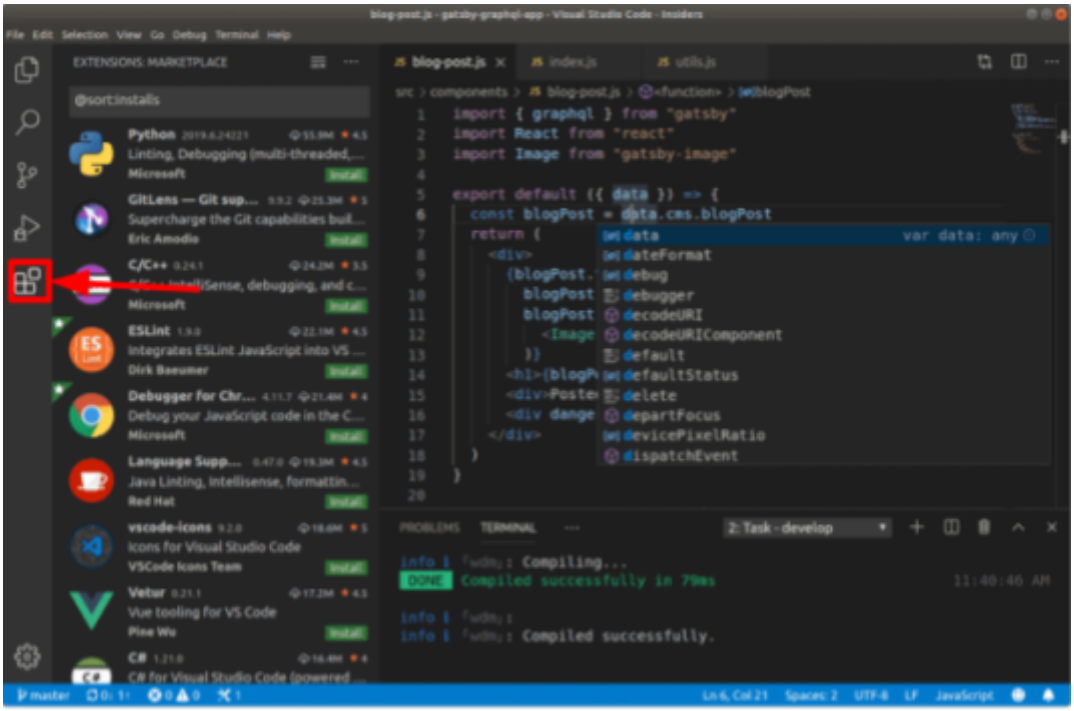
Python uvod

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

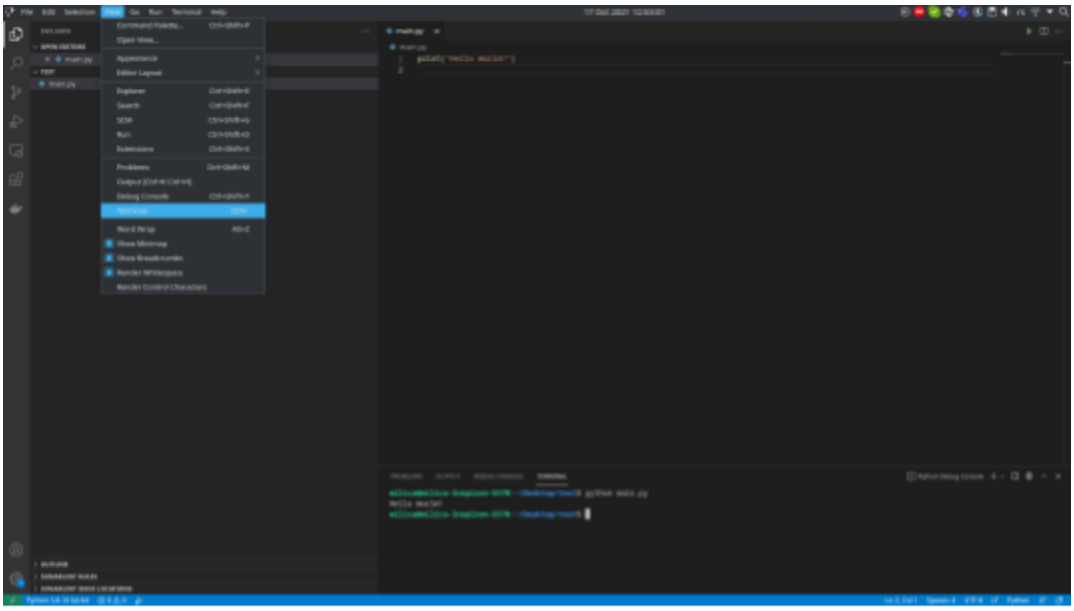
Alati

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)
```

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 (~)

Napomena:

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

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
```