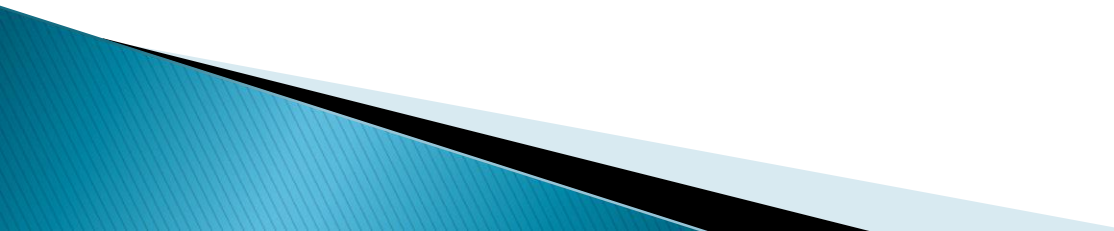


# Secvențe

# Secvențe

- **Colecție** – obiect care grupează mai multe obiecte într-o singură unitate, aceste obiecte fiind organizate în anumite forme.
  - ▶ Prin intermediul colecțiilor – acces la diferite **tipuri de date** cum ar fi liste, mulțimi matematice, tabele de dispersie, etc.
  - ▶ Utilitate: memorarea și manipularea datelor, cât și pentru transmiterea unor informații de la o metodă la alta.
- 

# Secvențe

- **Secvențe** = Colecție de elemente indexate (**de la 0**)
- Pot fi neomogene (elemente cu tipuri diferite)

```
ls = [1, "ab", 2.5]
```

# Secvențe

- ▶ Tipuri de secvențe – cele mai cunoscute:

❖ Liste – clasa `list`: `ls = [5, 7, 3]`

❖ Tupluri – clasa `tuple`: `t = (5, 7, 3)`

❖ Șiruri de caractere – clasa `str`: `s = "sirul"`

❖ `range`

# Secvențe

- ▶ După cum elementele secvenței pot fi modificate sau nu, secvențele se clasifică în două categorii:
  - **mutable:** lista `list`

```
ls = [3,5];    ls[0] = 1;    print(ls)
```

# Secvențe

- ▶ După cum elementele secvenței pot fi modificate sau nu, secvențele se clasifică în două categorii:

- **mutable:** lista `list`

```
ls = [3,5];    ls[0] = 1;    print(ls)
```

- **imutable:** tupluri, șiruri de caractere

```
s = "ab"
```

```
NU s[0] = 'a'
```

```
TypeError: 'str' object does not support item assignment
```

# Operații uzuale

# Operații uzuale

- ▶ Operații uzuale comune pentru secvențele din Python => se pot folosi și pentru liste, tuple, șiruri de caractere



# Operații uzuale

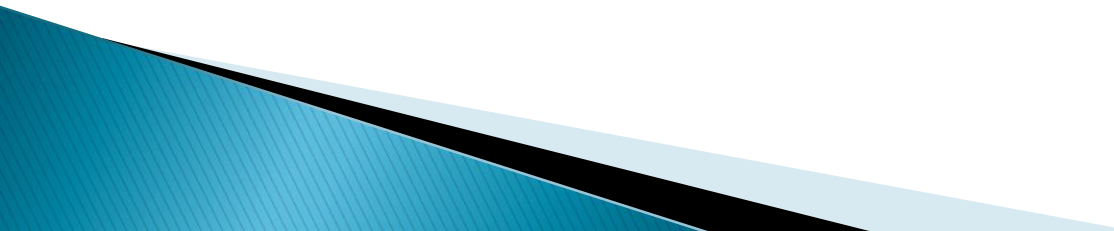
- ▶ Convenție – parametri opționali:

- `functie(x [, y])` – `y` este opțional

- Apel: `f(x)` , `f(x,y)`

- `functie(x [, y [, z]])` – `y` și `z` sunt opționali, dar `z` poate primi valoare doar dacă și pentru `y` se transmite o valoare

- Apel: `f(x)` , `f(x,y)` sau `f(x,y,z)`



# Operații uzuale

## ▶ Lungime `len(s)`

```
s = "sir"; print(len(s))
```

```
ls = [3,1,5]; print(len(ls))
```

# Operații uzuale

## ▶ Lungime `len(s)`

```
s = "sir"; print(len(s))
```

```
ls = [3,1,5]; print(len(ls))
```

## ▶ Minim `min(s)`

```
s = "sir"; print(min(s))
```

```
ls = [3, 1, 5]; print(min(ls))
```

```
ls = [3, "a"]; print(min(ls)) #TypeError
```

## ▶ Maxim `max(s)`

# Operații uzuale

- ▶ **Test de apartenență:** operatorii `in`, `not in`:

```
s = "un sir"
```

```
if "a" not in s:
```

```
    print("sirul nu contine litera a ")
```

# Operații uzuale

## ► Test de apartenență: operatorii `in`, `not in`:

```
s = "un sir"
```

```
if "a" not in s:
```

```
    print("sirul nu contine litera a ")
```

```
if "si" in s: #pentru siruri putem verifica si subsecvente
```

```
    print("sirul contine secventa si")
```



# Operații uzuale

## ► Test de apartenență: operatorii `in`, `not in`:

```
s = "un sir"
```

```
if "a" not in s:
```

```
    print("sirul nu contine litera a ")
```

```
if "si" in s: #pentru siruri putem verifica si subsecvente
```

```
    print("sirul contine secventa si")
```

```
ls = [3,1,4,0,8]
```

```
if 0 in ls:
```

```
    print("lista are elemente nule")
```

# Operații uzuale

## ▶ Accesare elemente:

$s[i]$  = elementul de pe poziția/indexul  $i$   
(numerotare de la 0)

# Operații uzuale

## ► Accesare elemente:

$s[i]$  = elementul de pe poziția/indexul  $i$   
(numerotare de la 0)

Indexul  $i$  **poate fi negativ**, și atunci se consideră relativ la sfârșitul secvenței ( $\text{len}(s) + i$ )

0	1	2	3	4	5	6	7	8	9	10
P	r	o	g	r	a	m	a	r	e	a
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

$s[2], s[-9] \Rightarrow o$



# Operații uzuale

## ▶ Accesare elemente:

$s[i]$  = elementul de pe poziția/indexul  $i$   
(numerotare de la 0) ,

- **IndexError** – dacă  $i$  nu este valid

# Operații uzuale

## ▶ Accesare subsecvențe – feliere (slice):

$s[i:j]$  = subsecvența formată cu elementele  
dintre pozițiile  $i$  și  $j$ , **exclusiv  $j$**   
(cu indicii  $i, i+1, i+2, \dots, j-1$ )

- Indicii pot fi și negativi + pot lipsi

# Operații uzuale

## ▶ Accesare subsecvențe – feliere (slice):

$s[i:j]$  = subsecvența formată cu elementele dintre pozițiile  $i$  și  $j$ , exclusiv  $j$  (cu indicii  $i, i+1, i+2, \dots, j-1$ )

- Indicii pot fi și negativi + pot lipsi
- Dacă indicele  $i$  lipsește – considerat 0
- Dacă indicele  $j$  lipsește – considerat  $\text{len}(s)$
- Dacă depășește  $\text{len}(s)$  – considerat  $\text{len}(s)$

# Operații uzuale

## ▶ Accesare subsecvențe:

- $s[:p] \Rightarrow$
- $s[-p:] \Rightarrow$

# Operații uzuale

## ▶ Accesare subsecvențe:

- $s[:p] \Rightarrow$  prefixul de lungime  $p$  a lui  $s$
- $s[-p:] \Rightarrow$  sufixul de lungime  $p$  a lui  $s$

# Operații uzuale

## ▶ Accesare subsecvențe:

- $s[:]$   $\Rightarrow$  ?

# Operații uzuale

## ▶ Accesare subsecvențe:

- `s[:]`  $\Rightarrow$  toată secvența;
  - în cazul listelor întoarce o copie  
(pentru ca sunt mutabile, deci `s[:] == s` este **True**, dar `s[:] is s` este **False**)

# Operații uzuale

## ▶ Accesare subsecvențe:

- `s[::-1] => ??`



# Operații uzuale

## ▶ Accesare subsecvențe:

- $s[::-1] \Rightarrow$  secvența  $s$  inversată

# Operații uzuale

0	1	2	3	4	5	6	7	8	9	10
P	r	o	g	r	a	m	a	r	e	a
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

`s = "Programarea"`

<code>print(s[3:7])</code>	
<code>print(s[:-4])</code>	
<code>print(s[:3])</code>	
<code>print(s[5:])</code>	
<code>print(s[-8:])</code>	
<code>print(s[5:2])</code>	
<code>print(s[-5:-2])</code>	
<code>print(s[12:13])</code>	

# Operații uzuale

0	1	2	3	4	5	6	7	8	9	10
P	r	o	g	r	a	m	a	r	e	a
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

`s = "Programarea"`

<code>print(s[3:7])</code>	gram
<code>print(s[:-4])</code>	
<code>print(s[:3])</code>	
<code>print(s[5:])</code>	
<code>print(s[-8:])</code>	
<code>print(s[5:2])</code>	
<code>print(s[-5:-2])</code>	
<code>print(s[12:13])</code>	

# Operații uzuale

0	1	2	3	4	5	6	7	8	9	10
P	r	o	g	r	a	m	a	r	e	a
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

`s = "Programarea"`

<code>print(s[3:7])</code>	gram
<code>print(s[:-4])</code>	Program
<code>print(s[:3])</code>	
<code>print(s[5:])</code>	
<code>print(s[-8:])</code>	
<code>print(s[5:2])</code>	
<code>print(s[-5:-2])</code>	
<code>print(s[12:13])</code>	

# Operații uzuale

0	1	2	3	4	5	6	7	8	9	10
P	r	o	g	r	a	m	a	r	e	a
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

`s = "Programarea"`

<code>print(s[3:7])</code>	gram
<code>print(s[:-4])</code>	Program
<code>print(s[:3])</code>	Pro
<code>print(s[5:])</code>	
<code>print(s[-8:])</code>	
<code>print(s[5:2])</code>	
<code>print(s[-5:-2])</code>	
<code>print(s[12:13])</code>	

# Operații uzuale

0	1	2	3	4	5	6	7	8	9	10
P	r	o	g	r	a	m	a	r	e	a
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

`s = "Programarea"`

<code>print(s[3:7])</code>	gram
<code>print(s[:-4])</code>	Program
<code>print(s[:3])</code>	Pro
<code>print(s[5:])</code>	amarea
<code>print(s[-8:])</code>	gramarea
<code>print(s[5:2])</code>	
<code>print(s[-5:-2])</code>	
<code>print(s[12:13])</code>	

# Operații uzuale

0	1	2	3	4	5	6	7	8	9	10
P	r	o	g	r	a	m	a	r	e	a
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

`s = "Programarea"`

<code>print(s[3:7])</code>	gram
<code>print(s[:-4])</code>	Program
<code>print(s[:3])</code>	Pro
<code>print(s[5:])</code>	amarea
<code>print(s[-8:])</code>	gramarea
<code>print(s[5:2])</code>	
<code>print(s[-5:-2])</code>	mar
<code>print(s[12:13])</code>	

# Operații uzuale

## Temă

0	1	2	3	4	5	6
3	1	11	4	7	9	5
-7	-6	-5	-4	-3	-2	-1

```
ls = [3, 1, 11, 4, 7, 9, 5]
```

```
print(ls[3:7])
```

```
print(ls[:3])
```

```
print(ls[5:])
```

```
print(ls[12:13])
```



# Operații uzuale

0	1	2	3	4	5	6
3	1	11	4	7	9	5
-7	-6	-5	-4	-3	-2	-1

```
ls = [3, 1, 11, 4, 7, 9, 5]
```

```
print(ls[:])
```

```
print(ls == ls[:])
```

```
print(ls is ls[:]) #print(ls[:],id(ls[:]))
```

# Operații uzuale

## ▶ Accesare elemente și subsecvențe:

$s[i:j:k]$  = secvența formată cu elementele dintre pozițiile  $i$  și  $j$ , exclusiv  $j$ , cu pasul  $k$  (cu indicii  $i, i+k, i+2k, \dots$ )

# Operații uzuale

## ▶ Accesare elemente și subsecvențe:

$s[i:j:k]$  = secvența formată cu elementele dintre pozițiile  $i$  și  $j$ , exclusiv  $j$ , cu pasul  $k$  (cu indicii  $i, i+k, i+2k, \dots$ )

- Dacă  $i$  sau  $j$  lipsesc – se consideră a fi valori “de final”, după caz

# Operații uzuale

## ► Accesare elemente și subsecvențe:

$s[i:j:k]$  = secvența formată cu elementele dintre pozițiile  $i$  și  $j$ , exclusiv  $j$ , cu pasul  $k$  (cu indicii  $i, i+k, i+2k, \dots$ )

- Dacă  $i$  sau  $j$  lipsesc – se consideră a fi valori “de final”, după caz
- Dacă  $k$  pozitiv și  $i$  și  $j$  depășesc  $\text{len}(s)$  – considerați  $\text{len}(s)$
- Dacă  $k$  negativ și  $i$  și  $j$  depășesc  $\text{len}(s) - 1$  – considerați  $\text{len}(s) - 1$

# Operații uzuale

0	1	2	3	4	5	6	7	8	9	10
P	r	o	g	r	a	m	a	r	e	a
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
s = "Programarea"
```

<code>print(s[1:6:2])</code>	rga
<code>print(s[6:1:-2])</code>	
<code>print(s[1::2])</code> #index impar	
<code>print(s[6::-2])</code>	
<code>print(s[12:7:-1])</code>	
<code>print(s[:3:-1])</code>	
<code>print(s[::-1])</code>	
<code>print(s[-2:-5:1])</code>	
<code>print(s[-2:-5:-1])</code>	

# Operații uzuale

0	1	2	3	4	5	6	7	8	9	10
P	r	o	g	r	a	m	a	r	e	a
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
s = "Programarea"
```

<code>print(s[1:6:2])</code>	rga
<code>print(s[6:1:-2])</code>	mro
<code>print(s[1::2])</code> #index impar	rgaae
<code>print(s[6::-2])</code>	
<code>print(s[12:7:-1])</code>	
<code>print(s[:3:-1])</code>	
<code>print(s[::-1])</code>	
<code>print(s[-2:-5:1])</code>	
<code>print(s[-2:-5:-1])</code>	

# Operații uzuale

0	1	2	3	4	5	6	7	8	9	10
P	r	o	g	r	a	m	a	r	e	a
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
s = "Programarea"
```

<code>print(s[1:6:2])</code>	rga
<code>print(s[6:1:-2])</code>	mro
<code>print(s[1::2])</code> #index impar	rgaae
<code>print(s[6::-2])</code>	mroP
<code>print(s[12:7:-1])</code>	
<code>print(s[:3:-1])</code>	
<code>print(s[::-1])</code>	
<code>print(s[-2:-5:1])</code>	
<code>print(s[-2:-5:-1])</code>	

# Operații uzuale

0	1	2	3	4	5	6	7	8	9	10
P	r	o	g	r	a	m	a	r	e	a
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
s = "Programarea"
```

<code>print(s[1:6:2])</code>	rga
<code>print(s[6:1:-2])</code>	mro
<code>print(s[1::2])</code> #index impar	rgaae
<code>print(s[6::-2])</code>	mroP
<code>print(s[12:7:-1])</code>	aer
<code>print(s[:3:-1])</code>	
<code>print(s[::-1])</code>	
<code>print(s[-2:-5:1])</code>	
<code>print(s[-2:-5:-1])</code>	



# Operații uzuale

0	1	2	3	4	5	6	7	8	9	10
P	r	o	g	r	a	m	a	r	e	a
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
s = "Programarea"
```

<code>print(s[1:6:2])</code>	rga
<code>print(s[6:1:-2])</code>	mro
<code>print(s[1::2])</code> #index impar	rgaae
<code>print(s[6::-2])</code>	mroP
<code>print(s[12:7:-1])</code>	aer
<code>print(s[:3:-1])</code>	aeramar
<code>print(s[::-1])</code>	aeramargorP
<code>print(s[-2:-5:1])</code>	
<code>print(s[-2:-5:-1])</code>	

# Operații uzuale

0	1	2	3	4	5	6	7	8	9	10
P	r	o	g	r	a	m	a	r	e	a
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
s = "Programarea"
```

<code>print(s[1:6:2])</code>	rga
<code>print(s[6:1:-2])</code>	mro
<code>print(s[1::2])</code> #index impar	rgaae
<code>print(s[6::-2])</code>	mroP
<code>print(s[12:7:-1])</code>	aer
<code>print(s[:3:-1])</code>	aeramar
<code>print(s[::-1])</code>	aeramargorP
<code>print(s[-2:-5:1])</code>	
<code>print(s[-2:-5:-1])</code>	era

# Operații uzuale

## Temă

0	1	2	3	4	5	6
3	1	11	4	7	9	5
-7	-6	-5	-4	-3	-2	-1

```
ls = [3, 1, 11, 4, 7, 9, 5]
```

```
print(ls[1:6:2])
```

```
print(ls[6:1:-2])
```

```
print(ls[1::2])
```

```
print(ls[6::-2])
```

```
print(ls[12:5:-1])
```

```
print(ls[:3:-1])
```

```
print(ls[::-1])
```

```
print(ls[-2:-5:1])
```

```
print(ls[-2:-5:-1])
```

# Operații uzuale

## ▶ Parcurgere element cu element

```
ls = [3, 1, 6, 10]
```

```
for x in ls: #stil Python  
    print(x)
```

```
for i in range(len(ls)): #stil C/C++  
    print(ls[i])
```

# Operații uzuale

## ► Interogări, căutări:

- test dacă o valoare există în s
  - operatorul `in`
- determinare frecvență (număr de apariții)
  - metoda `count`
- poziția (prima, toate) pe care apare o valoare
  - metoda `index`

# Operații uzuale

- ▶ Interogări, căutări:

- Ce se întâmplă dacă valoarea nu se găsește în s?

# Operații uzuale

## ► Interogări, căutări:

Ce se întâmplă dacă valoarea nu se găsește în s?

- unele funcții, spre exemplu `index => ValueError`
- Pentru liste `find => -1`

# Operații uzuale

## ► Interogări, căutări:

`s.index(x[,i[,j]])` => prima apariție a lui `x`  
în `s` (începând cu indicele `i`, până la  
indicele `j` **exclusiv**, dacă sunt specificați,  
ca la feliere)

`ValueError` dacă nu există

`s.count(x[,i[,j]])` = numărul de apariții



# Operații uzuale

```
s = 'acesta este un sir'

print("Sirul 'es' apare de ", s.count('es'), " ori")

print("Litera 'e' apare de ", s.count('e',0,10), "
ori printre primele 10 caractere ale sirului")
```

# Operații uzuale

```
s = "programarea"
```

```
x = s.index('a')
```

```
print("Prima pozitie pe care apare litera 'a' este ", x)
```

# Operații uzuale

```
s = "programarea"
```

```
x = s.index('a')
```

```
print("Prima pozitie pe care apare litera 'a' este ", x)
```

```
x = s.index('a', x+1)
```

```
print("A doua pozitie pe care apare litera 'a' este ", x)
```

```
x = s.index('ar')
```

```
print("Prima pozitie pe care apare sirul 'ar' este ", x)
```



# Operații uzuale

```
s = "programarea"
```

```
x = s.index('b')    #??
```

```
print("Prima pozitie pe care apare litera 'b' este ", x)
```

# Operații uzuale

```
try:
```

```
    x = s.index('b')
```

```
    print("Prima pozitie pe care apare litera b este ", x)
```

```
except ValueError:
```

```
    print("Litera b nu apare in sir")
```



# Operații uzuale

```
try:
```

```
    x = s.index('b')
```

```
    print("Prima pozitie pe care apare litera b este ", x)
```

```
except:
```

```
    pass
```



# Operații uzuale

## ► Concatenari:

`s + t`

```
s = "Programarea"
```

```
t = "Algoritmilor"
```

```
print("s=", s, "t=", t, id(s), id(t))
```

`s + t`

```
print("s=", s, "t=", t, id(s), id(t))
```

# Operații uzuale

## ► Concatenari:

`s + t`

```
s = "Programarea"
```

```
t = "Algoritmilor"
```

```
print("s=", s, "t=", t, id(s), id(t))
```

`s + t`

```
print("s=", s, "t=", t, id(s), id(t))
```

```
s = s + " " + t
```

```
print("s=", s, "t=", t, id(s), id(t))
```

`#obiect nou, nu adauga la vechiul s`



# Operații uzuale

```
s = [1,4,6]
```

```
t = [8,10]
```

```
print("s=", s, "t=", t, id(s),id(t))
```

```
s + t
```

```
print("s=", s, t, id(s),id(t))
```

```
s1 = s + t
```

```
print("s1=", s1,"s=",s, "t=", t, id(s1), id(s),id(t))
```

```
s1[0] = 12
```

```
print(s1,s) #s nu se modifica
```

```
s[0] = 14
```

```
print(s1,s) #s1 nu se modifica
```



# Operații uzuale

```
s = [[1,3],4,6]
```

```
t = [8,10]
```

```
s1 = s + t
```

```
print("s1=", s1,"s=",s, "t=", t, id(s[0]),id(s1[0]))
```

```
s1[0][0] = 14
```

```
print(s1, s)
```

```
s[0][0] = 15
```

```
print(s1, s)
```

# Operații uzuale

Dacă vrem să adăugăm elemente la finalul unei liste este indicat să folosim metoda **extend** (sau operatorul **+=**):

```
ls.extend(ls1)
```

NU concatenare de tipul **ls = ls + ls1**, aceasta fiind mai lentă

# Operații uzuale

```
s = "programare"
```

```
s = s[:2]+s[3:]
```

```
print(s)
```



# Operații uzuale

## ► Concatenari:

**s + t**

- La concatenarea de obiecte imutabile => **un nou obiect** => **complexitate mare** (mereu un nou obiect => copieri repetate)
- **Recomandare**: construim o listă și folosim **join** la **str** și **extend** pe liste (pentru tuple-uri – le transformăm în liste)

# Operații uzuale

## ► Concatenari:

$s * n$

$n * s \Rightarrow$  adunare  $s$  de  $n$  ori

$n \leq 0 \Rightarrow$  secvență vidă

- se multiplica referințele (nu valorile referite de elemente)

# Operații uzuale

```
n = 4
```

```
s = "ab"
```

```
print(s*n)
```

# Operații uzuale

```
n = 4
```

```
s = "ab"
```

```
print(s*n)
```

```
ls = [s]*n
```

```
print(ls, id(ls[0]),id(ls[1]))
```

```
s = "".join(ls)  #vom reveni
```

```
print(s)
```





# Operații uzuale

```
n = 4
```

```
s = "ab"
```

```
print(s*n)
```

```
ls = [s]*n
```

```
print(ls, id(ls[0]),id(ls[1]))
```

```
s = "".join(ls)  #vom reveni
```

```
print(s)
```



# Operații uzuale

```
ls = [1]*3  
print(ls, id(ls[0]), id(ls[1]))  
ls[1] += 1  
print(ls)
```

```
ls = [[1]]*3  
print(ls)  
ls[1].append(2)  
print(ls)
```

# Operații uzuale

- ▶ **Sortarea:** – returnează o **listă** cu elementele secvenței ordonate (!nu modifică secvența)

`sorted(iterable, key=None, reverse=False)`

# Operații uzuale

## ► Sortarea:

```
ls = [7, 2, 4, 3]
```

```
ls1 = sorted(ls, reverse = True) #descrescator
```

```
print("lista sortata descrescator: ", ls1)
```

```
print("lista initiala nu s-a modificat: ", ls)
```

```
s = "alfabet"
```

```
ls = sorted(s)
```

```
print(ls) #rezultatul este lista, desi s e sir
```

# Operații uzuale

- ▶ **Sortarea:** – returnează o **listă** cu elementele secvenței ordonate (!nu modifică secvența)

`sorted(iterable, key=None, reverse=False)`

- **key** – Vom detalia la liste

