CURS 7
Funcții (continuare)

Transmiterea unei funcții ca parametru al altei funcții (call-back)

a) funcții generice (exemplul cu sume)
b) operații de sortare

- **metoda sort() pentru liste**
- **funcția sorted() pentru orice colecție iterabilă**

```
T = [5, 2, 1, 2, 7, 10, 8, 8]

# L.sort()

L = sorted("testare")      # L = ['a', 'e', 'e', 'r', 's', 't', 't']

L = "".join(sorted("testare"))      # L = "aeerstt"

print("Lista sortata:", L)
```

```
T = (5, 2, 1, 2, 7, 10, 8, 8)

T = tuple(sorted(T, reverse=True))

print("Lista sortata:", T)
```

Un element al structurii de date —funcție→ **o cheie după care se va realiza sortarea = un tuplu**

Funcția sort() și metoda sorted() implementează sortări stabile, deci păstrează ordinea inițială a elementelor cu chei egale!

```
L = [5020, 27, 111, 71, 101, 2020, 17, 19, 40, 107, 12, 81, 18]

# 5020 -> (0, 502)
#   27 -> (1, 27)
#  111 -> (1, 111)
#   71 -> (1, 71)
#  101 -> (1, 101)
# 2020 -> (0, 2020)
```

```
# (0, 502) < (0, 2020)

def cheie(x):
    if x % 2 == 0:
        return x % 2, x
    else:
        return x % 2, -x

L = sorted(L, key=cheie)

print("Lista sortata:", L)
```

```
T = (502, 27, 111, 71, 101, 2020, 107, 12, 81, 18)

def cheie_1(x):
    s = 0
    while x != 0:
        s = s + x % 10
        x = x/ 10
    return s


T = sorted(T, key=cheie_1)

print("Lista sortata:", T)  # [101, 111, 12, 2020, 502, 71, 107, 27,
81, 18]
```

```
L = ["mere", "are", "multe", "Maria", "si", "pere"]

L.sort(key=len)

print("Lista sortata:", L)
```

```
# va sorta studentii in ordinea crescatoare a grupelor,
# iar in aceeasi grupa studentii vor fi sortati descrescator dupa
medii,
# iar in cazul unor medii egale studentii vor fi sortati alfabetic
def cheie_student(t):
    return t[1], -t[2], t[0]

L = [( "Popescu Ion",  131, 9.20),
     ( "Ionescu Ana",  133, 8.70),
     ( "Popa Marian",  131, 9.85),
     ( "David Maria",  132, 9.95),
     ("Gheorghe Ana",  131, 9.85),
     ("Corbu Florin",  133, 8.00)]

S = sorted(L, key=cheie_student)
```

```
print(*S, sep="\n")
```

Funcții imbricate (nested functions)

Într-o funcție putem defini alte funcții!

```
def combinari(n, k):

    def factorial(x):
        p = 1
        for i in range(1, x+1):
            p = p * i
        return p

    return factorial(n) // (factorial(k) * factorial(n-k))


print(combinari(5, 3))
```

O funcție imbricată are acces implicit la parametrii funcției în care este definită!

```
def putere(baza):

    def paux(exponent):
        return baza ** exponent

    return paux


putere10 = putere(10)

print(putere10(3))
```

```
T = (502, 27, 111, 71, 101, 2020, 107, 12, 81, 18)
x = 133

def cheie(x):
    def caux(elem):
        return x % 2 == elem % 2

    return caux


T = sorted(T, key=cheie(x))
```

```
print(T)
```

```
# vreau sa sortez lista in ordinea crescatoare a
# cmmdc-urilor elementelor listei cu un numar dat t

# cheie(element) = cmmdc(element, t)
# functia care furnizeaza cheia asociata unui element
# trebuie sa aiba un singur parametru, respectiv elementul curent

def cheie_cmmdc(t):

    def cmmdc(a, b):
        while a != b:
            if a > b:
                a = a - b
            else:
                b = b - a

        return a

    def cmmdc_aux(element):
        return cmmdc(t, element), element

    return cmmdc_aux


L = [20, 18, 46, 100, 90, 35, 8, 11]
valoare = 4

L = sorted(L, key=cheie_cmmdc(valoare))

print(L)
```

Variabile globale și locale

```
def afisare():
    print("x = ", x)


x = 100
afisare()       # x = 100
```

```
def afisare():
    x = 200
    print("x = ", x)
```

```
x = 100
afisare()           # x = 200
print("x = ", x)    # x = 100
```

```
def afisare():
    global x

    print("x = ", x)
    x = 200


x = 100
afisare()           # x = 100
print("x = ", x)    # x = 200
```

Generatoare

Generator = funcție în care folosim instrucțiunea `yield` în locul instrucțiunii `return`, ceea ce îi permite funcției respective să-și continue executarea și să furnizeze și alte valori, în mod repetat.

Exemplu: funcția range(...) este un generator!

```
def numere_pare(n):
    for x in range(0, n+1, 2):
        yield x


# for t in numere_pare(25):
#     print(t, end=" ")

pgen = numere_pare(30)
x = next(pgen)
while x <= 30:
    print(x, end=" ")
    x = next(pgen)
```

```
# generator infinit
def numere_pare():
    x = 0
    while True:
        yield x
        x = x + 2
```

```
# for t in numere_pare():
#     print(t, end=" ")

pgen = numere_pare()
x = next(pgen)
while x <= 100:
    print(x, end=" ")
    x = next(pgen)
```

Recursivitate

Recursivitate = proprietatea unei funcții de a se autoapela

În limbajul Python, adâncimea apelurilor recursive nu poate depăși valoarea 1000 în mod implicit!

Contextul de apel (stack frame) al unei funcții constă din:

- **numele funcției**
- **adresa de revenire din funcție**
- **parametrii funcției**
- **variabile locale**

Adâncimea apelurilor recursive = numărul maxim de stack frame-uri salvate pe stivă

```
# suma_cifre(n) = n%10 + suma_cifre(n/10)
def suma_cifre(n):
    if n < 10:
        return n
    return n % 10 + suma_cifre(n // 10)


print(suma_cifre(12345))
```

```
def suma_lista(L):
    if len(L) == 0:
        return 0
    return L[0] + suma_lista(L[1:])


print(suma_lista([12, 10, -13, -10, 11]))
```