

Șiruri de caractere – Metode specifice

Formatare

`template.format(<positional_arguments>,<keyword_arguments>)`

- ▶ `template` – Șir conține secvențe speciale cuprinse între `{}` care vor fi înlocuite cu parametri ai metodei `format` (numite **campuri de formatare**), de tipul

`{ [<camp>] [!<fct_conversie>] [:<format_spec>] }`

Șiruri de caractere – Metode specifice

Formatare

`template.format(<positional_arguments>,<keyword_arguments>)`

- ▶ `template` – Șir conține secvențe speciale cuprinse între `{}` care vor fi înlocuite cu parametri ai metodei `format` (numite **campuri de formatare**), de tipul

`{ [<camp>] [!<fct_conversie>] [:<format_spec>] }`

- ▶ Parametri
 - **poziționali** – se identifica prin poziție
 - **cu nume (numiți)** – se pot identifica și prin nume

Șiruri de caractere – Metode specifice

Formatare

`template.format(<positional_arguments>,<keyword_arguments>)`

- ▶ `template` – Șir conține secvențe speciale cuprinse între `{}` care vor fi înlocuite cu parametri ai metodei `format` (numite **campuri de formatare**), de tipul

`{ [<camp>] [!<fct_conversie>] [:<format_spec>] }`

- ▶ Parametri
 - **poziționali** – se identifica prin poziție
 - **cu nume (numiți)** – se pot identifica și prin nume
- ▶ Returnează sirul `template` formatat, înlocuind câmpurile de formatare cu valorile parametrilor (formatate conform cu specificațiilor din câmpuri)

Șiruri de caractere – Metode specifice

Variante de a specifica ce parametru pozițional se folosește într-un câmp de formatare:

- Specificăm numărul parametrului pozițional pe care îl folosim (numerotare de la 0)

```
s = "Nota la {0} = {1}".format("PA",10)
```

—

—

Șiruri de caractere – Metode specifice

Varianțe de a specifica ce parametru pozițional se folosește într-un câmp de formatare:

- Specificăm numărul parametrului pozițional pe care îl folosim (numerotare de la 0)

```
s = "Nota la {0} = {1}".format("PA",10)
```

- Nu specificăm nimic => parametrii poziționali sunt considerați în ordine (numerotare automată)

```
s = "Nota la {} = {}".format("PA",10)
```

Șiruri de caractere – Metode specifice

Variante de a specifica ce parametru pozițional se folosește într-un câmp de formatare:

- Specificăm numărul parametrului pozițional pe care îl folosim (numerotare de la 0)

```
s = "Nota la {0} = {1}".format("PA",10)
```

- Nu specificăm nimic => parametrii poziționali sunt considerați în ordine (numerotare automată)

```
s = "Nota la {} = {}".format("PA",10)
```

- Nu se pot combina cele două abordări

Șiruri de caractere – Metode specifice

```
x=3;y=4
```

```
print("x={},y={}".format(x,y))
```

```
print("x={0},y={1}".format(x,y))
```

Șiruri de caractere – Metode specifice

```
x=3;y=4
```

```
print("x={},y={}".format(x,y))
```

```
print("x={0},y={1}".format(x,y))
```

```
print("x={1},y={0},x+y={1}+{0}={2}".format(y,x,x+y))
```


Șiruri de caractere – Metode specifice

```
x=3;y=4
```

```
print("x={},y{}".format(x,y))
```

```
print("x={0},y={1}".format(x,y))
```

```
print("x={1},y={0},x+y={1}+{0}={2}".format(y,x,x+y))
```

```
print("x={0},y={1}, s={2}".format(x,y))
```

```
#eroare IndexError: Replacement index 2 out of range
```

```
#for positional args tuple
```



Șiruri de caractere – Metode specifice

Pentru a indica ce parametru numit (cu nume) se folosește într-un câmp de formatare putem folosi direct numele parametrului

```
x = 3
```

```
y = 4
```

```
print("x={p1},y={p2},s={suma}".format(suma=x+y,p1=x,p2=y))
```

Șiruri de caractere – Metode specifice

Se pot combina parametri poziționali cu cei numiți, dar cei numiți se dau la final

```
x = 3; y = 4
```

```
print("{p1}+{p2}={suma}, {p1}*{p2}={}")
```

```
format(x*y, suma=x+y, p1=x, p2=y))
```

Șiruri de caractere – Metode specifice

Pentru a include acoladă în șirul template fără a fi interpretat ca delimitator de câmp se dublează:

```
x = 3; y = 4; z = 5  
s = "Multimea {{{}}, {}, {}}".format(x, y, z)  
print(s)
```

Șiruri de caractere – Metode specifice

```
z = 1 + 3j
```

```
print('z = {0.real}+ {0.imag}i'.format(z))
```

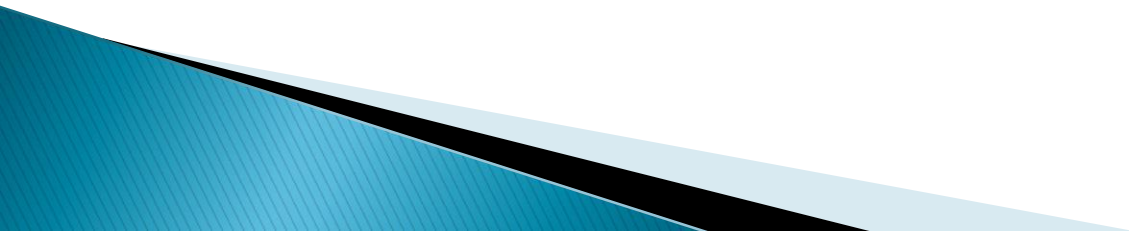
```
ls = [10,20,30]
```

```
print("primul si al doilea element din lista:  
{0[0]} si {0[1]}".format(ls))
```

Șiruri de caractere – Metode specifice

Specificarea formatului de afisare

Lungimea ocupată la afișare, precizie, aliniere, baza în care se afișează, formatul (notație cu exponent etc)



Șiruri de caractere – Metode specifice

{[<camp>][!<fct_conversie>][:<format_spec>]}

<fct_conversie>

- ▶ !s – convertește cu str()
- ▶ !r – convertește cu repr()
- ▶ !a – convertește cu ascii()

Șiruri de caractere – Metode specifice

```
s = "Programarea\talgoritmilor Ă"  
print('{!s}'.format(s))  
print('{!r}'.format(s))  
print('{!a}'.format(s))
```


Șiruri de caractere – Metode specifice

`<format_spec>` – poate include (printre altele)

`[0] [<dimensiune>] [.<precizie>] [<tip>]`

`<tip>`

- **d**: întreg zecimal
- **b,o x,X**: întreg în baza 2,8 respectiv 16 cu litere mici/mari
- **s** șir de caractere
- **f**: număr real în virgulă mobilă (afișat implicit 6 cifre)
... etc

Șiruri de caractere – Metode specifice

```
z = 1.1 + 2.2
```

```
print('{}'.format(z))
```

```
print('{:f}'.format(z))
```

```
print('{:.2f}'.format(z))
```

Șiruri de caractere – Metode specifice

```
z = 12
```

```
print('{}'.format(z))
```

```
print('{:8}'.format(z))
```

```
print('{:8b}'.format(z))
```

```
print('{:08b}'.format(z))
```



Șiruri de caractere – Metode specifice

Formatare – Interpolarea șirurilor: f-stringuri

– începând cu Python 3.6

- șir <template> precedat de f sau F
- în câmpurile de formatare putem folosi direct nume de **variabile**, chiar și expresii

Șiruri de caractere – Metode specifice

```
nume = 'Ionescu'
```

```
prenume = 'Ion'
```

```
varsta = 20
```

```
s = f"{nume} {prenume}: {varsta} ani"
```

```
print(s)
```



```
s = "{} {}: {} ani".format(nume, prenume, varsta)
```

```
print(s)
```

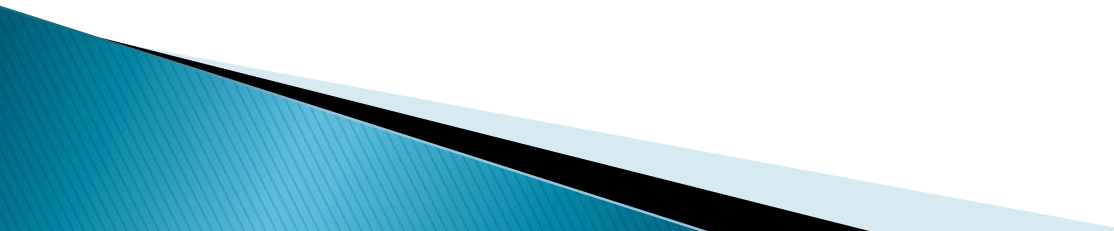
Șiruri de caractere – Metode specifice

```
x=3;y=4
```

```
print(f" {x} ")
```

```
print(f" {x}+{y}={x+y} ,  {x}*{y}={x*y} ")
```

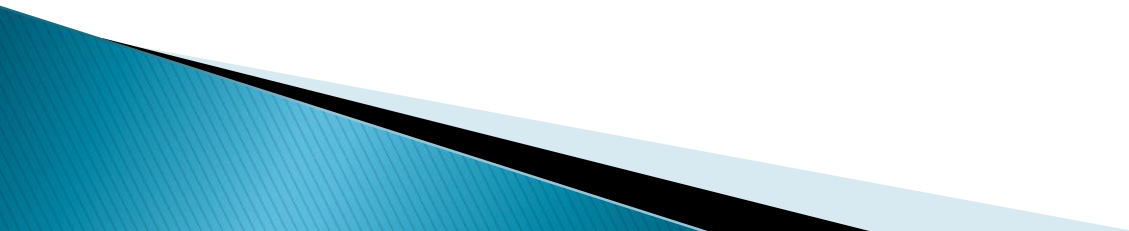
```
print(f' {x:08b} ')
```



Șiruri de caractere – Metode specifice

Formatare

<https://realpython.com/python-formatted-output/>



Șiruri de caractere – Metode specifice

Formatare cu operatorul %

- similar cu limbajul C (funcția printf)
- ▶ old- style – nu se mai recomandă folosirea ei

`<template> % (<values>)`

Șiruri de caractere – Metode specifice

```
s = "Nota la %s = %d" % ("PA",10)
```

```
print(s)
```

```
x = 3.1415
```

```
print("%d %.2f" % (x,x))
```

Liste

Liste

- Clasa `list`
- Mutabile
- Elementele pot avea tipuri diferite (de preferat nu)

Liste

Creare

- ▶ `ls = []` #lista vida
- ▶ `ls = [7, 5, 13]` #specificam elementele, intre []

Liste

Creare

- ▶ `ls = []` #lista vida
- ▶ `ls = [7, 5, 13]` #specificam elementele, intre []
- ▶ `list([iterabil])`
 - `ls = list()`
 - `ls = list("abc") => ??`

Liste

Creare

- ▶ Liste imbricate

```
ls = [ [1,3], 4, [5,6,7]]
```

```
print(len(ls))
```

```
ls[0][1] = 11
```

```
print(ls)
```

Liste

Creare

- ▶ comprehensiune: `[expresie for x in iterable]`

```
ls = [x*x for x in range(1,10)]  
print(ls)
```

Liste

Creare

- ▶ comprehensiune: `[expresie for x in iterable]`

```
ls = [x*x for x in range(1,10)]
```

```
print(ls)
```

```
n = 10
```

```
ls = [0 for i in range(n)] #ls = [0]*n
```

```
print(ls)
```



Liste

Creare

- ▶ comprehensiune: `[expresie for x in iterable]`

```
s = input()
```

```
ls = [int(x) for x in s.split()]
```

```
print(ls)
```



Liste

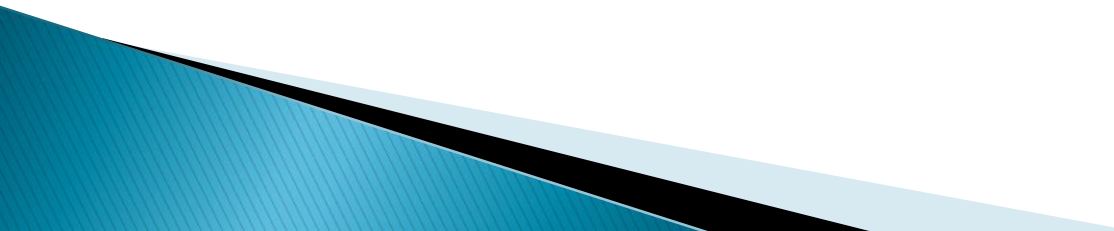
- ▶ Comprehensiune - varianta conditionată:

`[expresie for x in iterable if conditie]`

```
ls = [1,-2,3,4,-5,6]
```

```
ls_poz = [x for x in ls if x>0]
```

```
print(ls_poz)
```



Liste

- Comprehensiune - varianta conditionată:

[expresie for x in iterable if conditie]

```
ls = [1,-2,3,4,-5,6]
```

```
ls_poz = [x for x in ls if x>0]
```

```
print(ls_poz)
```

```
l1 = [2,4,6,8]
```

```
l2 = [1,2,3,4,5]
```

```
intersectie = [x for x in l1 if x in l2]
```

```
print(intersectie)
```

Liste

- ▶ Comprehensiune:

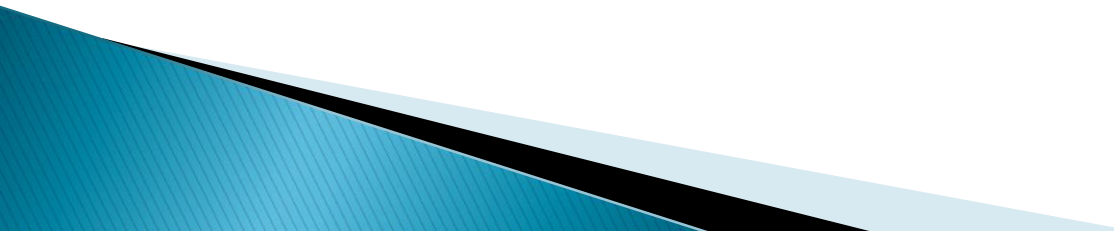
Pentru înlocuire, nu filtrare :

[expresie1 if conditie else expresie2 for x in iterable]

```
ls = [1,-2, 3, 4,-5,6]
```

```
ls_0 = [x if x>0 else 0 for x in ls]
```

```
print(ls_0)
```



Liste

► Comprehensiune

```
ls = [(x, y) for x in range(1,4) for y in range(1,4)
      if x != y]
print(ls)
```

Liste

Operatori

- ▶ `in, not in`
- ▶ Concatenare `+, *`
- ▶ Comparare: `<, <=, >=, >, >=, ==, !=`

Liste

```
ls1 = [1, 4, 2]
```

```
ls2 = [4, 1, 2]
```

```
print(ls1 == ls2)
```

Liste

```
ls1 = [1, 4, 2]
```

```
ls2 = [4, 1, 2]
```

```
print(ls1 == ls2)
```

```
ls1 = [1, 4, 2]
```

```
ls2 = [1, 4, 2]
```

```
print(ls1 == ls2)
```

```
print(ls1 is ls2)
```



Liste

```
ls1 = [1, 40, 2]
```

```
ls2 = [3, 7]
```

```
print(ls1 > ls2)
```

```
ls1 = [1, "b"]
```

```
ls2 = [1, "a", 2]
```

```
print(ls1 > ls2)
```



Liste

Operații prin care **se modifică** elementele listei

- ▶ `ls[i] = x`
- ▶ `ls[i:j] = iterabil` \Rightarrow subsecvența cu elem $i, \dots, j-1$
este înlocuită cu elementele lui `iterabil`
- ▶ `ls[i:j:k] = iterabil` (de aceeași lungime)

Liste

Operații prin care **se modifică** elementele listei

- ▶ `ls[i] = x`
- ▶ `ls[i:j] = iterabil` \Rightarrow subsecvența cu elem $i, \dots, j-1$ este înlocuită cu elementele lui `iterabil`
- ▶ `ls[i:j:k] = iterabil` (de aceeași lungime)
- ▶ `del ls[i:j]` `#ls[i:j]=[]`
- ▶ `del ls[i:j:k]`

Liste

```
ls = [1,10,20,30,60]
```

```
ls[2:4] = [12,14,16,18]
```

```
print(ls)
```

```
ls[1:1] = [5,6,7]
```

```
print(ls)
```

```
ls[1:6:2] = [0,0,0]
```

```
print(ls)
```

```
ls[1:4] = []
```

```
print(ls)
```



Liste

```
ls = [1,10,20,30]
```

```
del ls[-1]
```

```
print(ls)
```

```
del ls[0:1]
```

```
print(ls)
```

```
del ls[:]
```

```
print(ls)
```

```
del ls
```



Liste

Operații prin care **se modifică** elementele listei

- ▶ `ls.append(element)` – adaugă un **element** la sfârșit

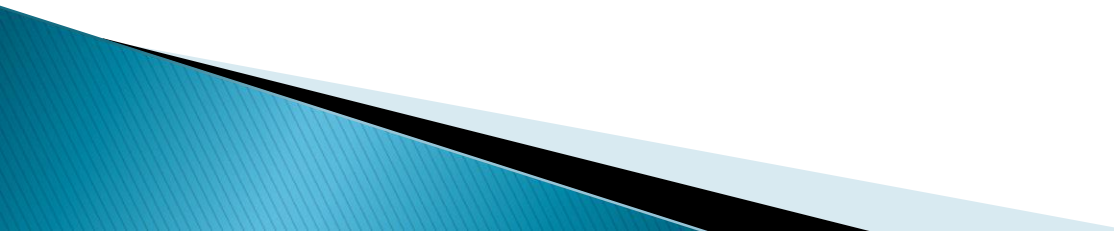
`ls[len(s):] = element`

- ▶ `ls.extend(iterabil)` – adaugă o **colecție** la sfârșit

`ls[len(s):] = iterabil`

Liste

```
ls = []  
  
ls.append(1)  
  
print(ls)  
  
ls.extend([2,3])  
  
print(ls)  
  
ls.append([2,3])  
  
print(ls)  
  
ls = ls+[5] # nerecomandat  
  
print(ls)
```



Liste

```
ls = []
```

```
ls.append("ab")
```

```
print(ls)
```

```
ls.extend("ab")
```

```
print(ls)
```

```
ls = ls+["abc"]
```

```
print(ls)
```

```
ls+="abc"
```

```
print(ls)
```



Liste

```
#citire element cu element
```

```
n = int(input())
```

```
ls = []
```

```
for i in range(n):
```

```
    ls.append(int(input()))
```

Liste

#citire element cu element

```
n = int(input())
```

```
ls = []
```

```
for i in range(n):
```

```
    ls.append(int(input()))
```

#sau - cu initializare

```
ls = [0 for i in range(n)]
```

```
for i in range(n):
```

```
    ls[i] = int(input())
```

Liste

```
n = 10
```

```
ls = []
```

```
for i in range(n):
```

```
    ls.append(i*i)
```

```
ls1 = []
```

```
for i in range(n):
```

```
    ls1 = ls1 + [i*i] #nu
```

```
ls2=[i*i for i in range(n)] #like
```

Liste

```
import time

n=10**5

start_time = time.time()

ls=[]

for i in range(n):

    ls.append(i*i)

print( time.time() - start_time)


start_time = time.time()

ls1=[]

for i in range(n):

    ls1 = ls1+[i*i]

print( time.time() - start_time)


start_time = time.time()

ls2=[i*i for i in range(n)]

print( time.time() - start_time)
```

Liste

Căutare – metode comune pentru secvențe

- `index`
- `count`

Liste

Operații prin care se modifică elementele listei

- `ls.insert(i,x)` \Leftrightarrow inserează `x` pe poziția `i`
`ls[i:i] = x`
- `ls.pop([i])` – elimină elementul de pe **poziția** `i`
sau ultimul dacă `i` nu este dat (`i` default este `-1`)
- `ls.remove(x)` – elimină prima apariție a lui `x`
(`ValueError` dacă nu există)
- `ls.clear()`

Liste

```
ls = [10,20,30,40,50]
```

```
ls.insert(2,60)
```

```
print(ls)
```

```
ls.remove(10) #prima aparitie
```

```
print(ls)
```

Liste

```
ls = [10,20,30,40,50]
```

```
poz = 2
```

```
print(f"se elimina elementul {ls.pop(poz)} de pe  
pozitia {poz} => {ls}")
```

```
print(f"se elimina ultimul element {ls.pop()} =>  
{ls}")
```

```
ls.clear()
```

```
print(ls)
```


Liste

Copiere

- `ls.copy()` – copiere superficială

Liste

```
ls1 = [1,2]
```

```
ls2 = ls1
```

```
ls1[0] = 13
```

```
print(ls1,ls2)
```

Liste

```
ls1 = [1,2]
```

```
ls2 = ls1
```

```
ls1[0] = 13
```

```
print(ls1,ls2) #si ls2 s-a modificat
```

```
ls1 = [1,2]
```

```
ls2 = ls1.copy() #ls2 = ls1[:]
```

```
ls1[0] = 13
```

```
print(ls1,ls2)
```

Liste

```
ls1 = [[1,2], [3,4]]
```

```
ls2 = ls1.copy()
```

```
ls1[0][0] = 13
```

```
print(f"{ls1}    {ls2}")
```

Liste

```
ls1 = [[1,2], [3,4]]
```

```
ls2 = ls1.copy()
```

```
ls1[0][0] = 13
```

```
print(f"{ls1}    {ls2}") #se modifica ls2[0]
```

```
import copy
```

```
ls1 = [[1,2], [3,4]]
```

```
ls2 = copy.deepcopy(ls1)
```

```
ls1[0][0] = 13
```

```
print(f"{ls1}    {ls2}")
```

Liste

```
#matrice
```

```
m=2; n=3
```

```
a = [[0]*n]*m
```

```
a[0][0] = 3
```

```
print(a)
```

```
a = [[0 for i in range(n)] for j in range(m)]
```

```
a[0][0] = 3
```

```
print(a)
```

```
a = [[0 for i in range(n)]]*m
```

```
a[0][0] = 3
```

```
print(a)
```

```
a = [[0]*n for i in range(m)]
```

```
a[0][0] = 3
```

```
print(a)
```

