Praktikum iz programiranja 3



2024/25

NumPy

- NumPy predstavlja jednu od mnogobrojnih biblioteka u programskom jeziku Pajton.
- NumPy, skraceno od "Numerical Python" se sastoji od mnoštva numeričkih funkcija, koje služe za obradu višedimenzionih nizovnih objekata.
- Prvi put se matrice pojavljuju u okviru Pajtona 1994. godine, kada ih je uveo Jim Fulton stvorivši biblioteku „Matrix Object in Python“.
- Jim Hugunin 1995. godine proširio, stvorivši biblioteku pod nazivom „Numeric“.
- Posle šest godina Perry Greenfield, Rick White i Todd Miller su stvorili biblioteku „Numarray“. Sa bibliotekom NumPy se prvi put susrećemo 2005. godine, kada je stvorio Travis Oliphant.

NumPy

- Koristeći NumPy, programer može da izvede sledeće operacije:
  - Matematičke i logičke operacije nad nizovima
  - Furierove transformacije i rutine za manipulaciju oblicima
  - Operacije povezane sa linearnom algebrom, poput operacija nad vektorima, matricama, resavanje sistema jednacina i tome slično
- NumPy ima ugrađene funkcije za linearnu algebru i nasumičnu generaciju brojeva.
- NumPy se često koristi zajedno sa paketima kao sto su SciPy ("Scientific Python") i Mat-plotlib (biblioteka za grafike), u kombinaciji sa kojima nam omogućava moderniju i kompletniju zamenu za MatLab.

NumPy okruženje

- Standardna Pajton instalacija ne dolazi u paketu sa NumPy modulom.
- Laka alternativa je da instalirate NumPy koristeći popularni program za instalaciju Pajton paketa, pip.

```
pip install numpy
```

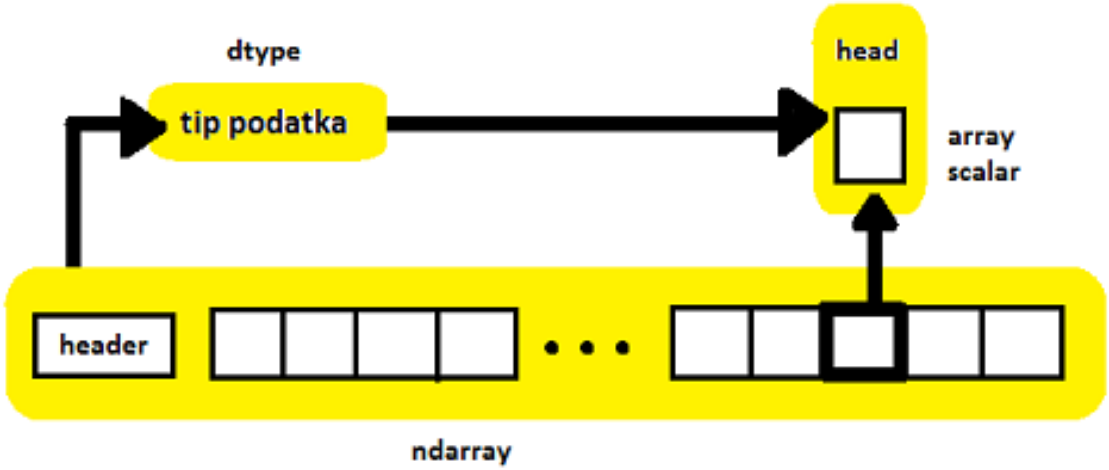
- Preuzeti Pajton instalaciju sa python.org , prilikom instalacije odabrati da ista uključi pip paket menadžer
- Nakon instalacije u komandnoj liniji (cmd) izvršiti „pip install numpy“ da bi dodatno instalirali NumPy.

Tipovi podataka u NumPy-u

- U NumPy paketu, najvažniji objekat, `ndarray`, predstavlja n-dimezionalni niz elemenata gde svaki element zauzima neki fiksirani broj bajtova. Obično taj broj bajtova reprezentuje neki broj, celi, kompleksni, realni itd.
- Taj broj bajtova može da predstavlja zapis sačinjen od bilo kog drugog tipa podatka (karakter, logički tip – boolean, string itd.).
- Svaki NumPy niz ima svoj `dtype` objekat koji opisuje tip podataka u tom nizu.
- `Ndarray` predstavlja kolekciju podataka istoga tipa. Pristupa im se koristeći indekse počevši od 0 do n-1 (gde je n ukupan broj podataka). Ndarray objekat se koristi i za matrice i za vektore.

Tipovi podataka u NumPy-u

- Veza između tri osnovna objekta korišćena da opišu podatak u nizu:
  - **ndarray**
  - **dtype** - tip podataka objekta koji opisuje plan jednog elementa niza fiksirane veličine
  - **array scalar** – objekat (niz) koji se vraća kada se pristupi nekom elementu niza



NumPy
ndarray

```
import numpy as np

a = np.array([1, 2, 3])
print(a)
print(type(a))
print(a.shape)  #oblik niza
print(a[0], a[1], a[2]) #pristupanje elementima
a[0] = 5 #promena vrednosti
print(a)
```

```
[1 2 3]
<class 'numpy.ndarray'>
(3,)
1 2 3
[5 2 3]
```

NumPy
ndarray

```
# tip niza može eksplicitno da se definiše pri kreiranju niza.
import numpy as np

# kreiranje niza od liste sa realnim tipom:
a = np.array([[1, 2, 3], [4, 5, 6]], dtype = 'float')
print ("Niz kreiran iz nekadasnje liste:\n", a)

# kreiranje niza od torke:
b = np.array((9, 8, 7))
print ("\nNiz kreiran iz nekadasnje torke:\n", b)

# kreiranje 3X4 niza sa svim nulama:
c = np.zeros((3, 4))
print ("\nNiz inicijalizovan sa nulama:\n", c)
```

```
Niz kreiran iz nekadasnje liste:
[[1. 2. 3.]
[4. 5. 6.]]

Niz kreiran iz nekadasnje torke:
[9 8 7]

Niz inicijalizovan sa nulama:
[[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]]
```

NumPy
ndarray

```
# kreiranje konstantne vrednosti niza kompleksnog tipa:
d = np.full((3, 3), 6, dtype = 'complex')
print ("\nNiz inicijalizovan sa svim 6s. Tip niza je kompleksan:\n", d)

# kreiranje niza sa random vrednostima:
e = np.random.random((2, 2))
print ("\nRandom niz:\n", e)

# kreiranje niza celih brojeva od 0 do 30 sa korakom 5:
f = np.arange(0, 30, 5)
print ("\nNiz celih brojeva sa korakom 5:\n", f)
```

```
Niz inicijalizovan sa svim 6s. Tip niza je kompleksan:
[[6.+0.j 6.+0.j 6.+0.j]
 [6.+0.j 6.+0.j 6.+0.j]
 [6.+0.j 6.+0.j 6.+0.j]]

Random niz:
[[0.90989476 0.84011767]
 [0.72367953 0.24145321]]

Niz celih brojeva sa korakom 5:
[0 5 10 15 20 25]
```

NumPy
ndarray

```
# kreiranje niza od 10 vrednosti u rangu od 0 do 5:
g = np.linspace(0, 5, 10)
print ("\nNiz od 10 vrednosti izmedju 0 i 5:\n", g)
# Promena oblika 3X4 niza u 2X2X3 niz:
h = np.array([[1, 2, 3, 4], [5, 2, 4, 2], [1, 2, 0, 1]])
h1 = h.reshape(2, 2, 3)
print ("\nOriginalni niz:\n", h)
print ("Niz posle promenjenog oblika:\n", h1)
# spljosten niz:
i = np.array([[1, 2, 3], [4, 5, 6]])
i1 = i.flatten()
print ("\nOriginalni niz:\n", i)
print ("Spljosten niz:\n", i1)
```

```
iz od 10 vrednosti izmedju 0
I 5:
[0.         0.55555556
1.11111111 1.66666667
2.22222222 2.77777778
3.33333333 3.88888889
4.44444444 5.        ]

Originalni niz:
[[1 2 3 4]
[5 2 4 2]
[1 2 0 1]]

Niz posle promenjenog
oblika:
[[[1 2 3 ]
[4 5 2]]
[[4 2 1 ]
[2 0 1]]]

Originalni niz:
[[1 2 3]
[4 5 6]]

Spljosteni niz:
[1 2 3 4 5 6]
```

NumPy
Indeksiranje nizova

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

b = a[:2, 1:3]
print(b)
c=np.copy(a[:2,1:3])
print(c)
print(a[0, 1])
b[0, 0] = 77
print(a[0, 1])

c[0, 0] = 100
print(a[0, 1])
```

```
[[2 3]
 [6 7]]
[[2 3]
 [6 7]]
2
77
77
```

NumPy
Indeksiranje nizova

```
a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
print(a)
b = np.array([0, 2, 0, 1])

print(a[np.arange(4), b])
a[np.arange(4), b] += 10
print(a)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
[ 1  6  7 11]
[[11  2  3]
 [ 4  5 16]
 [17  8  9]
 [10 21 12]]
```

NumPy
Indeksi sa ispunjenim uslovom

```
a = np.array([[1,2], [3, 4], [5, 6]])

bool_idx = (a > 2)
print(bool_idx)

print(a[bool_idx])
```

Ili

```
print(a[a > 2])
```

```
[[False False]
 [ True  True]
 [ True  True]]
[3 4 5 6]
[3 4 5 6]
```

NumPy
Vrste i kolone

Postoje dva načina za pristupanje vrstama/kolonama

```
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
row_r1 = a[1, :]
row_r2 = a[1:2, :]
print(row_r1, row_r1.shape)
print(row_r2, row_r2.shape)

col_r1 = a[:, 1]
col_r2 = a[:, 1:2]
print(col_r1, col_r1.shape)
print(col_r2, col_r2.shape)
```

```
[5 6 7 8] (4,)
[[5 6 7 8]] (1, 4)

[ 2  6 10] (3,)
[[ 2]
 [ 6]
 [10]] (3, 1)
```

NumPy
Tipovi elemenata

```
x = np.array([1, 2])
print(x.dtype)
```

```
int64
```

```
x = np.array([1.0, 2.0])
print(x.dtype)
```

```
float64
```

```
x = np.array([1, 2], dtype=np.float64)
print(x.dtype)
print(x)
```

```
float64
[1. 2.]
```

NumPy
Operacije nad nizovima

```
import numpy as np
a = np.array([[0,2],[3,8],[1,6]])
b = np.array([[1,0],[0,1],[1,1]])
print("a<b\n",a<b)
print("a+1\n",a+1)
print("a+b\n",a-b)
print("a*10\n",a*10)
print("a**2\n",a**2)
```

```
a<b
 [[ True False]
 [False False]
 [False False]]
a+1
 [[1 3]
 [4 9]
 [2 7]]
a+b
 [[-1  2]
 [ 3  7]
 [ 0  5]]
a*10
 [[ 0 20]
 [30 80]
 [10 60]]
a**2
 [[ 0  4]
 [ 9 64]
 [ 1 36]]
```

NumPy
Operacije nad nizovima

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

print(x + y)          ili     print(np.add(x, y))
[[ 6.0  8.0]
 [10.0 12.0]]

print(x - y)          ili     print(np.subtract(x, y))
[[-4.0 -4.0]
 [-4.0 -4.0]]
```

NumPy
Vektorski proizvod

```
x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])
v = np.array([9,10])
w = np.array([11, 1])
```

```
print(v.dot(w))
```
ili
```
print(np.dot(v, w))
```

```
109
```

```
print(x.dot(v))
```
ili
```
print(np.dot(x, v))
```

```
[29 67]
```

```
print(x.dot(y))
```
ili
```
print(np.dot(x, y))
```

```
[[19 22]
 [43 50]]
```

NumPy
Sumiranje, transponovanja

```
x = np.array([[1,2],[3,4]])

print(np.sum(x))
print(np.sum(x, axis=0))
print(np.sum(x, axis=1))

print(x)
print(x.T)
v = np.array([1,2,3])
print(v)     print(v.T)
```

```
10
[4 6]
[3 7]

[[1 2]
 [3 4]]

[[1 3]
 [2 4]]

[1 2 3]
```

Primer 1

- Napisati funkciju **SahovskaTabla(n)** koja vraća matricu dimenzija nxn u kojoj se jedinice nalaze na mestima gde stoje crna polja, a nule na mestima gde su bela polja.

```
def SahovskaTabla(n):
    v0=np.array([0,1]*(n//2+1))
    v1=np.array([1,0]*(n//2+1))
    b=np.array([v0,v1]*(n//2+1))
    c=b[0:n,0:n]
    print(c)

SahovskaTabla(8)
SahovskaTabla(5)
```

Primer 2

- U matematici skalarni proizvod predstavlja algebarsku operaciju koja uzima koordinate dva vektora iste veličine i vraća jedan broj. Definicija skalarnog proizvoda:

$$\vec{x} \cdot \vec{y} = |\vec{x}||\vec{y}| \sphericalangle(\vec{x}, \vec{y})$$

- odakle je

$$\sphericalangle(\vec{x}, \vec{y}) = arccos \frac{\vec{x} \cdot \vec{y}}{|\vec{x}||\vec{y}|}$$

- Postupak računanja ugla između dv vektora $\vec{x} = (5{,}2, -3)$ i $\vec{y} = (-1{,}0{,}9)$:

Primer 2

```
import numpy as np
x = np.array([5, 2, -3])
y = np.array([-1, 0, 9])
# na osnovu ova dva vektora se racuna njihov skalarni proizvod:
skal = np.dot(x, y)
# računanje modula vektora x i vektora y:
mod_x = np.sqrt((x*x).sum())
mod_y = np.sqrt((y*y).sum())
# racunanje cosinusa ugla između x i y vektora:
cos_xy = skal/mod_x/mod_y
# racunanje ugla između vektora x i y:
ugao = np.arccos(cos_xy)
print("Ugao izmedju vektora x i y u radijanima: ", ugao)
# prebacivanje ugla iz radijana u stepene:
ugao_stepeni = ugao*180/np.pi
print("Ugao izmedju vektora x i y u stepenima: ", ugao_stepeni)
```

NumPy
Rešavanje sistema jednačina

- Jedan od najčešćih problema u linearnoj algebra je rešavanje sistema jednačina. Svaki sistem jednačina može da se zapiše u matričnom obliku.
- Potrebno je da se odredi vector x, koji ispunjava jednačinu: Ax = b, gde je

$$A = \begin{bmatrix} 2 & 1 & -2 \\ 3 & 0 & 1 \\ 1 & 1 & -1 \end{bmatrix}, b = \begin{bmatrix} -3 \\ 5 \\ -2 \end{bmatrix}$$

NumPy
Rešavanje sistema jednačina

```
import numpy as np
#prvo formiramo matrice (nizove) A I b:
A = np.array([[2,1,-2],[3,0,1],[1,1,-1]])
b = np.array([-3,5,-2])
#sistem resavamo na sledeci nacin:
x = np.linalg.solve(A,b.T)
print(x)
#provera da li je resenje ispravno:
print(np.allclose(np.dot(A,x),b))
```

```
[1. -1. 2.]
True
```