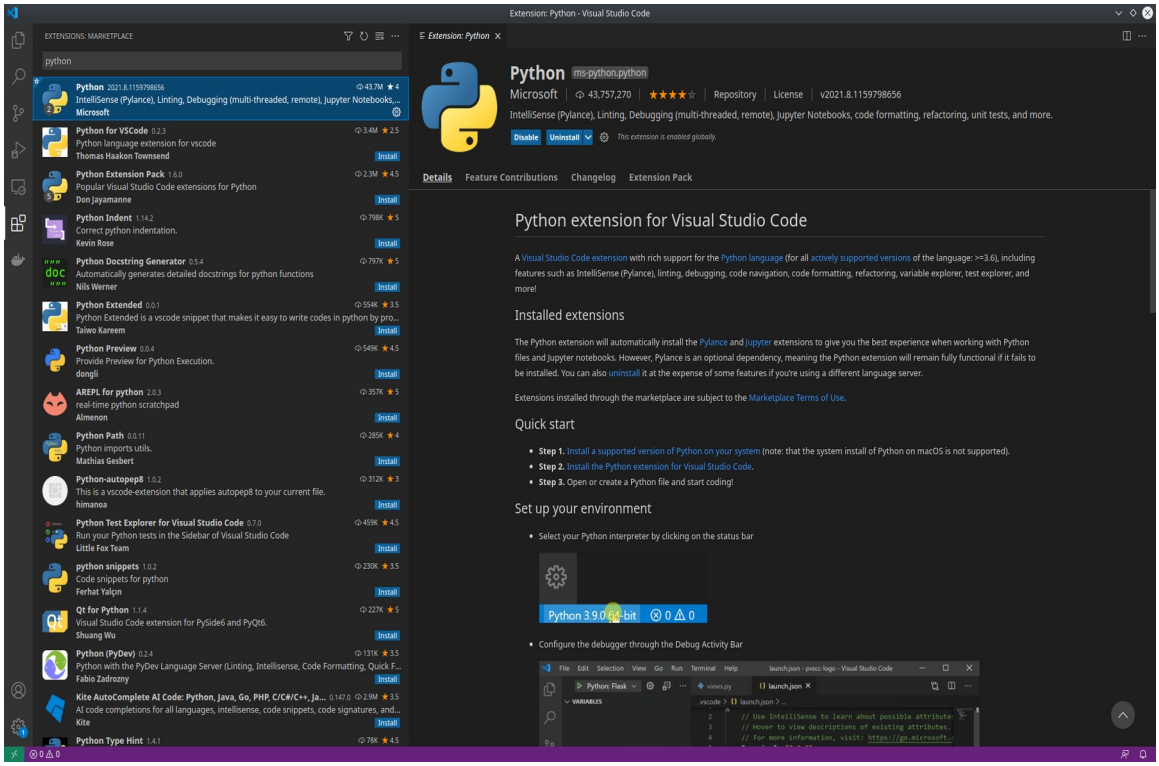
Python uvod

Python je interpreterski jezik, čija sintaksa liči na pseudo jezik. Python omogućava da se u nekoliko linija na čitljiv način predstave moćne ideje. Trenutno postoje dve različite verzije Python programskog jezika 2.7 i 3.x. , gde je podrška za 2.7 ukinuta 2020. godine. Na ovom kursu koristiće se Python 3.x.
Python možete preuzeti sa linka.
Provera Python verzije se vrši pomoću komande `python –version` u terminalu.
U ovom tutorijalu biće pokrivene:

- Osnove Python: Osnovni tipovi podataka, Liste, Rečnici, Funkcije, Klase
- Numpy biblioteka: Arrays, Array indexing, Datatypes, Array math
- Matplotlib biblioteka: Plotting, Subplots

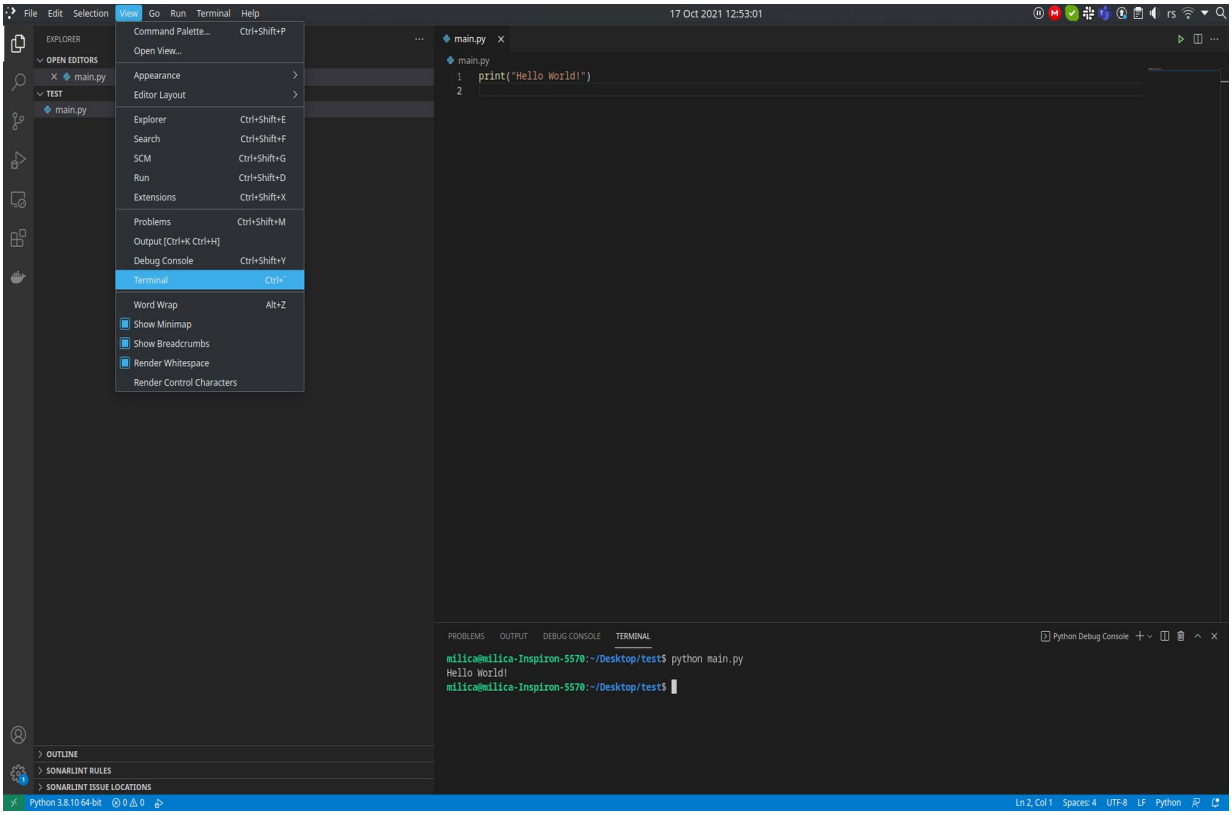
Alati

Okruženje koje će biti korišteno na vežbama je Visual Studio Code (VS Code). VS Code predstavlja *open source* editor koda. Desktop aplikacija je dostupna za Windows, macOS i Linux i može se preuzeti sa zvaničnog sajta. Dolazi sa ugrađenom podrškom za JavaScript, TypeScript i Node.js i ima široku podršku proširenja za druge jezike (kao što su C++, C#, Java, Pyhon, PHP, Go). Na slici 1 je prikazan dio za pregled i preuzimanje dodatnih ekstenzija u VS Code okruženju.



Slika 1 - Visual Studio Code Extensions

Na slici 2 je prikazan primer pokretanja Python skripte u VS Code. Za otvaranje terminala u VS Code iz menija *View* izabrati *Terminal*. Za pokretanje skipte u terminalu, pozicionirati se u folder gdje se nalazi fajl, a zatim pokrenuti naredbu `python naziv_fajla.py`.



Slika 2 – Primer pokretanja Python skripte

Alternativno, možete koristiti *PyCharm* koji predstavlja integrisano razvojno okruženje za programiranje u Python-u. Dostupan je za Windows, macOS i Linux operativne sisteme. *PyCharm* alat je dostupan na zvaničnom sajtu.

Kontrola Toka

Sintaksa:

```
if <condition>:
    <code block>
elif <condition>:
    <code block>
elif <condition>:
    <code block>
else:
    <code block>
```

Gde je:

- `<condition>`: uslov koji može biti true ili false.
- `<code block>`: sekvenca instrukcija.
- `elif` I `else` su opcioni i nekoliko `elif`-ova za jedan `if` može biti korišćeno, ali samo jedan `else` mora biti na kraju.

```
temp = 23 # upisati trenutnu temperaturu

if temp < 0:
    print ('Brrrr...')
elif 0 <= temp <= 20:
    print ('Hladno')
elif 21 <= temp <= 25:
    print ('Prijatno')
elif 26 <= temp <= 35:
    print ('Vruce')
else:
    print ('Veoma vruce!')
```

`Izlaz: Prijatno`

Petlje

FOR

Syntax:

```
for <reference> in <sequence>:
    <code block>
    continue
    break
else:
    <code block>
```

```
# Sum 0 to 99
s = 0
for x in range(1, 100):
    s = s + x
print (s)
```

`Izlaz: 4950`

Zadatak: *Ispisati zbir prvih 1000 parnih brojeva*

While

Syntax:

```
while <condition>:
    <code block>
    continue
```

```
        break
    else:
        <code block>
```

```
# Sum 0 to 99
s = 0
x = 1

while x < 100:
    s = s + x
    x = x + 1
print (s)
```

```
4950
```

Zadatak: *Generisati dva slucajna cela broja od 0-100 i ispisati ih ako im je razlika manja od 5. Nakon 50 pokusaja ispisati poruku neuspeha.*

Tipovi podataka

Brojevi

Integer i float su predstavljeni kao i u drugim programskim jezicima.

- Integer (*int*): i = 1
- Floating Point real (*float*): f = 3.14
- Complex (*complex*): c = 3 + 4j

Primeri:

```
x = 3
print (x, type(x))
```

```
3 <class 'int'>
```

```
print (x + 1)   # Addition;
print (x - 1)   # Subtraction;
print (x * 2)   # Multiplication;
print (x ** 2)  # Exponentiation;
```

```
4
2
6
9
```

```
x += 1
```

```
print (x)  # Prints "4"
x *= 2
print (x)  # Prints "8"
```

```
4
8
```

```
y = 2.5
print (type(y)) # Prints "<type 'float'>"
print (y, y + 1, y * 2, y ** 2) # Prints "2.5 3.5 5.0 6.25"
```

```
<class 'float'>
2.5 3.5 5.0 6.25
```

```
# Converting real to integer
print ('int(3.14) =', int(3.14))

# Converting integer to real
print ('float(5) =', float(5))

# Calculation between integer and real results in real
print ('5.0 / 2 + 3 = ', 5.0 / 2 + 3)

# Integers in other base
print ("int('20', 8) =", int('20', 8)) # base 8
print ("int('20', 16) =", int('20', 16)) # base 16

# Operations with complex numbers
c = 3 + 4j
print ('c =', c)
print ('Real Part:', c.real)
print ('Imaginary Part:', c.imag)
print ('Conjugate:', c.conjugate())
```

```
int(3.14) = 3
float(5) = 5.0
5.0 / 2 + 3 =  5.5
int('20', 8) = 16
int('20', 16) = 32
c = (3+4j)
Real Part: 3.0
Imaginary Part: 4.0
Conjugate: (3-4j)
```

Napomena:

Python ne podržava unarne operatore za inkrementiranje (x++) i dekrementiranje (x--)

Boolean tipovi

Python ima implementirane sve potrebne operatore za rad sa Boolean tipovima, ali koristi reči engleskog jezika umesto simbola (&&, | |, i sl.):

```
t, f = True, False
print (type(t)) # Prints "<type 'bool'>"
```

```
<class 'bool'>
```

```
print (t and f) # Logical AND;
print (t or f)  # Logical OR;
print (not t)   # Logical NOT;
print (t != f)  # Logical XOR;
```

```
False
True
False
True
```

Aritmetičke operacije:

- Sabiranje (+)
- Oduzimanje (-)
- Množenje (*)
- Deljenje (/): između dva integera rezultat je isti kao i kod integer deljenja.
- Integer Deljenje (//):
- Modul (%): vreća ostatak pri deljenju.
- Stepen (**):
- Pozitivan (+)
- Negativan (-)

Logičke Operacije:

- <
- >
- <=
- >=
- (==)

Bitwise Operacije:

- Left Shift (<<)

- Right Shift (>>)
- And (&)
- Or (|)
- Exclusive Or (^)
- Inversion (~)

Stringovi

```
hello = 'hello'   # String literals can use single quotes
world = "world"   # or double quotes; it does not matter.
print (hello, len(hello))
```

```
hello 5
```

```
hw = hello + ' ' + world  # String concatenation
print (hw)  # prints "hello world"
```

```
hello world
```

Stringovi su objekti, pa imaju mnoštvo metoda:

```
s = "hello"
print (s.capitalize())  # Capitalize a string; prints "Hello"
print (s.upper())       # Convert a string to uppercase; prints "HELLO"
print (s.rjust(7))      # Right-justify a string, padding with spaces;
prints "  hello"
print (s.center(7))     # Center a string, padding with spaces; prints "
hello "
print (s.replace('l', '(ell)'))  # Replace all instances of one substring
with another;
                                 # prints "he(ell)(ell)o"
print ('  world '.strip())  # Strip leading and trailing whitespace;
prints "world"
```

```
Hello
HELLO
  hello
 hello
he(ell)(ell)o
world
```

```
s = 'Helloo!!!'
# Interpolation
print ('Size of %s => %d' % (s, len(s)))

# String processed as a sequence
```

```
for ch in s: print (ch)
```

```
Size of Helloo!!! => 9
H
e
l
l
o
o
!
!
!
```

Više o stringovima se može pronaći u dokumentaciji.

Python kontejneri

Python sadrži nekoliko ugrađenih kontejenera: lists, dictionaries, sets, i tuples.

Python indexi:

- Počinju od nule.
- Broje se od nazad ako su negativni.
- Mogu biti definisani kao sekcije, `[start: end + 1: step]`. Ako se ne stavi start, biće računato od nule. Ako nije zadata vrednost end + 1, biće računata veličina objekta. Step je jedan, ako se ne zada nikakva vrednost.

```
print ('Python'[::-1])
# shows: nohtyP
```

```
nohtyP
```

```
numbers = range(0,40)
evens = numbers[2::2]

print (evens)
```

```
range(2, 40, 2)
```

```
numbers = range(0,40)
test = numbers[40:0:-2]
test
```

```
range(39, 0, -2)
```

Liste

Liste su Python ekvivalent nizovima. Mogu biti promenljive veličine i da sadrže elemente različitih tipova.

Sintaksa:

```
    list = [a, b, ..., z]
```

Operacije nad listama:

```
xs = [3, 1, 2]   # Create a list
print (xs, xs[2])
print (xs[-1])     # Negative indices count from the end of the list;
prints "2"
```

```
[3, 1, 2] 2
2
```

```
xs[2] = 'foo'    # Lists can contain elements of different types
print (xs)
```

```
[3, 1, 'foo']
```

```
xs.append('bar') # Add a new element to the end of the list
print (xs)
```

```
[3, 1, 'foo', 'bar']
```

```
x = xs.pop()     # Remove and return the last element of the list
print (x, xs)
```

```
bar [3, 1, 'foo']
```

```
# a new list: 70s Brit Progs
progs = ['Yes', 'Genesis', 'Pink Floyd', 'ELP']

# processing the entire list
for prog in progs:
    print (prog)

# Changing the last element
progs[-1] = 'King Crimson'

# Including
progs.append('Camel')

# Removing
```

```
progs.remove('Pink Floyd')

# Ordering
progs.sort()

# Inverting
progs.reverse()

# prints with number order
for i, prog in enumerate(progs):
    print (i + 1, '=>', prog)

# prints from de second item
print (progs[1:])

my_list = ['A', 'B', 'C']
print ('list:', my_list)

# The empty list is evaluated as false
while my_list:
    # In queues, the first item is the first to go out
    # pop(0) removes and returns the first item
    print ('Left', my_list.pop(0), ', remain', len(my_list))

# More items on the list
my_list += ['D', 'E', 'F']
print ('list:', my_list)

while my_list:
    # On stacks, the first item is the last to go out
    # pop() removes and retorns the last item
    print ('Left', my_list.pop(), ', remain', len(my_list))
```

```
Yes
Genesis
Pink Floyd
ELP
1 => Yes
2 => King Crimson
3 => Genesis
4 => Camel
['King Crimson', 'Genesis', 'Camel']
list: ['A', 'B', 'C']
Left A , remain 2
Left B , remain 1
Left C , remain 0
list: ['D', 'E', 'F']
Left F , remain 2
Left E , remain 1
Left D , remain 0
```

Više o listama se može pronaći u dokumentaciji.

Slicing

Slicing je način za pristupanje podskupu liste na veoma jednostavan način. Veoma moćna osobina u Python-u.

```
nums = list(range(5))
#range is a built-in function that creates a list of integers
print (nums)
#Prints "[0, 1, 2, 3, 4]"
print (nums[2:4])
# Get a slice from index 2 to 4 (exclusive); prints "[2, 3]"
print (nums[2:])
# Get a slice from index 2 to the end; prints "[2, 3, 4]"
print (nums[:2])
# Get a slice from the start to index 2 (exclusive); prints "[0, 1]"
print (nums[:])
# Get a slice of the whole list; prints ["0, 1, 2, 3, 4]"
print (nums[:-1])
# Slice indices can be negative; prints ["0, 1, 2, 3]"
nums[2:4] = [8, 9]
# Assign a new sublist to a slice
print (nums)
# Prints "[0, 1, 8, 8, 4]"
```

```
[0, 1, 2, 3, 4]
[2, 3]
[2, 3, 4]
[0, 1]
[0, 1, 2, 3, 4]
[0, 1, 2, 3]
[0, 1, 8, 9, 4]
```

Iteriranje kroz listu

```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print (animal)
```

```
cat
dog
monkey
```

```
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
    print ('#%d: %s' % (idx + 1, animal))
```

```
#1: cat
#2: dog
#3: monkey
```

List comprehensions

```
#the following code that computes square numbers
nums = [0, 1, 2, 3, 4]
```

```
squares = []
for x in nums:
    squares.append(x ** 2)
print (squares)
```

```
[0, 1, 4, 9, 16]
```

```
# with list comprehension
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print (squares)
```

```
[0, 1, 4, 9, 16]
```

```
# list comprehension with condition
nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 == 0]
print (even_squares)
```

```
[0, 4, 16]
```

Tuples (Skup)

Syntax:

```
    my_tuple = (a, b, ..., z)
```

```
t = ([1, 2], 4)
t[0].append(3)

print (t)
```

```
([1, 2, 3], 4)
```

Set

```
# Data sets
s1 = set(range(3))
s2 = set(range(10, 7, -1))
s3 = set(range(2, 10, 2))

# Shows the data
print ('s1:', s1, '\ns2:', s2, '\ns3:', s3)

# Union
s1s2 = s1.union(s2)
print ('Union of s1 and s2:', s1s2)

# Difference
print ('Difference with s3:', s1s2.difference(s3))
```

```
# Intersectiono
print ('Intersection with s3:', s1s2.intersection(s3))

# Tests if a set includes the other
if s1.issuperset([1, 2]):
    print ('s1 includes 1 and 2')

# Tests if there is no common elements
if s1.isdisjoint(s2):
    print ('s1 and s2 have no common elements')
```

```
s1: {0, 1, 2}
s2: {8, 9, 10}
s3: {8, 2, 4, 6}
Union of s1 and s2: {0, 1, 2, 8, 9, 10}
Difference with s3: {0, 1, 10, 9}
Intersection with s3: {8, 2}
s1 includes 1 and 2
s1 and s2 have no common elements
```

Numpy

Biblioteka za 'scientific computing'. Veoma je slična matlabu, tako da ako ste familijarni sa MATLAB-om, korisno je pogledati tutorijal.

```
#dodavanje numpy biblioteke
import numpy as np
```

Vektori i matrice

Numpy nudi mogućnost za rad sa vekorima i matricama.

```
print (np.array([1,2,3,4,5,6]))
print (np.array([1,2,3,4,5,6],'d'))
print (np.array([1,2,3,4,5,6],'D'))
```

```
[1 2 3 4 5 6]
[1. 2. 3. 4. 5. 6.]
[1.+0.j 2.+0.j 3.+0.j 4.+0.j 5.+0.j 6.+0.j]
```

```
a = np.array([1, 2, 3])  # Create a rank 1 array
print (type(a), a.shape, a[0], a[1], a[2])
a[0] = 5                 # Change an element of the array
print (a)
```

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

```
#kreiranje matrice kao dva vektora

b = np.array([[0,1],[1,0]],'d')
print (b.shape)
#kreiranje nula matrice
print ('\n nula matrica \n')
print (np.zeros((3,3),'d'))
```

```
(2, 2)

nula matrica

[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

```
#linspace lak nacin za pravljenje koordinata
print (np.linspace(0,1,11))
```

```
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
```

```
#primer mnozenja matrica

print (np.identity(2)*np.ones((2,2)))

print (np.dot(np.identity(2),np.ones((2,2))))
```

```
[[1. 0.]
 [0. 1.]]
[[1. 1.]
 [1. 1.]]
```

```
#transponovanje
m = np.array([[1,2],[3,4]])
m.T
```

```
array([[1, 3],
       [2, 4]])
```

```
#dijagonala
np.diag([1,2,3,4,5])
```

```
array([[1, 0, 0, 0, 0],
       [0, 2, 0, 0, 0],
       [0, 0, 3, 0, 0],
       [0, 0, 0, 4, 0],
       [0, 0, 0, 0, 5]])
```

kreiranje nizova:

```
a = np.zeros((2,2))  # Create an array of all zeros
print (a)
```

```
[[0. 0.]
 [0. 0.]]
```

```
b = np.ones((1,2))   # Create an array of all ones
print (b)
```

```
[[1. 1.]]
```

```
c = np.full((2,2), 7) # Create a constant array
print (c)
```

```
[[7 7]
 [7 7]]
```

```
d = np.eye(2)        # Create a 2x2 identity matrix
print (d)
```

```
[[1. 0.]
 [0. 1.]]
```

```
e = np.random.random((2,2)) # Create an array filled with random values
print (e)
```

```
[[0.67219631 0.59134672]
 [0.44267091 0.75269222]]
```

Indeksiranje nizova

Postoji nekoliko načina za indeksiranje nizova. Takođe, numpy podržava i slicing, uz jedan dodatak da nizovi mogu biti višedimenzionalani pa se mora zadati slice za svaku dimenziju.

```
# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]
print (b)
```

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

Slice nekog niza predstavlja pogled na iste podatke, tako da će se svaka modifikacija odraziti na originalni niz.

```
print (a[0, 1])
b[0, 0] = 77    # b[0, 0] je isti podatak kao a[0, 1]
print (a[0, 1])
```

```
2
77
```

Mix integer indeksiranja i slice indeksiranja.

Napomena Nije isto kao u MATLAB-u

```
# Create the following rank 2 array with shape (3, 4)
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
print (a)
```

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

Selekcija podataka u srednjem redu. Obratiti pažnju na rang niza koji je predstavljen kao rezultat.

```
row_r1 = a[1, :]    # Rank 1 view of the second row of a
row_r2 = a[1:2, :]  # Rank 2 view of the second row of a
row_r3 = a[[1], :]  # Rank 2 view of the second row of a
print (row_r1, row_r1.shape)
print (row_r2, row_r2.shape)
print (row_r3, row_r3.shape)
```

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

Isto važi i za kolone

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

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

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

Integer indeksiranje

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

# An example of integer array indexing.
# The returned array will have shape (3,) and
print (a[[0, 1, 2], [0, 1, 0]])

# The above example of integer array indexing is equivalent to this:
print (np.array([a[0, 0], a[1, 1], a[2, 0]]))
```

```
[[1 2]
 [3 4]
 [5 6]]
[1 4 5]
[1 4 5]
```

```
# When using integer array indexing, you can reuse the same
# element from the source array:
print (a[[0, 1], [1, 1]])

# Equivalent to the previous integer array indexing example
print (np.array([a[0, 1], a[1, 1]]))
```

```
[2 4]
[2 4]
```

Trik za selekciju ili izmenu jednog elementa iz svakog reda

```
# Create a new array from which we will select elements
a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
print (a)
```

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

```
# Create an array of indices
b = np.array([0, 2, 0, 1])

# Select one element from each row of a using the indices in b
print (a[np.arange(4), b])  # Prints "[ 1  6  7 11]"
```

```
[ 1 6 7 11]
```

```
# Mutate one element from each row of a using the indices in b
```

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

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

Boolean indeksiranje koristi se za selekciju elementata koji zadovoljavaju neki uslov

```
a = np.array([[1,2], [3, 4], [5, 6]])
print (a)
print ()
bool_idx = (a > 2)  # Find the elements of a that are bigger than 2;
                    # this returns a numpy array of Booleans of the same
                    # shape as a, where each slot of bool_idx tells
                    # whether that element of a is > 2.

print (bool_idx)
```

```
[[1 2]
 [3 4]
 [5 6]]
[[False False]
 [ True  True]
 [ True  True]]
```

```
# We use boolean array indexing to construct a rank 1 array
# consisting of the elements of a corresponding to the True values
# of bool_idx
print (a[bool_idx])

# We can do all of the above in a single concise statement:
print (a[a > 2])
```

```
[3 4 5 6]
[3 4 5 6]
```

Tipovi podataka

Svaki numpy niz je mreža elemenata istog tipa. Numpy nudi veliki skup numeričkih tipova podataka koji se mogu koristiti za pravljenje nizova. Ako se ne navede tip podataka, numpy pokušava da pogodi o kom tipu se radi. Obično se prilikom pravljenja niza navede i koji je tip podataka.

```
x = np.array([1, 2])  # Let numpy choose the datatype
y = np.array([1.0, 2.0])  # Let numpy choose the datatype
z = np.array([1, 2], dtype=np.int64)  # Force a particular datatype
```

```
print (x.dtype, y.dtype, z.dtype)
```

```
int64 float64 int64
```

Više o tipovima podataka u Numpy biblioteci se može pronaći na linku.

Matematičke operacije

Osnovne matematičke operacije se primenjuju na svaki element u nizu. Operacije su dostupne kao preklopljeni operator i kao funkcija.

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

# Elementwise sum; both produce the array
print (x + y)
print (np.add(x, y))
```

```
[[ 6.  8.]
 [10. 12.]]
 [[ 6. 8.]
 [10. 12.]]
```

```
# Elementwise difference; both produce the array
print (x - y)
print (np.subtract(x, y))
```

```
[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
```

```
# Elementwise product; both produce the array
print (x * y)
print (np.multiply(x, y))
```

```
[[ 5. 12.]
 [21. 32.]]
[[ 5. 12.]
 [21. 32.]]
```

```
# Elementwise division; both produce the array
# [[ 0.2         0.33333333]
#  [ 0.42857143  0.5       ]]
print (x / y)
print (np.divide(x, y))
```

```
[[0.2 0.33333333]
 [0.42857143 0.5 ]]
[[0.2 0.33333333]
 [0.42857143 0.5 ]]
```

```
# Elementwise square root; produces the array
# [[ 1.          1.41421356]
#  [ 1.73205081  2.        ]]
print (np.sqrt(x))
```

```
[[1. 1.41421356]
 [1.73205081 2. ]]
```

Za razliku od MATLAB-a * je operator koji množi svaki element, i ne predstavlja matrično množenje. Za matrično množenje, Numpy ima funkciju `dot`, koja je dostupna kao zasebna funkcija i kao funkcija samog niza.

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

v = np.array([9,10])
w = np.array([11, 12])

# Inner product of vectors; both produce 219
print (v.dot(w))
print (np.dot(v, w))
```

```
219
219
```

```
# Matrix / vector product; both produce the rank 1 array [29 67]
print (x.dot(v))
print (np.dot(x, v))
```

```
[29 67]
[29 67]
```

```
# Matrix / matrix product; both produce the rank 2 array
# [[19 22]
#  [43 50]]
print (x.dot(y))
print (np.dot(x, y))
```

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

Pored standardnih matematičkih operacija, Numpy nudi veoma korisne funkcije za izračunavanje nad nizovima.

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

print (np.sum(x))  # Compute sum of all elements; prints "10"
print (np.sum(x, axis=0))  # Compute sum of each column; prints "[4 6]"
print (np.sum(x, axis=1))  # Compute sum of each row; prints "[3 7]"
```

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

Više o matematičkim operacijama na linku.

Matplotlib

Matplotlib je biblioteka za plotovanje. U ovom delu će biti predstavljen deo ove biblioteke `matplotlib.pyplot`, koji nudi slične mogućnosti kao MATLAB.

```
import matplotlib.pyplot as plt
```

Da bi se slike prikazivale unutar samog notebook-a, koristimo sledeću komandu (potrebno samo za ipython)

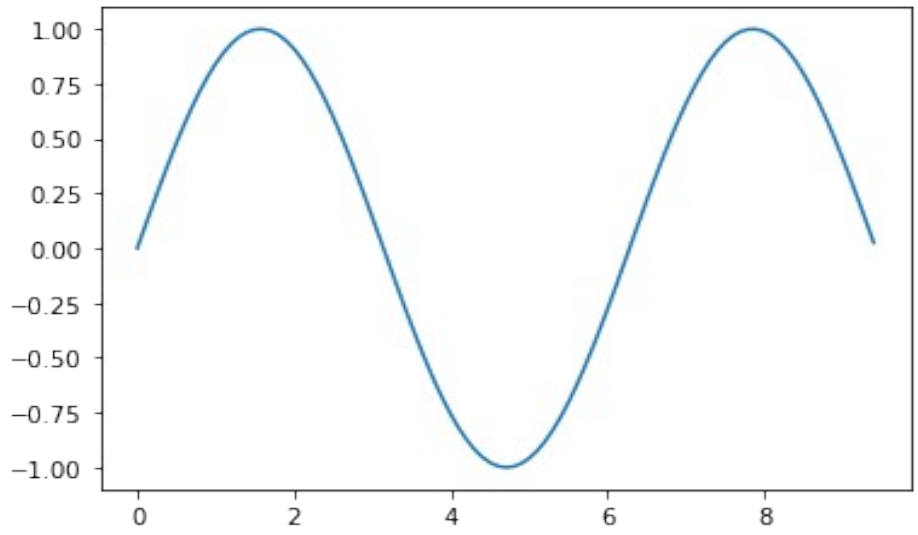
```
%matplotlib inline
```

Plot 2D podataka

```
# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)

# Plot the points using matplotlib
plt.plot(x, y_sin)
```

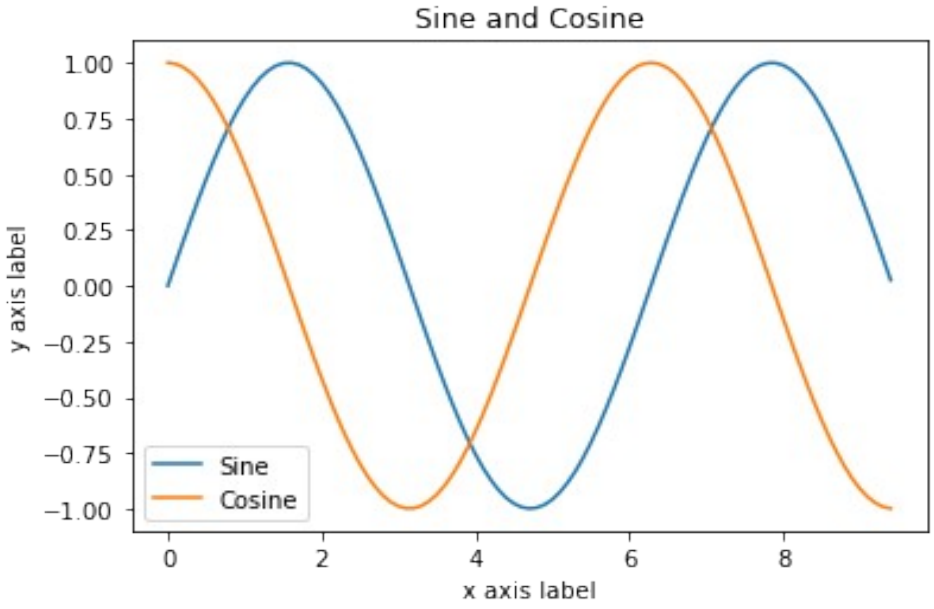
```
[<matplotlib.lines.Line2D at 0x7fa4a5b517b8>]
```




```
y_cos = np.cos(x)

# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
```

```
<matplotlib.legend.Legend at 0x7fa4a548f438>
```



Moguće je plotovati raličite stvari unutar iste slike

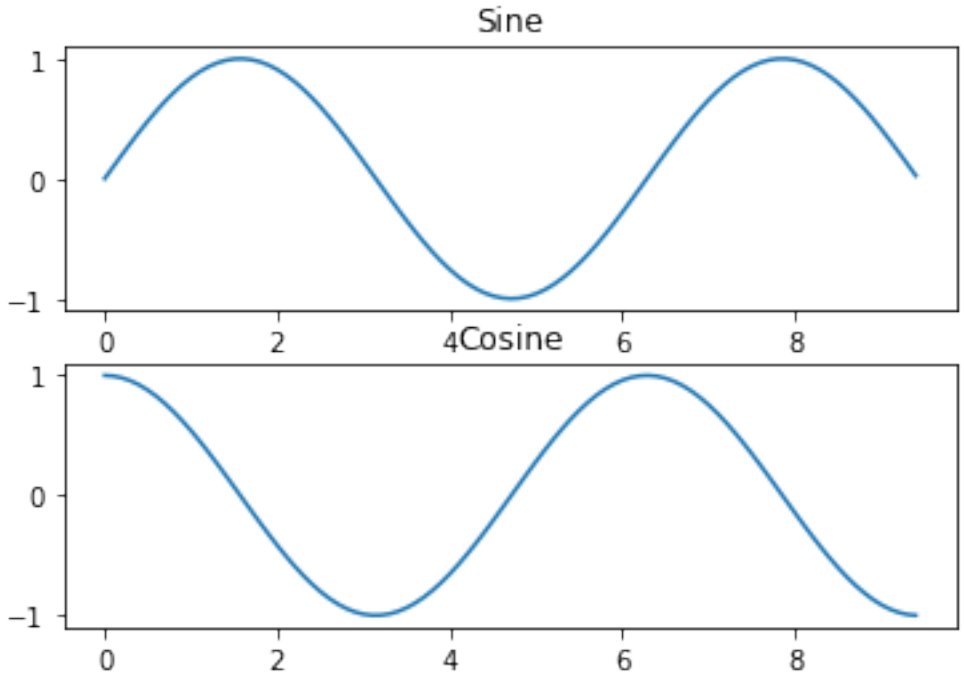
```
# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')

# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Show the figure.
plt.show()
```



Više o samoj biblioteci se može pronaći na linku.