

Instrucțiuni

#cel mai mic număr prim din intervalul [a,b]

```
a = int(input("a="))
b = int(input("b="))
for x in range(a,b+1):
    for d in range(2,x//2+1):
        if x%d == 0:
            break
    else:
        print(x)
        break
else:
    print("Nu exista numar prim in interval")
```

Instrucțiuni

#cel mai mic număr prim din intervalul [a,b]

```
a = int(input("a="))
```

```
b = int(input("b="))
```

```
for x in range(a,b+1):
```

```
    for d in range(2, int(x**0.5)+1):
```

```
        if x%d == 0:
```

```
            break
```

```
    else:
```

```
        print(x)
```

```
        break
```

```
else:
```

```
    print("Nu exista numar prim in interval")
```

Instrucțiuni

#cel mai mic număr prim din intervalul [a,b]

```
a = int(input("a="))
```

```
b = int(input("b="))
```

```
for x in range(a,b+1):
```

```
    for d in range(2, int(x**0.5)+1): #???sqrt?
```

```
        if x%d == 0:
```

```
            break
```

```
    else:
```

```
        print(x)
```

```
        break
```

```
else:
```

```
    print("Nu exista numar prim in interval")
```

Funcții predefinite

- **Modulul builtins**

<https://docs.python.org/3/library/functions.html#built-in-funcs>

Funcții predefinite

- **Conversie**

– constructori `int()`, `float()`, `str()`

```
print(bin(23), hex(23)) #str
```

Funcții predefinite

- **Matematică**

```
print(abs(-5))
```

```
print(min(5,2))
```

```
x = 3.0
```

```
print(x.is_integer())
```

```
print(round(x + 0.7))
```

```
import math
```

```
print(math.sqrt(4))
```



Funcții predefinite

- **Matematică**

```
import math
```

```
print(math.sqrt(4))
```

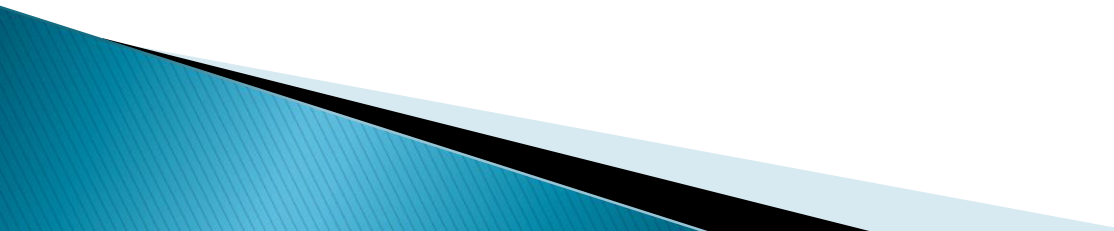
```
from math import sqrt
```

```
print(sqrt(5))
```



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:

❖ 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)
```

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

```
s = "ab";    s[0]='a'
```

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

Operații uzuale

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 și subsecvențe:

$s[i]$ = 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 și subsecvențe:

$s[i:j]$ = 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 elemente și subsecvențe:

$s[i:j]$ = 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 `len(s)`

Operații uzuale

▶ Accesare elemente și subsecvențe:

$s[i:j]$ = 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 indicii depășesc $\text{len}(s)$ – considerați $\text{len}(s)$

Operații uzuale

▶ Accesare elemente și subsecvențe:

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

- $s[:]$ => toată secvența;
 în cazul listelor întoarce o copie
 (pentru ca sunt mutabile)

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

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"
```

```
print(s[:])
```

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

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

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[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

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

► 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)
`ValueError` dacă nu există

`s.count(x)` = numărul de apariții

Operații uzuale

```
s = 'Programarea'
```

```
print(s.count('a')) #merg si siruri
```

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

```
print(x)
```



Operații uzuale

```
s = 'Programarea'
```

```
print(s.count('a')) #merg si siruri
```

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

```
print(x)
```

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

```
print(x)
```



Operații uzuale

```
s = 'Programarea'
```

```
print(s.count('a')) #merg si siruri
```

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

```
print(x)
```

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

```
print(x)
```

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

```
print(x)
```



Operații uzuale

```
x = -1
```

```
try:
```

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

```
except ValueError:
```

```
    pass
```



Operații uzuale

```
ls = [6,8,10,2,8,5]
```

```
x = ls.index(8)
```

```
print(x)
```

```
try:
```

```
    x = ls.index(7)
```

```
except:
```

```
    x = -1
```

```
print(x)
```



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)
```



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

► Concatenari:

$s * n$

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

$n \leq 0 \Rightarrow$ secvență vidă

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

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)
```

```
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)`

- **Vom detalia la liste**

Șiruri de caractere



Șiruri de caractere

Clasa `str`

- Constante (literali) delimitate de:
 - `' ... '`
 - `" ... "`
 - `''' ... '''` sau `""" ... """` – se pot întinde pe mai multe linii

Șiruri de caractere

```
s = 'acesta este un sir'
```

```
t = "acesta este un alt sir"
```

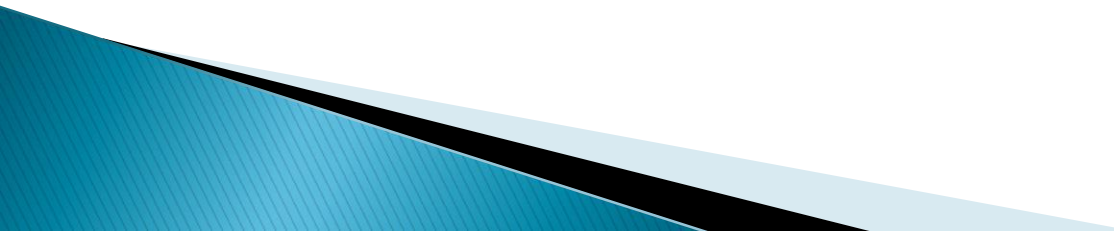
```
versuri = """A fost odata ca in povesti  
A fost ca niciodata"""
```

```
print(s)
```

```
print(t)
```

```
print(versuri)
```

```
s[0] = 'A' => TypeError: 'str' object does not  
support item assignment
```



Șiruri de caractere

Clasa `str`

- ▶ Pot fi create și folosind `str()` = constructor

```
x = str(3.1415)
```

- ▶ **Imutabil** – nu putem modifica **valoarea** după creare

Șiruri de caractere

- ▶ Nu există `char`, `s[0]` este tot de tip `str`

`s[0] == s[0:1] ?`

`s[0] is s[0:1] ?`

Șiruri de caractere

- Secvențe escape:

`\n` newline

`\t` tab

`\\`

`\'`

`\"`

https://docs.python.org/3/reference/lexical_analysis.html#string-and-bytes-literals

```
s = 'Programarea\talgoritmilor'
print(s)
```

Șiruri de caractere

```
s = "That's it"
```

```
print(s)
```



Şiruri de caractere

```
s = "That's it"
```

```
print(s)
```

s = 'That's it' ?!?!?!?!?

Șiruri de caractere

```
s = "That's it"
```

```
print(s)
```

```
s = 'That\'s it'
```

```
print(s)
```



Șiruri de caractere

```
s = "That's it"
```

```
print(s)
```

```
s = 'That\'s it'
```

```
print(s)
```

```
s = """I say "That's it" again"""
```

```
print(s)
```



Șiruri de caractere

Caractere Unicode

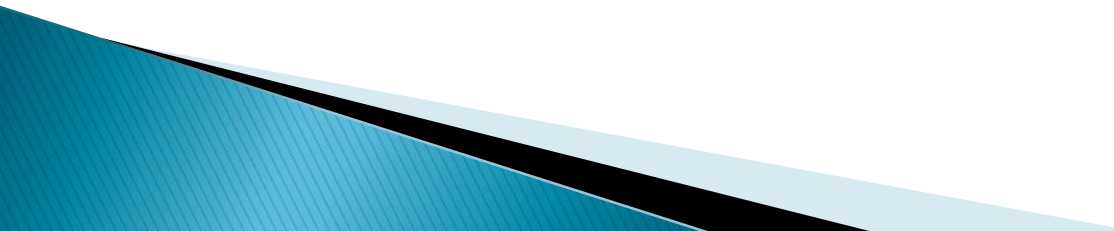
- ▶ Prefixam cu `\u` (cod hexa 16) sau `\U` (cod hexazecimal 32 biti) sau `\N{numele lor}`

```
s = "Aceasta este o pisic\u0103 \N{Cat}"  
print(s)
```

Șiruri de caractere

Metode uzuale

Cele comune pentru subsecvențe:

- Accesarea elementelor, feliere (slice)
 - Operatori
 - de concatenare `+`, `*n`
 - `in`, `not in`
 - Funcții uzuale: `len`, `min`, `max`
- 

Șiruri de caractere – Metode specifice

Căutare

Amintim

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

`s.count(sub)` = numărul de apariții

`s.rindex(sub[,i[,j]])` => ultima apariție

Șiruri de caractere – Metode specifice

Căutare

`s.find(x[,i[,j]])` => ca și `s.index`, dar
returnează -1 dacă nu găsește

`s.rfind`

Șiruri de caractere – Metode specifice

```
s = 'Programarea'
print(s.find("a"))
print(s.find("z")) #nu da eroare, ca s.index
print(s.rfind('a'))
```

Șiruri de caractere – Metode specifice


Căutare

`s.startswith`

`s.endswith`

```
if s.endswith('nt'):  
    print('Fazan!')
```

tuplu



```
if s.startswith(('a', 'e', 'i', 'o', 'u')):  
    print('incepe cu o vocala')
```

```
if s.startswith(tuple("aeiouAEIOU")):  
    print('incepe cu o vocala')
```


Șiruri de caractere – Metode specifice

Căutare+înlocuire

`s.replace(old, new[, count])` => returnează o copie a lui s în care toate aparițiile subsecvenței `old` sunt înlocuite cu `new`;

- dacă este dat și parametrul opțional `count`, atunci sunt înlocuite doar primele `count` apariții ale lui `old` (!!! nu se modifică `s`, este imutabil)

Șiruri de caractere – Metode specifice

```
t = s = 'Programarea algoritmilor'  
s.replace('a', 'aaa')  
print(s)
```

Șiruri de caractere – Metode specifice

```
t = s = 'Programarea algoritmilor'  
s.replace('a', 'aaa')  
print(s)  
s = s.replace('a', 'aaa')  
print(s)  
print(t)
```

Șiruri de caractere – Metode specifice

```
t = s = 'Programarea algoritmilor'
s.replace('a', 'aaa')
print(s)

s = s.replace('a', 'aaa')
print(s)

print(t)

s = s.replace('o', '', 1)
print(s)
```

Șiruri de caractere – Metode specifice

Transformare la nivel de caracter

- `s.lower()` – **returnează** șirul scris cu minuscule (!nu îl modifică)
- `s.upper()`
- `s.capitalize()`
- `s.strip([chars])` elimină caracterele din *chars* de la începutul și sfârșitul șirului;
implicit `chars=None` elimina caracterele albe
- `s.rstrip()` / `s.lstrip()`
- `s.center()` / `s.ljust(width, fillchar=' ')` / `rjust`

Șiruri de caractere – Metode specifice

```
s = '  Programarea algoritmilor!!**  '  
s.lower()  
print(s)
```

Șiruri de caractere – Metode specifice

```
s = '  Programarea algoritmilor!!**  '
s.lower()
print(s)
s = s.strip()
```

Șiruri de caractere – Metode specifice

```
s = '  Programarea algoritmilor!!**  '
s.lower()
print(s)
s = s.strip()
print(s.rstrip("!*"))
print(s)
```


Șiruri de caractere – Metode specifice

```
s = '  Programarea algoritmilor!!**  '
s.lower()
print(s)
s = s.strip()
print(s.rstrip("!*"))
print(s)
print('Programarea algoritmilor'.center(40))
print('Programarea algoritmilor'.center(40,"*"))
print('Programarea algoritmilor'.rjust(30,">"))
```

Șiruri de caractere – Metode specifice

Testare/clasificare la nivel de caracter

- `s.islower()` – returnează **True** dacă toate literele din șir sunt minuscule, **False** altfel
- `s.isupper()`
- `s.isdigit()`
- ...

Șiruri de caractere – Metode specifice

Parsare, divizare si unificare cu separatori

```
s.split(sep = None, maxsplit =-1)
```

- ▶ Împarte `s` în cuvinte folosind `sep` ca separator (delimiter) și returnează o lista acestor cuvinte.
- ▶ Dacă parametrul opțional `maxsplit` este specificat, sunt făcute cel mult `maxsplit` împărțiri (se obține o listă de maxim `maxsplit+1` cuvinte).

Șiruri de caractere – Metode specifice

Parsare, divizare si unificare cu separatori

```
s.split(sep = None, maxsplit =-1)
```

- ▶ Dacă nu este specificat sep – caracterele albe consecutive sunt considerate un separator
- ▶ între delimitatori consecutivi => cuvinte vide

Șiruri de caractere – Metode specifice

```
s = "acesta  este un sir"  
print(s.split())  
#['acesta', 'este', 'un', 'sir']
```

Șiruri de caractere – Metode specifice

```
s = "acesta  este un sir"
```

```
print(s.split())
```

```
['acesta', 'este', 'un', 'sir']
```

```
print(s.split(" "))
```

```
['acesta', '', '', 'este', 'un', 'sir']
```

Șiruri de caractere – Metode specifice

```
s = "acesta     este un sir"
```

```
print(s.split())
```

```
['acesta', 'este', 'un', 'sir']
```

```
print(s.split(" "))
```

```
['acesta', '', '', 'este', 'un', 'sir']
```

```
print(s.split(maxsplit=1))
```

```
['acesta', 'este un sir']
```



Șiruri de caractere – Metode specifice

```
s = input("numere pe o linie\n")  
ls = s.split()  
print(ls)  
s = 0  
for x in ls:  
    s = s + int(x)  
print(s)
```


Șiruri de caractere – Metode specifice

Parsare, divizare si unificare cu separatori

`s.join(iterable)`

- ▶ Returnează șirul obținut prin concatenarea șirurilor din parametrul `iterable`, folosind șirul `s` ca separator între șirurile concatenate

Șiruri de caractere – Metode specifice

```
ls = ["2", "3", "5"]
```

```
s = "*".join(ls)
```

```
print(s)
```

Șiruri de caractere – Metode specifice

```
ls = ["2", "3", "5"]
```

```
s = "*".join(ls)
```

```
print(s)
```

```
ls = ["ab"]*4
```

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

```
s = "".join(ls)
```

```
print(s)
```



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)
- Nu specificăm nimic => parametrii poziționali sunt considerați in ordine (numerotare automata)
- Nu se pot combina cele două abordări

Șiruri de caractere – Metode specifice

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

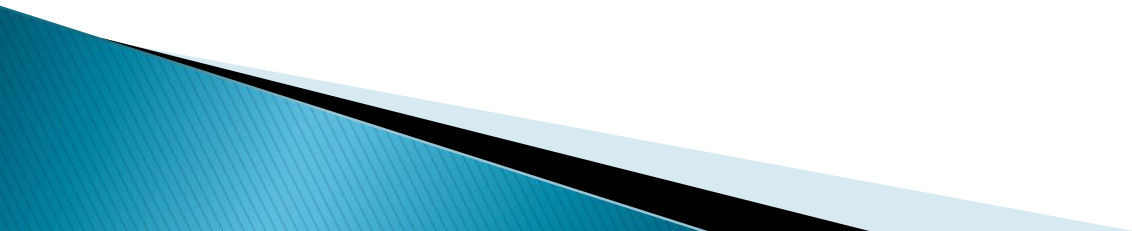

Șiruri de caractere – Metode specifice

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

```
print(s)
```

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

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



Șiruri de caractere – Metode specifice

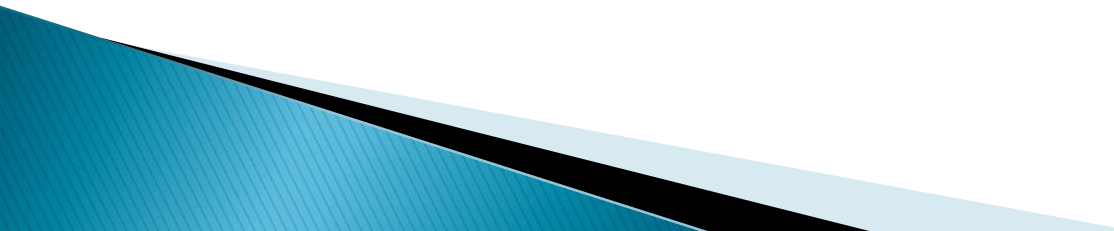
```
s = "Nota la {} = {}".format("PA",10)
```

```
print(s)
```

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

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

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



Șiruri de caractere – Metode specifice

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

```
print(s)
```

```
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

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

```
print(s)
```

```
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 acolada 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)
```

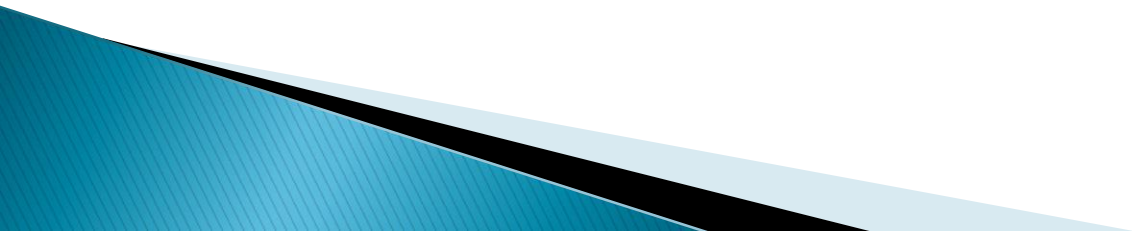

Șiruri de caractere – Metode specifice

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

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

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

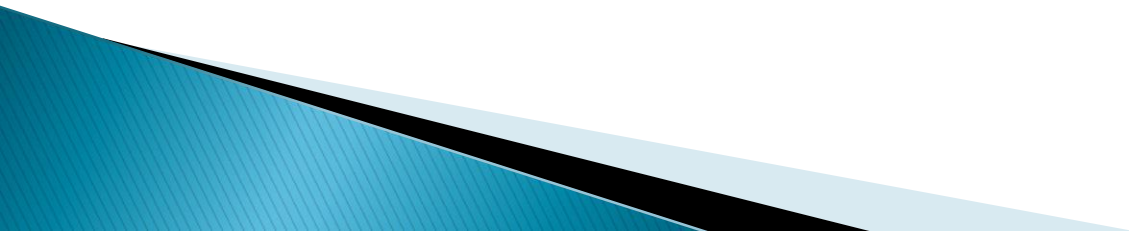
```
print(f' {x:08b} ')
```



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))
```

