][2] =", A[1][2])

# Posljednji element posljednjeg reda
print("A[-1][-1] =", A[-1][-1])

#-------------------------------------------------------------------------
#                         Pristupanje redovima
#-------------------------------------------------------------------------
A = np.array([[1, 4, 5, 12],
    [-5, 8, 9, 0],
    [-6, 7, 11, 19]])

print("A[0] =", A[0]) # Prvi red
print("A[2] =", A[2]) # Tre i red
```

```
print("A[-1] =", A[-1]) # Posljedni red (3. u na em slu aju)
                        # Sa -1 kre e unazad
```

Pristupanje kolonama

```
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 11 12:40:14 2020

@author: Minja
"""

#-------------------------------------------------------------------------
#                         Pristupanje kolonama
#-------------------------------------------------------------------------
import numpy as np

A = np.array([[1, 4, 5, 12],
    [-5, 8, 9, 0],
    [-6, 7, 11, 19]])

print("A[:,0] =",A[:,0]) # Prva kolona
print("A[:,3] =", A[:,3]) #   etvrta   kolona
print("A[:,-1] =", A[:,-1]) # Posljednja kolona
```

U nastavku ćemo uraditi par osnovnih primjera korištenja matrica, bez upotrebe numpy modula. Numpy moduli se koriste za kompleksnije operacije nad matricama, koje nisu tema ovog kursa.

Primjer 1

```
# -*- coding: utf-8 -*-
"""
Napraviti unos redova i kolona matrice preko tastature.
"""

a=input("Broj redova:")
a=eval(a)
b=input("Broj kolona:")
b=eval(b)
c=[]

for i in range(a): #odre ivanje redova
        c.append([]) #append dodaje elemente na kraj liste
        for j in range(b): #odre ivanje kolona
            c[i].append(i+j)

print(c)

"""
U for petljama na mjestu range() su sada upisane
varijable reda i kolone odnosno u na em slu aju a i b.
Koliko god stupaca i redaka unijeli, python  e
automatski popuniti pozicije elemenata u rangu
od nula do vrijednosti koju smo preko tastature odredili
kao krajnju vrijednost redova i kolona

"""
```

Primjer 2

```
# -*- coding: utf-8 -*-

"""
Kreirati dvije matrice s fiksnim elementima i sabrati ih.
"""

a=[]

for i in range(3): #odre ivanje redova
        a.append([])
        for j in range(3): #odre ivanje kolona
            a[i].append((i+j))
print("1.matrica")
print(a)

b=[]

for i in range(3): #odre ivanje redova
        b.append([])
        for j in range(3): #odre ivanje kolona
            b[i].append((i+j))
print("2.matrica")
print(b)

c=a+b
print(c)
```

Primjer 3

```
# -*- coding: utf-8 -*-
"""
Mno enje dvije matrice kori tenjem ugnje dene petlje
"""
#3x3 matrica
X = [[12,7,3],
    [4 ,5,6],
    [7 ,8,9]]
#3x4 matrica
Y = [[5,8,1,2],
    [6,7,3,0],
    [4,5,9,1]]
#Rezultat  e  biti 3x4
rezultat = [[0,0,0,0],
         [0,0,0,0],
         [0,0,0,0]]

# iteracije kroz redove x
for i in range(len(X)):
   # iteracije kroz kolone y
   for j in range(len(Y[0])):
       # iteracije kroz redove y
       for k in range(len(Y)):
           rezultat[i][j] += X[i][k] * Y[k][j]

for r in rezultat:
   print(r)
```