1. a) Scrieți o funcție care primește ca parametru un număr n și o funcție f și calculează suma $\sum_{i=1}^{n} f(i)$. Daca f nu este specificat se va calcula suma $\sum_{i=1}^{n} i$.

   Parametrul f va fi (obligatoriu) specificat prin nume

   Exemplu apel:

```
print(suma(10))
import math
print(suma(10,f=math.sqrt))
def radical(x):
    return x**0.5
print(suma(10,f=radical))
```

   b) Scrieți o funcție care primește ca parametri un număr variabil de numere $x_1,..,x_n$ și o funcție f și calculează suma $\sum_{i=1}^{n} f(x_i)$. Dacă f nu este specificat se va calcula suma $\sum_{i=1}^{n} x_i$

   Parametrul f va fi obligatoriu specificat prin nume. Folosiți funcția pentru a calcula suma pătratelor numerelor dintr-un fișier care conține numere naturale pe mai multe linii, pe fiecare linie numerele fiind separate prin spații.

   Exemplu apel:

```
print(suma(2,3,4,5))
ls = [3,1,7]
print(suma(*ls))
#print(suma(ls))
from math import sqrt
print(suma(4,9,16,f=sqrt))
```

2. Scrieți o funcție care primește un număr variabil de parametri și un filtru = o funcție booleana și returnează parametri care verifică filtrul:

   a) ca lista

   b) ca generator

   Dacă la apel nu este transmisă nicio funcție pentru filtru, atunci se vor returna parametrii primiți.

   Exemplu de apel:

```
def pozitiv(x):
     return x>0
```

   a) ca o noua lista

```
a = filtreaza(3,-1,6,8,-3,functie=pozitiv)
print(a)
a = filtreaza(3,-1,6,8,-3)
print(a)
a = filtreaza("ana","are","10","mere",functie=str.isalpha)
print(a)
```

b) un generator

```
a = filtreaza(3,-1,6,8,-3,functie=pozitiv)
print(a)
print(*a)
print(sum(a))
```

c) Modificați antetul de la a astfel încât funcția să primească parametru o listă pe care să o filtreze (nu un număr variabil de parametri). Exemplu de apel:

```
a = filtreaza([3,-1,6,8,-3],functie=pozitiv)
print(a)
```

3. a) Scrieți o funcție generică de căutare având următorul antet: **cautare(x, L, cmpValori)**
   Funcția trebuie să returneze indexul ultimei apariții a valorii x în lista L sau None dacă valoarea x nu se găsește în listă. Funcția comparator cmpValori se consideră că primește 2 parametri și returnează True dacă valorile primite ca parametrii sunt egale sau False în caz contrar.
   b) Se consideră lista de perechi (tupluri) l_pairs ale cărei elemente se citesc din fișierul perechi.txt (numerele dintr-o pereche sunt date pe o linie, separate prin -). Se citesc două numere x și y de la tastatură. Folosind un singur apel al funcției de la a) să se verifice dacă lista conține perechea (x,y) sau perechea (y,x) și, în caz afirmativ, afișează indexul ultimei apariții ale unei astfel de perechi.
   c) (Temă) Scrieți o funcție care să afișeze, folosind apeluri utile ale funcției cautare, mesajul DA în cazul în care o listă L formată din n numere întregi este modulo-palindrom sau mesajul NU în caz contrar. O listă este modulo-palindrom dacă prin parcurgerea modulelor elementelor sale de la dreapta la stânga sau de la stânga la dreapta se obține aceeași listă.
   De exemplu, lista L=[101,17,-101,13,5,-13,101,17,-101] este palindrom.

4. **Care este complexitatea următorului algoritm?**

a)

```
s = 0
i = n
while i>=1:
    j = 1
    while j<=n:
        s =s+1
        j = j + 1
    i = i/2
```

b)

```
s = 0
i = n
while i>=1:
    j = 1
    while j<=i:
        s =s+1
        j = j + 1
    i = i/2
```

Temă: 3-SUM Se dă un vector cu n elemente reale/întregi distincte. Să se afișeze toate tripletele de elemente din vector cu suma 0**.** Care este complexitatea algoritmului propus?

Metoda Greedy

Se consideră o mulțime de n activităţi care trebuie planificate pentru a folosi o aceeași resursă. Această resursă poate fi folosită de o singură activitate la un moment dat. Pentru fiecare activitate i se cunosc durata l_i şi termenul limită până la care se poate executa t_i (raportat la ora de început 0). Dorim să planificăm aceste activități astfel încât întârzierea fiecărei activități să fie cât mai mică. Mai exact, pentru o planificare a acestor activități astfel încât activitatea *i* este programată în intervalul de timp [s_i, f_i), definim întârzierea activității *i* ca fiind durata cu care a depășit termenul limită: $\boldsymbol{p_i = \max\{0, f_i - t_i\}}$.

Întârzierea planificării se definește ca fiind **maximul întârzierilor activităților**:

Exemplu. Pentru n = 3 şi $l_1 = 1$, $t_1 = 3$ / $l_2 = 2$, $t_2 = 2$ / $l_3 = 3$, $t_3 = 3$
o soluţie optimă se obţine dacă planificăm activităţile în ordinea 2, 3, 1; astfel:

- activitatea 2 în intervalul [0, 2) – întârziere 0
- activitatea 3 în intervalul [2, 5) – întârziere $5 - t_3 = 5 - 3 = 2$
- activitatea 1 în intervalul [5, 6) – întârziere $6 - t_1 = 6 - 3 = 3$

Întârzierea planificării este $\max\{0, 2, 3\}=3$ $\boldsymbol{P = \max\{p_1, p_2, ..., p_n\}}$

Să se determine o planificare a activităţilor date care să aibă întârzierea P minimă. Se vor afişa pentru fiecare activitate intervalul de desfăşurare şi întârzierea – **O(n log n)**

date.in	**date.out**
3 1 3 2 2 3 3	activitatea 2: intervalul 0 2 intarziere 0 activitatea 3: intervalul 2 5 intarziere 2 activitatea 1: intervalul 5 6 intarziere 3 Intarziere planificare 3
3 4 10 5 6 2 4	activitatea 3: intervalul 0 2 intarziere 0 activitatea 2: intervalul 2 7 intarziere 1 activitatea 1: intervalul 7 11 intarziere 1 Intarziere planificare 1