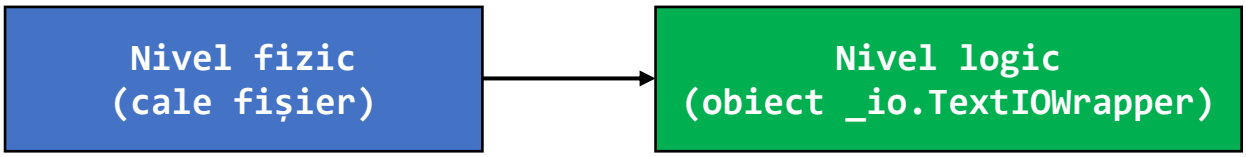
CURS 6
Fișiere text

Fișier text = o secvență de caractere organizată pe linii și stocată în memoria externă (SSD, HDD, DVD, CD etc.)

- Liniile sunt demarcate folosind caracterele:
  - `CR ('\r') + LF ('\n')` — Windows / MS-DOS
  - `LF ('\n')` — Unix / Linux / ≥ Mac OS X
  - `CR ('\r')` — ≤ Mac OS 9

- `CR = carriage return, LF = line feed`
- `ord('\n') = 10, ord('\r') = 13`

Deschiderea unui fișier text:




```
f = open("cale fișier", "mod de deschidere")
f = open("cale fișier")
```

Moduri de deschidere:

- pentru citire -> "r" (read) -> implicit -> dacă fișierul nu există, atunci se generează eroarea FileNotFoundError
- pentru scriere -> "w" (write) -> dacă fișierul există, atunci el va fi suprascris
- pentru scriere exclusivă -> "x" (exclusive write) -> dacă fișierul există, atunci acesta nu va fi suprascris automat, ci se va genera eroarea FileExistsError
- pentru adăugare la sfârșitul fișierului -> "a" (append) -> dacă fișierul există, atunci se va scrie la sfârșitul său, altfel se va crea un fișier nou

Excepții:

```
try:
    f = open("exemplu123456.txt")
    print("Fișier deschis cu succes!")
except FileNotFoundError:
    print("Fișier inexistent!")
```

```
try:
    f = open("exemplu123456.txt", "x")
    print("Fișier deschis cu succes!")
except FileExistsError:
    print("Fișierul există deja!")
```

Metode pentru citirea dintr-un fișier text:

- **`read()` -> tot conținutul fișierului într-un șir de caractere**
- **`readline()` -> tot conținutul liniei curente într-un șir de caractere**
- **`readlines()` -> toate liniile din fișier într-o listă de șiruri de caractere**

Exemple:

```
f = open("exemplu.txt")

# s = f.read()
# print("Metoda read():")
# print(s)

# print("Metoda readline():")
# s = f.readline()
# while s != "":
#     print(s, end="")
#     s = f.readline()

# print("Metoda readlines():")
# s = f.readlines()
# print(s)

print("Parcurgerea directă a liniilor:")
for linie in f:
    print(linie, end="")
```

Metode pentru scrierea într-un fișier text:

- **`write(șir de caractere)` -> scrie șirul respectiv în fișierul text**
- **`writelines(colecție de șiruri)` -> scrie șirurile respective în fișierul text, însă fără a adăuga linii noi între ele!!!**

```
f = open("numere.txt", "w")

n = 10
for x in range(n-1):
    f.write(str(x + 1) + " + ")
f.write(str(n))
f.close()

f = open("numere.txt", "r")
nr = [int(x) for x in f.read().split("+")]
s = sum(nr)
f.close()

f = open("numere.txt", "a")
f.write(" = " + str(s))
f.close()
```

```
f = open("exemplu.txt", "w")

cuvinte = ["Ana", "are", "mere", "si", "pere!"]

f.writelines("\n".join(cuvinte))

f.close()
```

Închiderea unui fișier text:

f.close()

```
with open("exemplu.txt", "w") as f:
    n = 1_000
    for x in range(n):
        f.write(str(x+1) + " ")

    # f.close()     #nu mai este necesara!!!
```

Funcții

Funcție = un set de instrucțiuni, grupate sub un nume unic, care efectuează o prelucrare specifică a unor date în momentul în care este apelată.

A function is a named section of a program that performs a specific task when it's called.

Definirea unei funcții:

```
def nume_funcție(parametrii)      ->   antet
    instrucțiuni                  ->   corp
```

Parametrii unei funcții:

a) parametri simpli

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


s = suma(3, 7)
print("s =", s)
```

b) parametri cu valori implicite

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


s = suma()
print("s =", s)

s = suma(7)
print("s =", s)

s = suma(y=19)
print("s =", s)

s = suma(7, 10)
print("s =", s)
```

c) număr variabil de parametri

```
def suma(*args):
    s = 0
    for x in args:
        s = s + x
    return s
```

```
s = suma()
print("s =", s)

s = suma(7)
print("s =", s)

s = suma(1, 2, 3, 4)
print("s =", s)

lista = [x for x in range(1, 11)]
s = suma(*lista)
print("s =", s)
```

Apelarea unei funcții:

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


s, p = suma_prod(3, 7)
print("s =", s, "\np =", p)

(s, p) = suma_prod(3, 7)
print("\ns =", s, "\np =", p)

t = suma_prod(3, 7)
print("\ns =", t[0], "\np =", t[1])
print("\nSuma si produsul:", *t)
```

Modalități de transmitere a parametrilor:

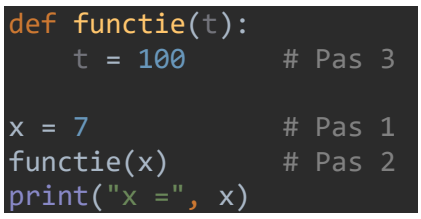
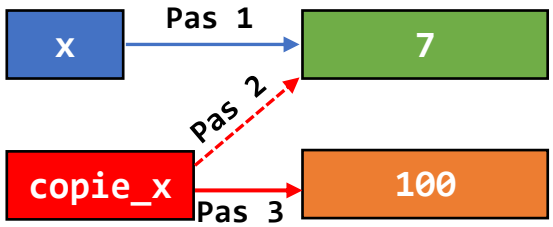
În limbajele C/C++ transmiterea unui parametru efectiv către o funcție se poate realiza în două moduri:

- *transmitere prin valoare*: se transmite o copie a valorii parametrului efectiv => modificările efectuate asupra parametrului respectiv în interiorul funcției NU se reflectă și în exteriorul său
- *transmitere prin adresă/referință*: se transmite adresa parametrului efectiv => modificările efectuate asupra parametrului respectiv în interiorul funcției se reflectă și în exteriorul său

În limbajul Python, orice parametru efectiv al unei funcții este o referință către un obiect (nu se transmite efectiv valoarea obiectului) și se transmite implicit prin valoare (deci se transmite o copie a referinței respective, deci modificarea referinței respective în interiorul funcției nu se va reflecta în exteriorul său)!!!

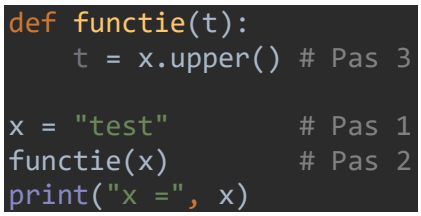
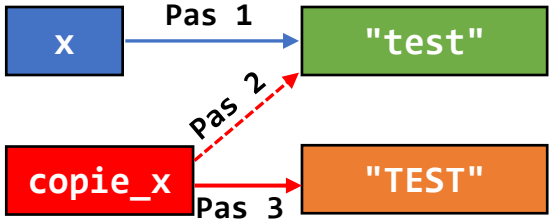
Mecanismul de transmitere a parametrilor către o funcție în limbajul Python se numește *transmitere prin referință la obiect* (*call by object reference*).

Exemplul 1:

Funcția	Executarea programului	Rezultat
		

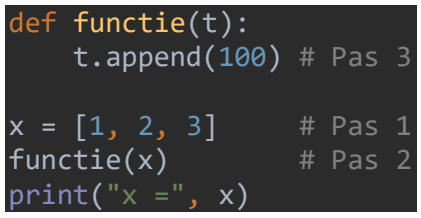
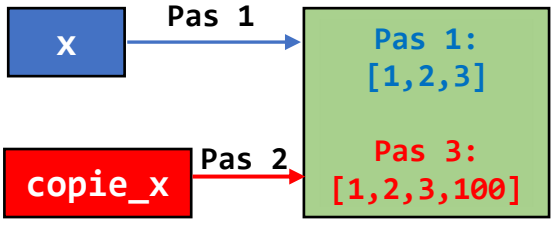
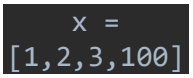
Explicație: După pasul 2, variabila x și `copie_x` vor conține aceeași referință (spre obiectul 7), dar după pasul 3 doar `copie_x` se va modifica (va conține referința obiectului 100), deoarece x este o referință transmisă prin valoare.

Exemplul 2:

Funcția	Executarea programului	Rezultat
		

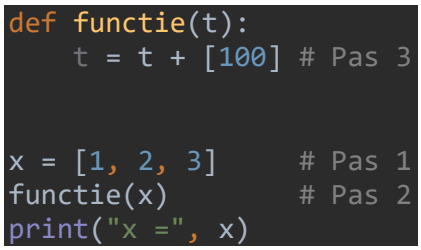
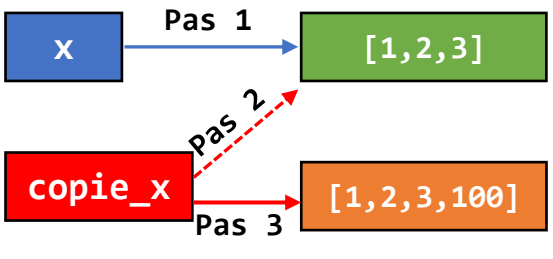
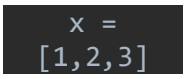
Explicație: vezi exemplul anterior!

Exemplul 3:

Funcția	Executarea programului	Rezultat
		

Explicație: După pasul 2, variabila x și `copie_x` vor conține aceeași referință, iar după pasul 3 conținutul listei se va modifica prin intermediul referinței din `copie_x` (însă fără a modifica referința listei!), deci variabila x va putea accesa lista modificată.

Exemplul 4:

Funcția	Executarea programului	Rezultat
		

Explicație: După pasul 2, variabila `x` și `copie_x` vor conține aceeași referință (spre lista `[1,2,3]`), dar la pasul 3 operatorul de concatenare (+) va crea o nouă listă `[1,2,3,100]`, deci după pasul 3 doar `copie_x` va conține referința noii liste. Practic, deși o listă este mutabilă, modalitatea prin care am adăugat elementul 100 este una specifică obiectelor imutabile!

În concluzie, mecanismul de *transmitere prin referință la obiect* a parametrilor efectivi către o funcție în limbajul Python utilizează transmiterea prin valoare a referințelor parametrilor efectivi (copii ale referințelor lor, deci modificarea acestora nu va fi vizibilă în exteriorul funcției) și acționează astfel:

- dacă obiectul asociat referinței este *imutabil*, atunci modificarea parametrului respectiv în interiorul funcției **nu** se va reflecta în exteriorul funcției deoarece conținutului obiectului asociat referinței nu poate fi modificat (din cauza imutabilității), iar crearea unei referințe noi nu se va reflecta în exteriorul funcției (din cauza transmiterii prin valoare a referinței);
- dacă obiectul asociat referinței este *mutabil*, atunci modificarea parametrului respectiv în interiorul funcției se va reflecta în exteriorul funcției deoarece nu se va modifica referința obiectului, ci doar conținutul său.

Atenție la exemplul 4 de mai sus, deoarece acolo nu se modifică direct conținutul obiectului mutabil de tip listă, ci se creează un nou obiect (tot o listă mutabilă) care conține noua listă și se atribuie referința sa copiei referinței parametrului!

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

Există mai multe situații în care este necesar să transmitem o funcție *f* ca parametru al unei funcții *g*, iar funcția *g* o va apela pe *f* atunci când este necesar. De exemplu, notificările utilizate în aplicațiile mobile utilizează mecanismul de call-back, respectiv o aplicație de tip server înregistrează faptul că un anumit utilizator a acceptat să primească notificări și în momentul în care apare un anumit eveniment pe server (de exemplu, când sunt depuși sau retrași bani dintr-un cont bancar), server-ul îi trimite utilizatorului respectiv o notificare.

Mecanismul de call-back mai este utilizat și în *programarea generică*, respectiv în scrierea unor funcții generice care realizează o prelucrare generică a unei funcții a cărei expresie nu este cunoscută (de exemplu, reprezentarea grafică a unei funcții, calculul unei integrale etc.).

În continuare, vom prezenta o funcție generică pentru calculul unor sume.

De exemplu, să considerăm următoarele 3 sume:

$$S_1 = 1 + \frac{1}{2} + \cdots + \frac{1}{n}$$
$$S_2 = 1^2 + 2^2 + \cdots + n^2$$
$$S_3 = e^1 + \mathrm{e}^2 + \cdots + \mathrm{e}^n$$

Putem observa cu ușurință faptul că toate cele 3 sume au următoarea formă generală:

$$S_k = \sum_{i=1}^{n} f_k(i)$$

unde prin $f_k(i)$ am notat termenul general de rang i al funcției f_k. Pentru cele 3 sume de mai sus, termenii respectivi sunt $f_1(i) = \frac{1}{i}$, $f_2(i) = i^2$ și $f_3(i) = e^i$.

Astfel, putem scrie o funcție generică pentru calculul unei astfel de sume, care va avea parametrii *n* și *tgen*, unde *tgen* va fi o funcție care implementează un termen general specific unei anumite sume:

```python
import math

def suma_generica(n, tgen):
    s = 0
    for i in range(1, n+1):
        s = s + tgen(i)
    return s

def tgen_1(i):
    return 1/i

def tgen_2(i):
    return i**2

n = 10

s = suma_generica(n, tgen_1)
print("Suma 1:", s)

s = suma_generica(n, tgen_2)
print("Suma 2:", s)

s = suma_generica(n, math.exp)
print("Suma 3:", s)
```