



# Programare 1

Siruri de caractere  
Cursul 5

# Despre ce am discutat în cursul precedent?

Procesul de dezvoltare software

Instrucțiuni repetitive

- For, while, break, continue

Tipuri de date predefinite

- List, tupluri, dictionare, set-uri

# Despre ce o să discutăm astăzi?

șiruri de caractere

formatare șirurilor de caractere

expresii regulate

# Şirurile de caractere (Strings)

Sunt o secvență de caractere (sequence)

Se pot utiliza:

- 'un şir de caractere' , nu `un sir de caractere`
- "un alt şir de caractere"
- "şir cu apostrof ‘ ’"
- 'şir cu "ghilimele '
- """sir si cu "ghilimele şi apostrof ' """
- “dar nu ‘ ”

# Şirurile de caractere (Strings)

- Sunt o secvențe ORDONATE de caractere (sequence)

- permit indexare

- `s="hello"`

- `s[0]` - h
      - `s[1]` - e
      - `s[-1]` - o

- slicing (feliere)

- `s[::-1]` - olleh
      - `s[1:3]` - el

- functia `len(s)`

- `len(s)` - 5



# Proprietăți și metode pe siruri

---

- sunt imutabile

```
In [1]: s = 'mere'
s[0] = 'P'

-----
TypeError: 'str' object does not support item assignment
<ipython-input-1-cd9ff8bb2ee7> in <module>()
      1 s = 'mere'
----> 2 s[0] = 'P'

Traceback (most recent call last)
```

- soluție posibilă: folosind concatenare - se creează un nou sir!

```
In [4]: s = 'mere'
s = 'p'+s[1:]
print (s)

pere
```

# Proprietăți și metode pe șiruri

- `dir([obiect])`
  - `dir(s)` - returnează o lista cu tot ce se poate face (built-in) cu obiectul respectiv
  - pentru șiruri - 77!!!
- în interpreter: obiect. - și apăsăm Tab pentru a primi lista
- `help([obiect]):` documentație explicită
  - exemplu: `help(s.upper)`

```
In [6]: len(dir(s))
```

```
Out[6]: 77
```

# Metode utile pe siruri

---

- `upper()` - analog `lower()`
  - `s = "mere"`
  - `s.upper() - "MERE"`
  - `s`-ul ramane neschimbat, dacă vrem să îl modificăm pe `s`
    - `s = s.upper()`
- `split()`
  - returnează o lista cu elementele din sir separate după delimiter (default " ")

```
In [10]: s.upper
```

```
Out[10]: <function str.upper>
```

```
In [11]: s = "ananas mere pere zmeura"  
s.split()
```

```
Out[11]: ['ananas', 'mere', 'pere', 'zmeura']
```

```
In [12]: s = "ananassiheresiperesizmeura"  
s.split("si")
```

```
Out[12]: ['ananas', 'mere', 'pere', 'zmeura']
```

# Metode utile pe siruri

- **join(iterabil)**

- returnează un sir care este o concatenare a elementelor din iterabil, iar separatorul fiind sirul care inițiază metoda

```
In [21]: " ".join(["ananas", "mere", "pere"])
```

```
Out[21]: 'ananas mere pere'
```

```
In [22]: " si ".join(["ananas", "mere", "pere"])
```

```
Out[22]: 'ananas si mere si pere'
```

```
| In [26]: " ".join([str(i) for i in [1,2,3]])
```

```
Out[26]: '1 2 3'
```

```
| In [23]: " ".join([1,2,3])
```

```
-----
TypeError                                         Traceback (most recent call la
st)
<ipython-input-23-c45d9ba95faf> in <module>()
      1 " ".join([1,2,3])
-----
```

```
TypeError: sequence item 0: expected str instance, int found
```

# Exercițiu laborator - discuție

- E3. Scrieți un program care citește un număr și afișează un triunghi. De exemplu, pentru numărul 5:

```
*  
**  
***  
****
```

In [2]:

```
n = 5  
for i in range(n+1):  
    print("*"*i)
```

| In [6]:

```
n = 5  
for i in range(n+1):  
    s = ""  
    for j in range(i):  
        s = s + "*"  
    print(s)
```

| In [5]:

```
n = 5  
for i in range(n+1):  
    for j in range(i):  
        print ("*")
```

| In [8]:

```
n = 5  
for i in range(n+1):  
    for j in range(i):  
        print("*", end=' ')  
    print()
```

# Exercițiu laborator - discuție

- E3. Scrieți un program care citește un număr și afișează un triunghi. De exemplu, pentru numărul 5:

```
*  
**  
***  
**** In [2]: n = 5  
*****  
  
for i in range(n+1):  
    print("*"*i)
```

| In [6]:



```
n = 5  
for i in range(n+1):  
    s = ""  
  
    for j in range(i):  
        s = s + "*"  
    print(s)
```



| In [5]: n = 5  
for i in range(n+1):  
 for j in range(i):  
 print ("\*")



| In [8]:

```
n = 5  
for i in range(n+1):  
    for j in range(i):  
        print("*", end=' ')  
    print()
```



# Formatare cu print() - cum am făcut pana acum?

```
In [16]: s="ananas"  
        print(s)|  
        print(s+" mere")
```

```
ananas  
ananas mere
```

```
In [18]: s="ananas"  
        print(s + str(7))
```

```
ananas7
```

```
In [17]: s="ananas"  
        print(s + 7)
```

```
-----  
---  
TypeError                                         Traceback (most recent call la  
st)  
<ipython-input-17-468eac26d4ba> in <module>()  
      1 s="ananas"  
----> 2 print(s + 7)  
  
TypeError: must be str, not int
```

# Print() folosind .format()

- metoda pe siruri de caractere:
  - exemplu:
    - "un sir în care urmează ceva{} și încă ceva{}".format(ceva1, ceva2)

```
In [19]: print("Aici e un sir {}".format("INSERAT"))
```

Aici e un sir INSERAT

```
In [28]: print("sir cu string {} si numar {}".format("STRING",16))
```

sir cu string STRING si numar 16

# Print() folosind .format()

```
In [29]: v1 = 5  
        v2 = 6  
        print("putem folosi si variabile: {} , {}".format(v1,v2))  
  
putem folosi si variabile: 5 , 6
```

```
In [30]: v1 = 5  
        v2 = 6  
        print("putem folosi si variabile: {1} , {0} ".format(v1,v2))  
  
putem folosi si variabile: 6 , 5
```

```
In [31]: v1 = 5  
        v2 = 6  
        print("putem folosi si variabile: {1} , {1} ".format(v1,v2))  
  
putem folosi si variabile: 6 , 6
```



Print()  
folosind  
.format() -  
precizie  
zecimale

```
In [37]: f = 10/7  
print(f)
```

```
1.4285714285714286
```

```
In [42]: print("Nr {nume_var:1.3}".format(nume_var=f))
```

```
Nr 1.43
```

```
In [43]: print("Nr {nume_var:1.4}".format(nume_var=f))
```

```
Nr 1.429
```

```
In [38]: print("Nr {}".format(f))
```

```
Nr 1.4285714285714286
```

# Formatare cu `f` sau `r` - Extra Python Tip

- În loc de `.format()` se poate utiliza f-string (din Python 3.6)

```
In [45]: nume = "Ana"  
varsta = "20"  
print(f"Studenta {nume} are {varsta} ani")
```

Studenta Ana are 20 ani

- 'r' - (raw) se poate utiliza pentru a utiliza caractere speciale ("escape sequences")

```
| In [46]: print("ananas \n mere \n pere\n")
```

ananas  
mere  
pere

```
| In [47]: print(r"ananas \n mere \n pere\n")
```

ananas \n mere \n pere\n

# Exemple pe șiruri

Avem un text și ne interesează doar o anumită bucată -> cautam după un cuvânt cheie -> căutarea unui subșir în alt sir:

- parcurgere pe sir
  - for ...
- folosind **find()**
  - `cuv_cheie = "abc"`
  - `text = "akslidakjdaldjajababc sdfsdfsadasdadas abc sdfsdf"`
  - `p = text.find(cuv_cheie)` -> returnează prima apariție a lui `cuv_cheie` în text

## Exemple pe şiruri

Vrem sa căutam, validăm și extrage toate datele în formatul zz/l/l/aaaa dintr-un text

Avem o serie de e-mail-uri și vrem sa validăm dacă sunt corecte sau nu

Avem un text în care delimitatorii pot fi semne de punctuație diferite și spații

- exemplu (cu spațiu, virgulă și punct)
  - ananas, mere.struguri; pere zmeura,rodie

# Expresii regulate - Regex

o notație compactă pentru a reprezenta o colecție de siruri

se definesc folosind un mini-limbaj, independent de limbajul Python

se utilizează pentru:

- parsare
- căutare
- căutare și înlocuire
- împărțirea sirului în subșiruri
- validare

# Regex - caractere

- forma cea mai simplă - caractere simple ('literal characters'): **a,b,7**
- dacă nu se specifică niciun cuantificator, implicit: **potrivire pe o singura apariție**
  - exemplu: regex - **abc** -> un **a**, urmat de un **b**, urmat de un **c**
    - **aabc**
    - **aabcc**
    - **abc**
    - **aabbcc**
  - caractere speciale, trebuie puse ca și sevenți escape, precedate de \
    - **\.^\$?+\*{}[]()|**
  - potrivire pe mai multe caractere -> expresie de tip **clasa de caractere** (fără legătura cu clasele din OOP): o serie de caractere între [ ], potrivire pe oricare din caracterele dintre [ ]

# Regex - caractere

- potrivire pe mai multe caractere -> expresie de tip ***clasa de caractere*** (fara legatura cu clasele din OOP): o serie de caractere intre [ ], potrivire pe oricare din caracterele dintre [ ]
  - exemplu: regex - a[bc]d
    - Abd
    - Acd
    - ~~Abcd~~
    - ~~Acbd~~
    - Aabdddd
    - acdd

# Regex - caractere

- exemplu: regex - [0123456789] sau [0-9] -> potrivire pe o singură cifră
  - - înseamnă interval de valori
- exemplu: regex - [^0-9] -> potrivire orice caracter care nu este cifra
  - ^ - se utilizează pentru a nega expresia, dacă este utilizat pe prima poziție într-un regex de tip clasa de caractere

# Regex - caractere - prescurtari

Simbol	Semnificatie
.	potrivire orice caracter cu exceptia liniei noi
\d	potrivire orice cifra
\D	potrivire orice non-cifra
\s	potrivire orice spatiu ([\t\n\r\f\v])
\S	potrivire orice non spatiu
\w	potrivire un “cuvant” ([a-zA-Z0-9_])
\W	potrivire un “non-cuvant”

# Exemple

identificarea datelor dintr-un text

- format: zz/ll/aaaa: Astazi, **15/10/2018**  
*discutam despre regex, în data de  
9/10/2018 am discutam despre funcții*
  - [0-9][0-9]/[0-9][0-9]/[0-9][0-9][0-9][0-9]
  - \d{2}\d{2}/\d{4}

validare adresa de email:

- format: orice litera, cifra si \_, urmat de @
  - regex: \w@[a-z]
  - sir: marian@google.com
  - rezultat: n@g

# Regex - cuantificatori

forma: {min,max} - de cate ori trebuie sa se potriveasca expresia la care se aplica cuantificatorul

- min - numărul minim de apariții
- max - numărul maxim de apariții
  - exemplu: regex - a{1,1}a{1,1}
  - exemplu: regex - a{2,2}
    - aab
    - baab
    - baa
    - abc

forma pe scurt: {nr}, unde nr este si minim si maxim

implicit, dacă nu se specifica niciun cuantificator: o singura data

## $\{0,1\} \rightarrow ?$

- exemplu: regex - ab{0,1}c
- exemplu: regex - ab?c

## $\{1,n\} \rightarrow +$

- cel puțin o apariție

## $\{0,n\} \rightarrow *$

- orice număr de apariții (incluzând niciunul)

# Regex - cuantificatori

# Exemple

- 1. Identificarea datelor dintr-un text
  - format: zz/ll/aaaa: *Astazi, 15/10/2018 discutam despre regex, in data de 9/10/2018 am discutam despre functii*
    - [0-9][0-9]/[0-9][0-9]/[0-9][0-9][0-9][0-9]  
-> [0-9]{1,2}/[0-9]{1,2}/[0-9]{4}
    - \d\d/\d\d/\d\d\d\d ->  
\d{1,2}/\d{1,2}/\d{4}

# Exemplu

---

- 2. Validare adresa de email:
  - format: orice litera, cifra si \_, urmat de @
    - regex: \w@[a-z] -> \w+@[a-z]+
    - sir: maria[@google.com](mailto:@google.com)
    - rezultat: maria@google
  - format: orice litera, cifra si \_, urmat de @, urmat de . , apoi orice litera mica
    - regex: \w@[a-z] -> \w+@[a-z]+\.[a-z]+
    - sir: maria[@google.com](mailto:@google.com)
    - rezultat: maria@google.com

# Modul pentru expresii regulate

- modulul **re**
- pasi:
  - compilarea expresiei:
    - **pattern** =  
**re.compile**("expresie regulata")
  - utilizarea pattern-ului
    - exemplu:
      - rez =  
**pattern.findall**("sirul in care se cauta")
    - .

```
>>> text = "Astazi, 25/10/2021 discutam despre regex, \
    in data de 18/10/2021 am discutat despre dictionare"
>>> pattern = re.compile(r'\d{1,2}/\d{1,2}/\d{4}')
>>> rez = pattern.findall(text)
>>> print(rez)
['25/10/2021', '18/10/2021']
```

# Dizivarea sirului dupa o expresie

- text = ananas, mere.struguri; pere zmeura,rodie

```
In [15]: text = "ananas, mere.struguri; pere          zmeura,rodie"
         r = text.split(",.l ")
         print(r)

['ananas, mere.struguri; pere          zmeura,rodie']
```

```
| In [28]: import re
           text = "ananas, mere.struguri; pere          zmeura,rodie"
           r = re.split(r"[.,;\s]\s*",text)
           print(r)

['ananas', 'mere', 'struguri', 'pere', 'zmeura', 'rodie']
```