NOTIȚE CURS 7 :

FUNCȚII - modularizează codul din program

↳ predefinite : de conversie (int, str, bool)
matematice (min, max, sum)
colecții (list, dict, set)
+ https://docs.python.org/3/library/functions.html

1. definirea

```
def nume_functie(parametri formali):
    instr.
```

2. parametri formali

parametri simpli :

```
def suma(x, y):
    return x+y
```

→ suma(3,7)
suma(y=7, x=3)

putem da pt. x și y val. pt. care se poate utiliza „+",
astfel putem da și două șiruri de caractere

parametri cu valori implicite:

```
def suma(x=0, y=0):
    return x+y
```

→ suma() = 0 (x=0, y=0)
suma(7) = 7 (x=7, y=0)
suma(y=19) = 19 (x=0, y=19)
suma(7,10) = 17 (x=7, y=10)

! contează ordinea
suma(x=0, y) pt. suma(7) y nu are val.
suma(x, y=0) pt. suma(7) x=7 și y=0 și fct. merge

parametri variabili : ↳ sub forma unui tuplu

```
def suma(*numere):
    sum=0
    for x in numere:
        sum+=x
    return sum
```

→ S=suma() = 0
S=suma(1,2) = 3
S=suma(...)

```
def suma(*numere, minim):
    sum=0
    for x in numere:
        if x>=minim:
            sum+=x
    return y
```

→ trebuie precizat când și care e minimul

S=suma(5,7, minim=6)
= 7

! putem returna mai multe valori

```
def suma_prod(x,y):
    return x+y, x*y
```

! transmitere prin referință la obiect

→ def funcție(t):
    t=100

```
x=7
functie(x)
print(x) → 7
```

x → [7]
copie x → [100]
↓
care se duce în funcție, dar x-ul „propriu-zis" nu se modifică

→ def functie(t):
    t.append(100)

```
x=[1,2,3]
functie(x)
print(x) → [1,2,3,100]
```

cum lista este mutabilă, val-ei se va schimba și în afara funcției

x → [1,2,3]
copie x → [1,2,3,100]

! dar dacă era t+=[100] atunci nu se modifica, de ce? pt. că programul ar fi creat o listă nouă cu o referință nouă unde ela memorată [1,2,3,100] și x nu s-ar fi modificat

Aminv: imutabil → nu se va modifica în afara funcției
mutabil → da

3. variabile: → locale
→ globale

```
def afisare()
    print(x)
        → 100
x=100
afisare()
```

```
def afisare()
    x=200
    print(x)
        → 200
x=100
afisare()
```

```
def afisare():
    global x
    print(x) → 100
    x=200      200
x=100
afisare()
print(x) → modifică val. lui x
```

4. funcții ca parametru

import math

def suma_generica(n, fk): ← putem transmite orice funcție predefinită sau definită de noi
    s=0
    for i in range(1, n+1):
        s = s + fk(i)
    return s

def fk_1(i):
    return 1/i

def fk_2(i):
    return i**2

n=10
s = suma_generica(n, fk_1) ↓ sau o altă fct.

5. fct. anonime lambda:

suma a două numere: lambda x, y: x+y
testarea parității: lambda x: x%2==0
testarea divizibilității: lambda x, y: False if y==0 else x%y==0
suma cifrelor unui nr: lambda x: sum([int(c) for c in str(x)])
nr. voc. dintr-un șir: lambda sir: len([c for c in sir if c in "aeiouAEIOU"])
                                  facem o listă cu voc. și după le nr. cu len

! le putem apela în momentul definirii lor

print((lambda x, y: x%y==0)(24,6)) → True

! le putem pune în antetul unei funcții atunci când o apelăm (spre ex. fk-ul de la 4)

6. fct. imbricate → fct. în interiorul altei funcții care nu pot fi apelate în cadrul programului ci doar în cadrul funcției

nonlocal → caută cea mai apr. var.

GENERATORI → un tip de funcție a cărei execuție nu se termină în mem. în care returnează o val., ceea ce permite returnarea mai multor val. într-o manieră secvențială

ex: fct. range

yield:

```
def generator_numere_pare(n):
    x = 0
    while x<=n:
        yield x
        x+=2
```

← val. furnizate de generator pot fi accesate printr-un for

next:

```
npare = generator_numere_pare(10)
x = next(npare, -1)
while x!=-1:
    print(x)
    x = next(npare, -1)
```

return → poate fi utilizat pt. a întrerupe un generator

g.close → închide generatorul (infinit sau nu)