

# OOP in Python

Silviu Ojog - <https://www.youtube.com/@SilviuOjog>

***LINK*Academy**



# Ce am făcut până acum?

# Python

- Ce știm până acum despre Python?
  - Limbaj interpretat?
  - Python2 vs Python3?
  - Unde este folosit?
  - Limbaj obiect orientat? (Object-Oriented-Programming)

# Funcția

- Funcția este o unitate separată în program care se poate executa la cererea unei alte părți a programului.
- Funcția facilitează codarea și reduce cantitatea de cod repetat.

# Tip de date

Alegeți varianta/ele corectă/e a output-ului?

*Care dintre variantele următoare nu reprezintă un tip de date de bază?*

- a) Listă (List)
- b) Dicționar (Dictionary)
- c) Tuplu (Tuple)
- d) Clasă (Class)

# Tip de date

Alegeți varianta/ele corectă/e a output-ului?

*Care dintre variantele următoare nu reprezintă un tip de date de bază?*

- a) Listă (List)
- b) Dicționar (Dictionary)
- c) Tuplu (Tuplu)
- d) **Clasă (Class)** #clasele sunt definite de utilizatori

# Type

Completați propoziția:

Este declarația unui dicționar.

a) `a = ()`

b) `a = []`

c) `a = {}`

# Type

Completați propoziția:

Este declarația unui dicționar.

a) `a = ()`

b) `a = []`

c) `a = {}`

R:

c) `a = { }` # reprezintă dicționar

Silviu Ojog



# Type

Completați propoziția:

Un dicționar în Python este ....

a) mutabil

b) imutabil

# Type

Completați propoziția:

Un dicționar în Python este ....

a) mutabil

b) imutabil

# Colecții

Care variantă este corectă?

- \_\_\_ *stochează mai multe valori mutabile*
- \_\_\_ *stochează mai multe valori imutabile*
- \_\_\_ *stochează mai multe valori unice*
- \_\_\_ *stochează mai multe perechi cheie valoare*

- a) Dictionary    Set    Tuple    List
- b) Tuple    Dictionary    List    Set
- c) Set    Tuple    List    Dictionary
- d) List    Tuple    Set    Dictionary

# Colecții

Care variantă este corectă?

- \_\_\_ *stochează mai multe valori mutabile*
- \_\_\_ *stochează mai multe valori imutabile*
- \_\_\_ *stochează mai multe valori unice*
- \_\_\_ *stochează mai multe perechi cheie valoare*

- a) Dictionary    Set    Tuple    List
- b) Tuple    Dictionary    List    Set
- c) Set    Tuple    List    Dictionary
- d) List    Tuple    Set    Dictionary

# Dictionar

Ce se va printa în consolă?

```
johannis = {'cars':2 , 'houses':6}
```

```
johannis['cars'] = 3
```

```
print(johannis)
```

a) [3, 6]

b) Syntax Error

c) {'cars':2 , 'houses':6, 'cars':3 }

d) {'cars':3 , 'houses':6}

# Dictionar

Ce se va printa în consolă?

```
johannis = {'cars':2 , 'houses':6}
```

```
johannis['cars'] = 3
```

a) [3, 6]

b) Syntax Error

c) {'cars':2 , 'houses':6, 'cars':3 }

d) {'cars':3 , 'houses':6}

# Boolean

Ce se va printa în consolă?

```
x = {"Hello" : ""}  
print(bool(x))
```

- a) False
- b) Hello
- c) True
- d) Type Error

# Boolean

Ce se va printa în consolă?

```
x = {"Hello" : ""}  
print(bool(x))
```

- a) False
- b) Hello
- c) **True**
- d) Type Error



# Boolean

Ce se va printa în consolă?

```
x = {[] : "Hello"}  
print(bool(x))
```

- a) False
- b) Hello
- c) True
- d) Type Error

# Boolean

Ce se va printa în consolă?

```
x = {[] : "Hello"}  
print(bool(x))
```

- a) False
- b) Hello
- c) True
- d) **Type Error # unhashable type: 'list'**

# Boolean

Ce se va printa în consolă?

```
x = {0 : None}  
print(bool(x))
```

- a) False
- b) Hello
- c) True
- d) Type Error

# Boolean

Ce se va printa în consolă?

```
x = {0 : None}  
print(bool(x))
```

- a) False
- b) Hello
- c) **True**      # cheia poate fi integer, valoarea orice
- d) Type Error

# Boolean

Ce se va printa în consolă?

```
x = {None : None}  
print(bool(x))
```

- a) False
- b) Hello
- c) True
- d) Type Error

# Boolean

Ce se va printa în consolă?

```
x = {0 : None}
```

```
print(bool(x))
```

a) False

b) Hello

c) **True**      # cheia poate fi orice valoare imutabila  
valoarea orice fel de valoare (mutabilă sau imutabilă)

d) Type Error



# Funcția

- Funcțiile pot exista **individual**
  - *def my\_function(x):*  
*return print(x)*
- sau **definite de utilizator**.
  - *def my\_function(x):*  
*return x\*\*2*
  - *def other\_function():*  
*my\_function(20)*

# Funcții încorporate

- Sursă <https://docs.python.org/3/library/functions.html>
  - În timpul executiei, interpretorul Python are la dispoziție anumite funcții încorporate, care se pot folosi imediat.
  - Unele dintre aceste funcții au fost explicate în lecțiile anterioare, unele sunt **suficient de intuitive** încât nu necesita explicatie.





# Funcții încorporate

<code>abs()</code>	<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>
<code>all()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>any()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>ascii()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>bin()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bool()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>breakpoint()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	



# Funcții încorporate

Function Name	% of Projects	Total # of Uses	% of All Builtins	Cumulative %
len	68.02	222626	15.91	15.91
print	50.1	170384	12.18	28.09
format	45.59	124423	8.89	36.98
isinstance	45.69	104212	7.45	44.44
str	54.91	90316	7.03	51.46
int	51.34	67493	4.82	56.29
range	45.61	65060	4.65	60.94
open	60.09	48092	3.44	64.38
list	41.95	47710	3.41	67.79
super	38.01	46163	3.3	71.09
set	30.55	34706	2.48	73.57
dict	34.08	32770	2.34	75.91
getattr	33.57	28820	2.07	77.98
hasattr	30.07	26279	1.88	79.86
type	27.06	25280	1.81	81.68
float	25.36	23165	1.66	83.32
enumerate	31.72	19791	1.41	84.73
sorted	26.81	15846	1.14	85.87
max	20.68	12685	0.91	86.78
repr	15.01	12059	0.86	87.64
zip	20.72	11994	0.86	88.5
tuple	19.77	11984	0.86	89.36
map	20.52	10239	0.73	90.09
min	18.51	9718	0.69	90.78

- Cele mai folosite funcții Python încorporate, în proiectele de pe Github

# Exercițiu

- Găsiți cel mai mic și cel mai mare număr dintr-o listă neordonată
  - $l = [42, 13, 61, 2, -42, 868, 1, -11, -32, 300]$



# Exercițiu

- Găsiți cel mai mic și cel mai mare număr dintr-o listă neordonată
  - `l = [ 42 , 13 , 61 , 2 , -42, 868 , 1, -11 , -32, 300]`
  - `print(max(l)) # 868`
  - `print(min(l)) # -42`



# Funcția

- Funcțiile pot exista **individual**
  - *def my\_function(x):*  
*return print(x)*
- sau **definite de utilizator**.
  - *def my\_function(x):*  
*return x\*\*2*
  - *def other\_function():*  
*my\_function(20)*



# Funcția

- Funcțiile pot fi **încorporate**
  - `print("something")`
  - `x = input ("Introduceți un număr")`
- sau pot exista în cadrul unor obiecte și sunt numite **metode**.
  - `class Calculator:`
    - `def power_of(x):`
      - `return x**2`
    - `def is_zero(x):`
      - `return x==0`

# Crearea funcției

- Cuvântul-cheie pentru crearea funcției este **def**.

```
def power_of(x):  
    return x**2
```

- Parantezele rotunde după denumirea funcției sunt obligatorii și pot fi goale sau pot conține parametri de intrare ai funcției.

```
x = 0  
def doSomething():  
    print("Test function")
```

- Corpul funcției conține linii cu **indentare**. Două puncte reprezintă începutul corpului funcției.

Silviu Ojog

LINK Academy



# Apelarea funcției

- Funcția se apelează cu **numele funcției** după care urmează **parantezele rotunde**.

```
def doSomething():  
    print("Test function")
```



```
doSomething()
```



# Apelarea funcției

- !!!! Funcția trebuie definită ca să poată fi apelă

*doSomething()*



```
def doSomething():  
    print("Test function")
```

# Parametrii funcției

- Funcția poate primi parametri în timpul executiei.
- Parametri se marchează în paranteze rotunde, în semnătura funcției.

```
def power_of(x):  
    return x**2
```

- O astfel de funcție parametrizată se apelează parametrizat.

```
power_of(14)
```



# Parametrii funcției

- Funcția poate primi parametri în timpul executiei.
- Parametri se marchează în paranteze rotunde, în semnătura funcției.

```
def sum(x, y, z):
```

```
    return x+y+z
```

- O astfel de funcție parametrizată se apelează parametrizat.

```
h = sum(3,4,5)
```

# Parametrii funcției

- Apelarea fără parametri nu este permisă și provoacă erori.

```
def sum(x, y, z):  
    return x+y+z
```



`sum(2)`  $\longrightarrow$  `TypeError (missing argument)`

# Parametri implicați

- Parametri implicați sunt parametri care pot, dar nu neapărat, să se transmită unei funcții. Acești parametri trebuie marcați special, în așa fel încât le va fi atribuită valoarea implicită.

```
def power_of(x = 10):  
    return x**2
```

- O astfel de funcție parametrizată se poate apela
  - parametrizat.

`power_of(14)` #În acest caz,  $x = 14$

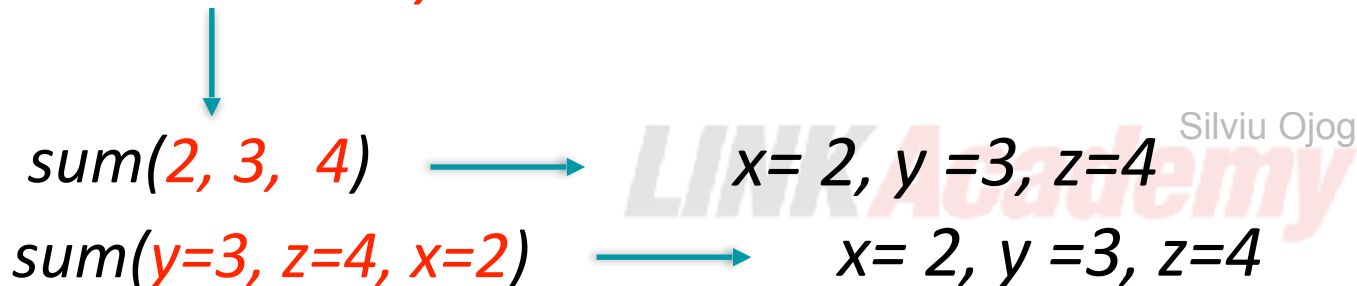
- Cu parametrul implicit.

`power_of()` #În acest caz,  $x = 10$

# Parametri denumiți

- În timpul apelării, parametri se pot transmite folosind ordinea lor în semnătura funcției, dar, de asemenea, folosind și denumirile lor

```
def sum(x, y, z):  
    return x+y+z
```


  
`sum(2, 3, 4)`       $x=2, y=3, z=4$ 
  
`sum(y=3, z=4, x=2)`       $x=2, y=3, z=4$

# Parametri impliciti

```
def f(a, b=2, c=3):  
    print(a, b , c)
```

```
f(1)          # 1 2 3  
f(1, b=0)     # 1 0 3  
f(1, c=0)     # 1 2 0  
f(1, c=0, b=5) # 1 5 0
```

```
# f(b=0, 1) #generează eroarea:  
# SyntaxError: non-keyword arg after keyword arg
```

```
f(b=0, a=1)   # 1 0 3
```

# Exercițiu

Verificați dacă două stringuri sunt anagrame/palindroame:

```
def este_palindrom(s):
```

```
    pass
```

```
def sunt_anagrame(s1, s2):
```

```
    pass
```

epurasuuserupe

elefaccafele

caiac

roma -> amor



# Exercițiu

Verificați dacă două stringuri sunt anagrame

```
def are_palindrom(s1, s2):  
    return set(s1) == set(s2)
```



# Exercițiu

Verificați dacă două stringuri sunt palindroame:

```
def are_palindrom(s1, s2):  
    return s1 == s2[::-1]
```