

Python

Cuprins

Instalare și configurare python

Introducere în python

Variabile

Operatori

Inputs

Operații cu string-uri

Liste

Instrucțiunea 'if'

Instrucțiunea 'while'

Instrucțiunea 'for'

Matrice bidimensională

Listă, tuple, set și dicționar

Funcții

De instalat python și vscode

Tool-uri utile pentru acest curs:

[VSCode](#) + Python for VSCode plugin

[Python](#)

[Online python compiler](#)



De ce folosim python?

Pe lângă dezvoltarea web și software, Python este folosit pentru analiză datelor, învățarea automată sau pentru proiectare.

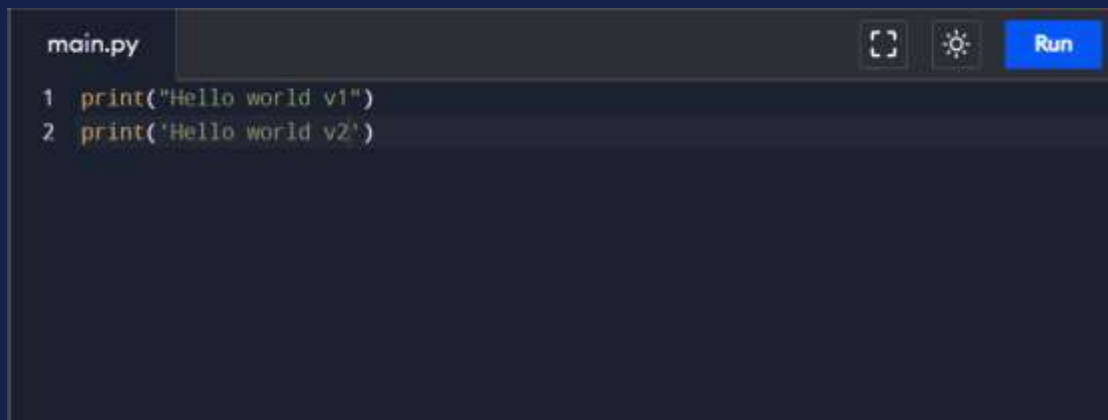
Sintaxa este foarte simplă, ușor de învățat accentuând lizibilitatea.

Python este una dintre tehnologiile principale utilizate de echipele care practică DevOps.

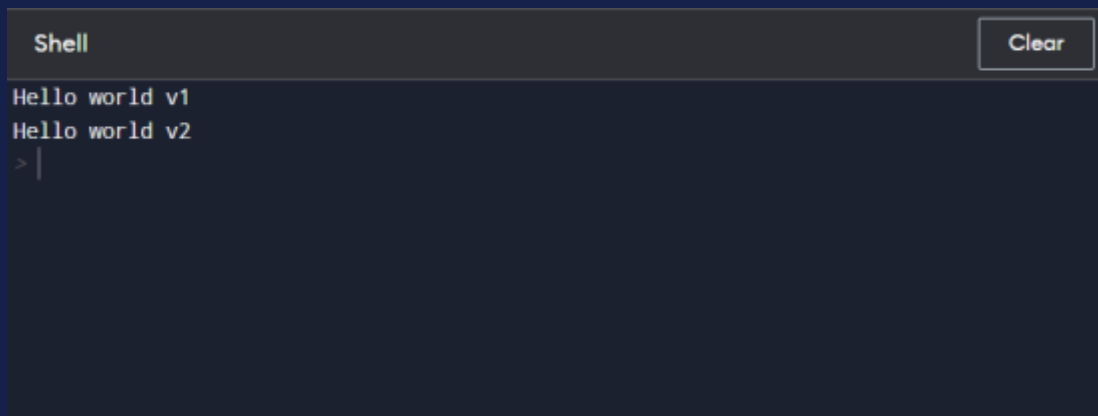


Primii pași în python

Prin convenție, toate fișierele python vor avea extensia **.py** .
Pentru a afișa un text în python, putem folosi comanda `print()`





```
main.py
1 print("Hello world v1")
2 print('Hello world v2')
```



```
Shell
Hello world v1
Hello world v2
> |
```

Primii pași în python

main.py



Run

```
1 print('mesaj ' * 3)
2
```

Shell

Clear

```
mesaj mesaj mesaj
> |
```

Comentarii

Comentariile sunt comenzi neexecutabile în Python și ele sunt destinate înțelegerii umane pentru a ne fi nouă, dar și altor programatori mai ușor să urmărim ce se întâmplă în cod .

main.py



Run

```
1 #acesta este un comentariu si nu se va afisa la rularea codului  
2 print("hello world")
```

Shell

Clear

hello world

> |

Variabile

```
main.py  [ ] [ ] [Run]
1  numar = 25
2  print("variabila are valoarea: ",numar)
```

```
Shell  [Clear]
variabila are valoarea: 25
>
```

În acest mod putem atribui valori unei variabile în python. De asemenea, o putem integra în comanda `print()` și a o separa de text-ul afișat .

Suma a două numere

main.py



Run

```
1  numar1 = 25
2  numar2 = 30
3  suma = numar1+numar2
4  print("suma celor doua numere este: ")
5  print(suma)
6  print("Suma se putea afisa si pe aceeaasi linie ca exemplul precedent: ",suma)
7
```

Shell

Clear

suma celor doua numere este:

55

Suma se putea afisa si pe aceeaasi linie ca exemplul precedent: 55

> |

Suma a două numere de tip float

main.py



Run

```
1 numar1 = 2.4
2 numar2 = 10.1
3 suma = numar1+numar2
4 |
5 print("Suma este: ",suma)
6
```

Shell

Clear

Suma este: 12.5

> |

Operatori

Operatorul de adunare, poate fi simplificat din $x=x+3$ cu $x += 3$
Acest lucru este valabil și la operatorul de diferență $y=y+10$ cu $y -= 10$

Operatorul înmulțire “*” și diviziune “/” înmulțesc și respectiv împart două numere.

Operatorul modulo “%” returnează restul împărțirii primului număr la alt număr .

Ultimul operator “**” este cel exponențial. Pentru a calcula x^4 putem scrie
 $x = x^{**}4$

Operații matematice

main.py



Run

```
1 x = 20
2 y = 15
3 print("X: ",x," si ","Y: ",y)
4
5 x += 10
6 y -= 3
7 print("X: ",x," si ","Y: ",y)
8
9 x = x%4
10 y = y / 4
11 print("X: ",x," si ","Y: ",y)
12
13
14 x = x * 3
15 y = y ** 3
16 print("X: ",x," si ","Y: ",y)
17
```

Shell

Clear

```
X: 20 si Y: 15
X: 30 si Y: 12
X: 2 si Y: 3.0
X: 6 si Y: 27.0
>
```



Tipuri de variabile în python

Python are mai multe tipuri standard de variabile:

- Numar ex: 4, 10 , 50 , 12.5 , -54.5
- String ex: 'apple' , 'hello world'
- Array ex `array_1 = arr.array("i", [3, 6, 9, 12])`
- Lista ex: [54,65,123,32,10,-4] sau ['random text', 43 , 2.23, 'second text']
- Tuple ex: ('abc', 43 , 54, 'efg', 70)
- Dictionary ex: {'name': 'john','code': 123, 'dept': 'hr'}
- Boolean ex: True sau False
- Seturi ex: {"apple", "banana", "cherry"}

Input în Python

main.py



Run

```
1 age = input("cati ani ai? ")
2 name = input("cum te numesti? ")
3 print("numele meu este ",name," si am varsta de: ",age," ani")
```

Shell

Clear

```
cati ani ai? 22
cum te numesti? Mircea
numele meu este  Mircea  si am varsta de:  22  ani
> |
```

`input ()` : Această funcție preia mai întâi intrarea de la utilizator și apoi evaluează expresia, ceea ce înseamnă că Python identifică automat dacă utilizatorul a introdus un șir, un număr sau o listă.



String-uri

Șirurile din python sunt înconjurate fie de ghilimele simple, fie de ghilimele duble . Putem afișa caracter cu caracter din string-ul text. Asemănător cu listele, se începe de la indexul 0. Astfel, dacă vrem să afișăm primul caracter al unui string se află în `text[0]` și este corespunzător caracterului "w".

main.py

```
1 text = "welcome to the course"
2 print(text)
3 print(text[0])
4 print(text[1])
5 print(text[2])
6 print(text[3])
7 print(text[4])
8 print(text[5])
9 print(text[6])
```

String-uri

```
Shell Clear  
welcome to the course  
w  
e  
l  
c  
o  
m  
e  
>
```

Ce se întâmplă în cazul în care se afișează `text[-1]` sau `text[-2]` ?
Dar în cazul în care se afișează `text[0:6]` ?

Concatenarea

Față de exemplul precedent când “+” era interpretat ca adunare atunci când variabilele erau de tip numeric, acum compilatorul de python știe că sunt string și le interpretează ca atare. Astfel, “+” înseamnă alăturarea string-urilor într-un string mai mare “z”.

main.py

Run

```
1 x = "Python is "  
2 y = "awesome"  
3 z = x + y  
4 print(z)
```

Shell

Clear

```
Python is awesome  
> |
```

Lungimea unui string

Funcția **len()** returnează numărul de caractere din șir .

Exemplu:

```
text = "Hello"  
print(len(text))
```

Va afișa numărul 5, corespunzător cu numărul de caractere din string-ul de mai sus ("Hello") .

Această funcție poate fi utilă în condițiile în care vrem să parcurgem caracter cu caracter dintr-un șir într-o structură repetitivă de tip **for** sau **while** .

Operații cu șiruri

Python are un set de metode încorporate ce pot fi folosite pentru string-uri. Scopul lor este acela de face mai ușoare operațiunile cu ele .

find() Caută în șir o valoare specificată și returnează poziția unde a fost găsită aceasta

upper() Convertește un șir în majuscule

lower() Convertește toate caracterele dintr-un șir în litere mici

count() Returnează de câte ori apare o valoare specificată într-un șir

replace() Returnează un șir în care o valoare specificată este înlocuită cu o valoare specificată

Alte metode utile de pot vedea aici: [String methods](#)



Operații cu șiruri

```
text = "HELlo World"  
print(text.upper())  
print(text.lower())  
print(text)  
  
print(text.count("o"))  
print(text.find("E"))  
print(text.replace("World", "People"))
```

```
HELLO WORLD  
hello world  
HELlo World  
2  
1  
HELlo People
```

Liste

O listă este o variabilă specială, care poate conține mai multe valori simultan . Dacă aveți mai multe numere stocarea acelor valori în variabile individuale ar putea complica lucrurile. Soluția cea mai la îndemână este să folosim o listă și să punem acolo valorile. Astfel, le putem parcurge, sorta , modifica și șterge mai ușor .

Exemplu:

```
Mylist = ["George", "Mircea", "Adrian", "Denisa", "Vlad"]
```

Indexul primului element din listă va fi mereu 0. Astfel, Mylist[0] va avea valoarea "George", precum și Mylist[1] va avea valoarea "Mircea".

Precum în exemplele precedente cu string-uri și în cazul listelor avem diferite metode ce ne ajută să manipulăm și să prelucrăm valorile mai ușor .

Liste

Printre cele mai cunoscute metode sunt:

- count() Returnează numărul de elemente cu valoarea specificată
- index() Returnează indexul primului element cu valoarea specificată
- insert() Adaugă un element la poziția specificată
- pop() Îndepărtează elementul din poziția specificată
- reverse() Inversează ordinea listei
- sort() Sortează lista

Liste

main.py



Run

```
1 Myarray = ["George","Mircea","Adrian","Denisa","Vlad"]
2
3 print(Myarray[0]) # valoarea primului element
4 print(Myarray.count("Denisa")) # de cate ori apare cuvantul 'Denisa' in array
5 print(Myarray.index("George")) # Indexul primului element cu valoarea 'George'
6
7 Myarray.insert(1,"Daniel") # se insereaza pe pozitia 1, elementul cu valoarea 'Daniel'
8 print(Myarray) # array-ul isi va schimba continul si va avea valoarea adaugata mai devreme
9
10 Myarray.pop(5) # stergem elementul cu index-ul 5
11 print(Myarray)
12
13 Myarray.reverse() #inversam ordinea elementelor din array
14 print(Myarray)
15
16 Myarray.sort() #sortam elementele in ordin alfabetica
17 print(Myarray)
```

Shell

Clear

George

```
1
0
['George', 'Daniel', 'Mircea', 'Adrian', 'Denisa', 'Vlad']
['George', 'Daniel', 'Mircea', 'Adrian', 'Denisa']
['Denisa', 'Adrian', 'Mircea', 'Daniel', 'George']
['Adrian', 'Daniel', 'Denisa', 'George', 'Mircea']
> |
```



Liste

Listele sunt folosite pentru a stoca mai multe articole într-o singură variabilă. Diferența față de array-uri este faptul că într-o listă pot fi elemente de mai multe tipuri de date, pe când în array-uri nu puteam avea decât elemente omogene (de același tip) .

Lista este modificabilă, ceea ce înseamnă că putem modifica, adăuga și elimina elemente dintr-o listă după ce aceasta a fost creată .

Liste

main.py



Run

```
1 lista1 = ["apple", "banana", 5, 10 , 11]
2 print("lista 1 este: ", lista1)
3 lista2 = [5,6,7]
4 print("lista 2 este: ", lista2)
5 lista = lista1 + lista2
6 print("ambele liste: ", lista)
7 print("Tipul acestei variabile este :", type(lista))
```

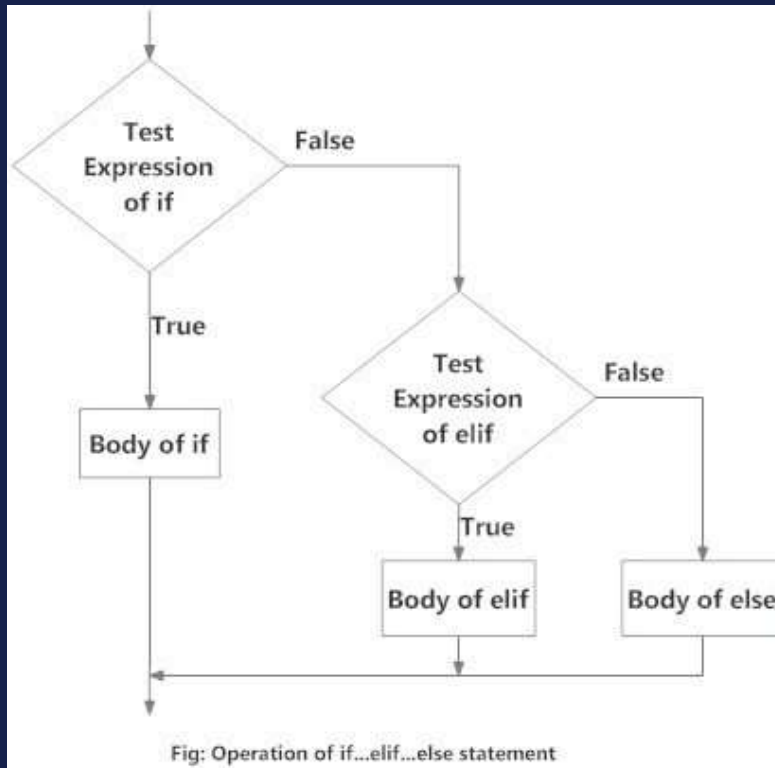
Shell

Clear

```
lista 1 este: ['apple', 'banana', 5, 10, 11]
lista 2 este: [5, 6, 7]
ambele liste: ['apple', 'banana', 5, 10, 11, 5, 6, 7]
Tipul acestei variabile este : <class 'list'>
```

Instrucțiunea 'if'

Instrucțiunea 'if' ajută la luarea deciziei atunci când dorim să executăm un cod numai dacă o anumită condiție este îndeplinită .



Sintaxa:

if test expression:
statement(s)

Instrucțiunea 'if'

main.py



Run

```
1 x = 10
2 if x > 0:
3     print("numarul este pozitiv: ",x)
4
```

Shell

Clear

```
numarul este pozitiv: 10
> |
```

Instrucțiunea 'if-else'

main.py

```
1 x = -10
2 if x > 0:
3     print("numarul este pozitiv: ",x)
4 else:
5     print("numarul este 0 sau negativ: ",x)
```



Run

Shell

Clear

numarul este 0 sau negativ: -10

> |

Instrucțiunea 'if-elif-else'

main.py



Run

```
1 x = 0
2 if x > 0:
3     print("numarul este pozitiv: ",x)
4 elif x == 0:
5     print("numarul este egal cu 0: ",x)
6 else:
7     print("numarul este negativ: ",x)
```

Shell

Clear

numarul este egal cu 0: 0

> |

Multiple condiții pentru 'if'

Pentru o filtrare mai clară a condițiilor este necesar să punem condiții multiple .
Pentru asta ne folosim de 'and' sau 'or' .

Sintaxa:

```
if (cond1 AND/OR COND2) AND/OR (cond3 AND/OR cond4):  
    code1  
else:  
    code2
```

AND

| A | B | Y |
|---|---|---|
| 1 | 1 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 0 | 1 |

OR

| A | B | Y |
|---|---|---|
| 1 | 1 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 0 | 1 |

Multiple condiții pentru 'if'

```
main.py  [Full Screen] [Theme] [Run]

1  varsta = 18
2
3  if ((varsta >= 0) and (varsta <= 17)):
4      print("Nu ai varsta necesara pentru a vota !")
5  else:
6      print("Poti merge la vot !")
7
```

```
Shell  [Clear]

Poti merge la vot !
> |
```

Instrucțiunea while

Cu bucla while putem executa un set de instrucțiuni atâta timp cât o condiție este adevărată .

Sintaxa:

```
while test_expression:  
    Body of while
```


Instrucțiunea while

main.py



Run

```
1 i = 1
2 while i < 10:
3     print("numarul este: ",i)
4     i += 1
```

Shell

Clear

```
numarul este: 1
numarul este: 2
numarul este: 3
numarul este: 4
numarul este: 5
numarul este: 6
numarul este: 7
numarul este: 8
numarul este: 9
```

Break și continue

În Python, instrucțiunile 'break' și 'continue' pot modifica fluxul unei bucle normale .

Folosim 'break' și 'continue' pentru că uneori dorim să încheiem iterația curentă sau chiar întreaga buclă fără a verifica expresia de testare .

Break

Comanda '**break**' încheie din care face parte .

main.py



Run

```
1 x=0
2 while x < 10:
3     if x == 5:
4         print("break va intrerupe instrutiunea while pentru ca x este 5")
5         break
6     print(x)
7     x=x+1
```

Shell

Clear

```
0
1
2
3
4
break va intrerupe instrutiunea while pentru ca x este 5
> |
```

Continue

Cu instrucțiunea '**continue**' putem opri iterația curentă și continua cu următoarea .

```
main.py ⌵ ☀ Run  
1 x = 0  
2 while x < 5:  
3     x += 1  
4     if x == 2:  
5         print("nu se va afisa acest numar, deoarece folosim 'continue', while-ul va merge mai  
           departe la urmatoarea iteratie")  
6         continue  
7     print(x)
```



```
Shell Clear  
1  
nu se va afisa acest numar, deoarece folosim 'continue', while-ul va merge mai departe la  
    urmatoarea iteratie  
3  
4  
5
```

Instrucțiunea 'for'

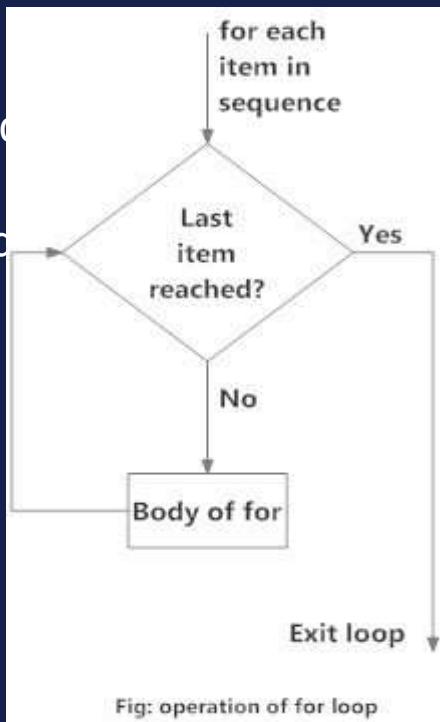
O buclă 'for' este utilizată pentru iterarea unei secvențe (adică fie o listă, tuple, un dicționar, un set sau un șir) .

Sintaxa:

val in sec

loop bod

for



Instrucțiunea 'for'

main.py



Run

```
1 lista = [1,43,10,54,102]
2 for i in lista:
3     print(i)
```

Shell

Clear

```
1
43
10
54
102
```

Range

Funcția `range()` returnează o secvență de numere, începând de la 0 în mod implicit și crește cu 1 (în mod implicit) și se oprește înaintea unui număr specificat .

Sintaxa:

`range(start, stop, step)`

Exemplu:

```
for i in range(3):  
    print(i)
```

Va afișa 0,1,2

Exemplu:

```
for x in range(1,5):  
    print(x)
```

De data acesta, va incepe cu 1 și va continua cu 2,3,4

Exemplu:

```
for i in range(1,10,2):  
    print(i)
```

Va afișa doar numerele impare de la 1 la 9



Matrice bidimensională

Matrice bidimensională este o matrice în cadrul unei matrice. În acest tip de matrice, poziția unui element de date este dată prin doi indici în loc de unul . Deci reprezintă un tabel cu rânduri și de coloane de date .

Exemplu:

```
T = [  
    [1,2,3],  
    [4,5,6],  
    [7,8,9]  
]  
print(T[0][0])
```

În exemplul din stanga vedem modul cum se definește o matrice bidimensională și cum putem afișa elementele .

| | Column 0 | Column 1 | Column 2 |
|-------|----------|----------|----------|
| Row 0 | x[0][0] | x[0][1] | x[0][2] |
| Row 1 | x[1][0] | x[1][1] | x[1][2] |
| Row 2 | x[2][0] | x[2][1] | x[2][2] |

Matrice bidimensională

```
main.py
1- T = [
2     [1,2,3],
3     [4,5,6],
4     [7,8,9]
5 ]
6 print("Afisarea primului element din matrice")
7 print(T[0][0])
8 print("")
9 print("Afisarea liniilor din matrice ca array-uri separate")
10- for i in T:
11     print(i)
12     print("")
13 print("Afisarea separat a fiecarui element din matrice")
14- for row in T:
15     for j in row:
16         print(j)
```

```
Shell
Afisarea primului element din matrice
1

Afisarea liniilor din matrice ca array-uri separate
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]

Afisarea separat a fiecarui element din matrice
1
2
3
4
5
6
7
8
9
```

Tuples

Tuplu este o colecție de valori separate prin virgulă și cuprinse între paranteze. Spre deosebire de liste, tuplurile sunt imuabile. Imuabilitatea poate fi considerată ca trăsătură de identificare a tuplurilor (nu putem modifica, adăuga sau elimina elemente după ce a fost creat)

Cum se definește?

```
mytuple = ("lime", "banana", "watermelon")
```

Cum se poate parcurge element cu element?

```
mytuple = ("lime", "banana", "watermelon")  
for i in mytuple :  
    print(i)
```

Seturi

În cazul seturilor, elementele setate sunt neordonate, neschimbabile și nu permit valori duplicate .

Neordonat înseamnă că articolele dintr-un set nu au o ordine definită .

Elementele setului sunt neschimbabile, ceea ce înseamnă că nu le putem modifica după ce setul a fost creat .

Seturile nu pot avea două articole cu aceeași valoare.

Exemplu de set-uri:

```
myset = {"lime", "banana", "watermelon"}
```

Dicționare

Dicționarele sunt folosite pentru a stoca valorile datelor în perechi **cheie:valoare** .

Un dicționar este o colecție care este ordonată, poate fi schimbată și nu permite duplicate .

Exemplu:

```
Mydictionary = {  
    "cetatenie": "roman",  
    "datanasterii": "1 Oct 1996",  
    "inaltime": 190  
}  
print(Mydictionary)
```

Dictionare

main.py



Run

```
1 Mydictionary = { "cetatenie": "roman", "datanasterii": "1 Oct 1996", "inaltime": 190 }
2 print(Mydictionary)
3 print("Cetatenia este: ",Mydictionary["cetatenie"])
4 print("Data nasterii este: ",Mydictionary["datanasterii"])
5 print("Inaltimea este: ",Mydictionary["inaltime"])
```

Shell

Clear

```
{'cetatenie': 'roman', 'datanasterii': '1 Oct 1996', 'inaltime': 190}
Cetatenia este:  roman
Data nasterii este:  1 Oct 1996
Inaltimea este:  190
```

Funcții

O funcție este pur și simplu o „porțiune” de cod pe care o puteți folosi din nou și din nou, în loc să o fie scrisă de mai multe ori. Funcțiile le permit programatorilor să descompună o problemă în bucăți mai mici, fiecare dintre ele îndeplinind o anumită sarcină.

Cum se definește o funcție în python?

```
def my_function():  
    print("Hello from a function")
```

Funcția type

type() returnează tipul de clasă al argumentului (obiectului) transmis ca parametru. Funcția **type()** este folosită în principal în scopuri de debugging .

```
main.py
1 mynumber=3
2 mystring="hello world"
3 mylist=[4,5,6,10]
4 mytuple=("abc",43,54)
5 mydictionary = {
6     "name": "john",
7     "code": 123,
8     "dept": "hr"}
9 boolean_value=True
10
11 print(type(mynumber))
12 print(type(mystring))
13 print(type(mylist))
14 print(type(mytuple))
15 print(type(mydictionary))
16 print(type(boolean_value))
```

Funcția type

Astfel, pentru fiecare variabilă putem vedea tipul ei .

```
Shell Clear  
<class 'int'>  
<class 'str'>  
<class 'list'>  
<class 'tuple'>  
<class 'dict'>  
<class 'bool'>  
>
```


Argumentele unei funcții

Informațiile pot fi transmise în funcții ca argumente. Argumentele sunt specificate după numele funcției, în interiorul parantezei. Se pot adăuga câte argumente se dorește separate cu o virgulă .

Exemplu de funcție:

```
def my_function(ume):  
    print("Numele meu este ", ume)
```

Cum apelam funcția?

```
my_function('Vlad')
```

Funcții

```
main.py [ ] [ ] Run  
1 def my_function(ume):  
2     print("Numele meu este ",ume)  
3  
4 def suma(x,y):  
5     return x+y  
6  
7 my_function('Andrei')  
8 print("Suma numerelor este:",suma(10,3))
```

```
Shell Clear  
Numele meu este  Andrei  
Suma numerelor este: 13
```

La unele funcții putem folosi return (fără să afișăm direct ceva) pentru a folosi rezultatul funcției în alte operații. Dacă nu folosești “return” într-o funcție, în python se va folosi implicit “return None” (se poate vedea cu comanda **print(my_function('orice'))**)

Sfârșit

