

Lucrarea 2. Funcții. Structuri de date

A. Obiectivele lucrării

1. Familiarizarea cu sintaxa și modul de declararea a funcțiilor în **Python**
2. Prezentarea generală a structurilor de date disponibile în limbajul de bază
3. Familiarizarea cu instrucțiunile necesare pentru a lucra cu fișiere
4. Prezentarea formatelor disponibile pentru salvarea datelor

B. Funcții

Funcțiile sunt o metodă de împărțire a codului în secvențe ce au anumite funcționalități și care pot fi apoi refolosite, salvând astfel timp la implementarea aplicațiilor. În **Python**, definirea unei funcții se face folosind cuvântul cheie `def`, urmat de numele funcției, parametrii de intrare ai acesteia, separați de virgulă și plasați între paranteze rotunde și simbolul „`:`” (două puncte). După aceasta urmează blocul de cod aferent funcției (ca și la structurile de control, acesta trebuie indentat pentru a sugera interpretorului faptul că este cod interior funcției).

Exemplu de definire și apelare a unei funcții care să afișeze pe ecran mesajul „Informatică Aplicată II”:

main.py
<pre>def afiseaza_titlu(): # continutul functiei print("Informatica Aplicata II") # apelarea functiei afiseaza_titlu()</pre>
rezultat
Informatica Aplicata II

rezultat

```
2  
3  
4  
5  
6  
7
```

În plus, funcțiile pot, folosind cuvântul cheie `return`, să furnizeze o valoare la sfârșitul execuției acestora. Exemplu:

main.py

```
def farenheit_to_celsius(degF):  
    degC = (degF - 32) * 5 / 9  
    return degC  
  
print(farenheit_to_celsius(50))  
  
val_degC = farenheit_to_celsius(66.3)  
print(val_degC)
```

rezultat

```
10.0  
19.055555555555557
```

C. Structuri de date

Python oferă în limbajul de bază o serie de structuri de date: liste, tupluri, seturi și dicționare [1].

a) Liste

Lista este un vector unidimensional ce poate conține orice număr de elemente de orice tip. Valoarea acesteia se poate inițializa enumerând valorile elementelor acesteia, separate de virgulă între un set de paranteze drepte. Ulterior, un element al listei se poate accesa scriind numele variabilei urmat de indexul elementului între paranteze pătrate. În *Python* numărarea index-urilor începe de la 0, dar se pot folosi și indecsi negativi, caz în care elementele vor fi numărate de la sfârșitul listei spre început (e.g. indexul -1 reprezintă ultimul element al listei, indexul -2 reprezintă penultimul element, etc) [1]. Exemplu:

main.py

```
lista = ["triunghi", "patrat", "cerc"]  
  
print(lista[1]) # elementul cu indexul 1 ("patrat")
```

Lucrarea 2. Funcții. Structuri de date

```

print(lista[2]) # elementul cu indexul 2 ("cerc")
print(lista[-1]) # ultimul element ("cerc")
lista[0] = "hexagon" # modificare primul element
print(lista)

print(lista[0:2]) # elementele cu indecsii 0 și 1

```

rezultat

```

patrat
cerc
cerc
['hexagon', 'patrat', 'cerc']
['hexagon', 'patrat']

```

Conținutul unei liste poate fi alterat folosind o serie de metode ceea ce înseamnă că listele sunt **MUTABILE**. Modul în care se poate utiliza o metodă de alterare: se scrie numele listei, urmat de simbolul punct și de numele metodei (`lista.append`, unde `lista` reprezintă numele listei). Cele mai uzuale metode pentru liste se regăsesc în tabelul următor [1]:

Tabelul 1 - Metode pentru alterarea listelor

Metoda	Descriere
<code>append(x)</code>	adăugă valoarea <code>x</code> la sfârșitul listei
<code>insert(i, x)</code>	inserează un element cu valoarea <code>x</code> la poziția <code>i</code>
<code>extend(l)</code>	adăugă elementele din lista <code>l</code> la sfârșitul listei
<code>pop(i)</code>	șterge elementul de pe poziția <code>i</code> și returnează valoarea acestuia
<code>clear()</code>	șterge toate elementele listei
<code>sort()</code>	sorțează lista

Exemple de utilizare a unor metode de alterare a conținutului unei liste:

main.py

```

lista = [1, 2, 3]
print(lista)
# [1, 2, 3]

lista.append(4)
print(lista)
# [1, 2, 3, 4]

lista.insert(1, 5)
print(lista)
# [1, 5, 2, 3, 4]

lista.pop(2)
print(lista)
# [1, 5, 3, 4]

lista.clear()
print(lista)
# []

```

Pentru a determina numărul de elemente dintr-o listă se va folosi funcția `len` dându-i ca parametru lista. Exemplu:

main.py

```
lista = [7, 4, 5, 12, -2.2]
print(len(lista))
```

rezultat

```
5
```

b) Tupluri

Un tuplu este un vector unidimensional ce poate conține orice număr de elemente de orice tip. Valoarea acesteia se poate inițializa enumerând valorile elementelor acesteia, separate de virgulă între un set de paranteze rotunde. Ulterior, un element al listei se poate accesa scriind numele variabilei urmat de indexul elementului între paranteze pătrate. Indexarea elementelor se poate face ca la liste, utilizând inclusiv indecsări negativi [1].

Diferența dintre un tuplu și o listă este că, un tuplu, odată definit, nu mai poate suferi modificări ale elementelor, lucru care înseamnă că tuplurile sunt IMMUTABILE. Conversia unui tuplu într-o listă se poate face utilizând funcția `list` căreia îi dam ca parametru tuplul.

Comparativ cu listele tuplurile se definesc utilizând paranteze rotunde.

Exemplu de definire a unui tuplu, accesarea elementelor acestuia și convertire în listă:

main.py

```
tuplu = (1, 2, 3, 4, 5)

print(tuplu)
print(tuplu[1])
print(tuplu[-2])

lista = list(tuplu)
print(lista)
```

rezultat

```
(1, 2, 3, 4, 5)
2
4
[1, 2, 3, 4, 5]
```

c) Seturi

Un set este un vector unidimensional ce poate conține orice număr de elemente de orice tip. Valoarea acestuia se poate inițializa enumerând valorile elementelor acestuia, separate de virgulă între un set de accolade. Diferența dintre un set și o listă este că elementele setului nu au o anumită ordine, deci nu pot fi indexate. De asemenea, deoarece elementele nu pot fi indexate, nu se pot modifica valorile elementelor deja existente. În plus, un set nu poate avea două elemente cu aceeași valoare [1].

Exemplu de definire a unui set:

main.py
my_set = {7, 4, 6, 2, 3, 4} print (my_set)
rezultat
{2, 3, 4, 6, 7}

Se pot adăuga elemente noi, sau se pot șterge elemente dintr-un set folosind metodele din tabelul următor:

Tabelul 2 - Metode pentru alterarea seturilor

Metoda	Descriere
<code>add(x)</code>	adăugă elementul x în set
<code>update(y)</code>	adăugă elementele din iterabilul y (lista/tuplu/set) în set
<code>remove(x)</code>	șterge elementul x din set

Exemplu:

main.py
my_set = {7, 4, 6, 2, 3, 7} print (my_set) my_set.add(-1) print (my_set) my_set.remove(7) print (my_set)
rezultat
{2, 3, 4, 6, 7}
{2, 3, 4, 6, 7, -1}
{2, 3, 4, -1}

Seturile pot fi văzute ca mulțimi matematice, de aceea **Python** implementează o serie de operații cu mulțimi sub forma unor metode pentru seturi [1]:

Tabelul 3 - Metode pentru operații cu mulțimi (seturi)

Metoda	Descriere
x.intersection(y)	returnează un set ce conține elementele comune seturilor x și y
x.union(y)	returnează un set ce conține toate elementele din seturile x și y
x.difference(y)	returnează un set ce conține toate elementele din setul x care nu sunt și în setul y
x.issuperset(y)	returnează True doar dacă toate elementele din setul y se află și în setul x
x.issubset(y)	returnează True doar dacă toate elementele din setul x se află și în setul y

Exemplu:

main.py
my_set1 = {1, 2, 3, 4} my_set2 = {3, 4, 5, 6}
print(my_set1.intersection(my_set2)) print(my_set1.union(my_set2)) print(my_set1.difference(my_set2))
rezultat
{3, 4} {1, 2, 3, 4, 5, 6} {1, 2}

d) Dicționare

Dicționarele sunt utilizate la stocarea perechilor de date de tipul cheie – valoare. Cheile trebuie să fie unice și de obicei de tip sir de caractere (deși se acceptă și tipuri numerice), în timp ce valorile pot fi de orice tip, inclusiv liste, obiecte sau chiar alte dicționare. Dicționarele de definesc între acolade, perechile de tip cheie – valoare fiind separate de virgulă, în timp ce între cheie și valoare se pune simbolul „:” (două puncte). Accesarea elementelor unui dicționar se face scriind numele dicționarului și între paranteze pătrate cheia căreia dorim să îi accesăm valoarea. Dacă încercăm să modificăm valoarea unei chei care nu există în dicționar, atunci se va adăuga o nouă pereche cheie – valoare în dicționar. Stergerea unei perechi din dicționar se face folosind instrucțiunea del aşa cum se observă în exemplul următor [1]:

main.py
nota = {"Colocviu": 3, "Prezenta": 1, "Examen": 6} print(nota) print(nota["Examen"]) # accesam un element
nota["Tema"] = 1 # adaugam element nou print(nota)

```
del nota["Prezenta"] # stergem element
print(nota)

rezultat
{'Colocviu': 3, 'Prezenta': 1, 'Examen': 6}
6
{'Colocviu': 3, 'Prezenta': 1, 'Examen': 6, 'Tema': 1}
{'Colocviu': 3, 'Examen': 6, 'Tema': 1}
```

D. List Comprehension

„List comprehension” este o modalitate de definire a unei noi liste pe baza elementelor unei alte liste, oferind o sintaxă mai scurtă decât dacă s-ar defini în mod clasic. Definirea unei noi liste folosind „list comprehension” se face scriind între paranteze pătrate expresia de generare a elementelor noii liste urmată de iterarea listei deja existente utilizând *for* și, optional, condiționarea elementelor utilizând *if*[1].

Spre exemplu, se dă o listă *listă1* conținând valorile $[-5, -2, 0, 1, 12, -1, 5, 7]$. Se dorește să se definească o nouă listă care să conțină rădăcina pătrată a elementelor mai mari sau egale cu 0 din *listă1*. În mod clasic, problema s-ar rezolva astfel:

```
main.py
listă1 = [-5, -2, 0, 1, 12, -1, 5, 7]
listă2 = []

for elem in listă1:
    if elem  $\geq 0$ :
        listă2.append(elem ** (1/2))

print(listă2)

rezultat
[0.0, 1.0, 3.4641016151377544, 2.23606797749979, 2.6457513110645907]
```

Utilizând „list comprehension” codul s-ar reduce la:

```
main.py
listă1 = [-5, -2, 0, 1, 12, -1, 5, 7]
listă2 = [elem ** (1/2) for elem in listă1 if elem  $\geq 0$ ]

print(listă2)

rezultat
[0.0, 1.0, 3.4641016151377544, 2.23606797749979, 2.6457513110645907]
```

E. Salvarea datelor

a) Lucrul cu fișiere

Pentru a lucra cu fișiere în **Python** (citire / salvare date) există mai multe metode. Una din ele este utilizând structura `with` [1]. Exemplu:

main.py

```
with open("exemplu.txt", "r") as my_file:
    # scriem / citim fisierul in interiorul structurii with
    ...
```

Această structură va crea o variabilă `my_file` ce va face referire la fișierul deschis cu funcția `open`. Această funcție primește doi parametri, primul fiind calea spre fișier, iar al doilea, modul de deschidere a fișierului, conform tabelului de mai jos [1]:

Tabelul 4 - Moduri de deschidere a fișierelor în Python

Mod	Descriere
r	deschide un fișier doar pentru citire; dacă fișierul nu există va rezulta o eroare
w	deschide un fișier pentru scriere; dacă fișierul nu există, îl va crea. Dacă fișierul există, conținutul acestuia va fi suprascris
a	deschide un fișier pentru scriere; dacă fișierul nu există, îl va crea. Dacă fișierul există, scrierea se va face după conținutul deja existent
x	deschide un fișier pentru scriere; dacă fișierul există deja, va rezulta o eroare

Astfel, pentru citire și scriere se vor folosi o serie de metode disponibile pentru variabila `my_file`:

Tabelul 5 - Metode pentru citirea și scrierea fișierelor

Mod	Metoda	Descriere
Citire	<code>my_file.read()</code>	va returna tot conținutul fișierului
	<code>my_file.read(x)</code>	va returna primele x caractere din fișier
	<code>my_file.readline()</code>	va returna o linie din fișier; dacă se va mai apela din nou această metodă, va rezulta următoarea linie, și aşa mai departe.
Scriere	<code>my_file.write(text)</code>	va scrie valoarea sirului de caractere <code>text</code> în fișier

Exemplu:

main.py

```
with open("tcst.txt", "w") as my_file:
    my_file.write("Prima linie din fisier\n")
    my_file.write("A doua linie din fisier\n")
    my_file.write("Ultima linie din fisier")
```

```
with open("test.txt", "r") as my_file:  
    print(my_file.readline())  
    print(my_file.readline())  
    print(my_file.readline())
```

rezultat

```
Prima linie din fisier  
A doua linie din fisier  
Ultima linie din fisier
```

b) Fișiere CSV

Formatul CSV (Comma Separated Values – Valori Separate de Virgulă) este unul din cele mai cunoscute formate pentru importare și exportarea datelor sub formă de tabel. Fiecare rând al fișierului CSV reprezintă un rând al tabelului, în timp ce valorile fiecărui rând sunt separate utilizând un separator (de obicei virgula). **Python** oferă suport pentru fișierele CSV în librăria csv (inclusă în limbajul de bază **Python** – nu este necesară instalarea). Această librărie definește două tipuri de obiecte [1]:

- `reader` – utilizat pentru a citi datele dintr-un fișier CSV. Este iterabil, deci practic fișierul se poate citi linie cu linie prin iterarea acestui obiect.
- `writer` - utilizat pentru a scrie date într-un fișier CSV. Pune la dispoziție metoda `writerow` ce primește ca parametru un obiect iterabil și îl adaugă ca linie în fișierul CSV.

În exemplul următor se poate vedea modul de definire al obiectelor `reader` și `writer`, precum și modul de scriere / citire a datelor

main.py

```
import csv  
  
x = [1, 7, 4, 5, 2, 3, 2, 2]  
y = [round(elem ** (1/2)) for elem in x]  
  
with open("test.csv", "w") as my_file:  
    writer_object = csv.writer(my_file, delimiter=',')  
  
    writer_object.writerow(x)  
    writer_object.writerow(y)  
  
with open("test.csv", "r") as my_file:  
    reader_object = csv.reader(my_file, delimiter=',')  
  
    for row in reader_object:  
        print(row)
```

rezultat

```
['1', '7', '4', '5', '2', '3', '2', '2']
['1', '3', '2', '2', '1', '2', '1', '1']
```

c) Fișiere JSON

Formatul JSON (JavaScript Object Notation) este un format foarte popular de salvare a datelor asemănător formatului în care poate fi definit un dicționar în **Python**. Astfel, utilizând acest format se pot salva orice tip de dată din **Python**. **Python** oferă suport pentru fișierele JSON în librăria `json` (inclusă în limbajul de bază **Python** – nu este necesară instalarea). Această librărie definește mai multe funcții, printre care `dump` (utilizată la scrierea unui fișier JSON) și `load` (utilizată la citirea unui fișier JSON) [1].

Exemplu de utilizare:

main.py

```
import json

studenti = [
    {
        "Nume": "Ion",
        "Prezente": 7,
        "Nota": 7.5
    },
    {
        "Nume": "Vasile",
        "Prezente": 0,
        "Nota": 2.5
    },
    {
        "Nume": "Gheorghe",
        "Prezente": 14,
        "Nota": 9.5
    }
]

with open("test.json", "w") as my_file:
    json.dump(studenti, my_file)

with open("test.json", "r") as my_file:
    bd_studenti = json.load(my_file)
    print(bd_studenti)
    print(bd_studenti[2]["Nume"])
```

rezultat

```
[{"Nume": "Ion", "Prezente": 7, "Nota": 7.5}, {"Nume": "Vasile", "Prezente": 0, "Nota": 2.5}, {"Nume": "Gheorghe", "Prezente": 14, "Nota": 9.5}]
```

Gheorghe

F. Cerințe

1. Fie lista de siruri de caractere: „8T”, „9IN”, „JT”, „QT”, „AR”. Definiți lista în **Python** și afișați-o pe ecran. Eliminați elementele „9IN” și „AR”, iar apoi inserați elementele „9T” și „10T” după elementul „8T” folosind metodele specifice listelor.
2. Se dă următoarea listă: `lista = [10, 6, 3, 4, 1, 2, 5, 7, 9]`
 - a) Afișați lista inițială.
 - b) Verificați tipul de date.
 - c) Adăugați un element la sfârșitul listei.
 - d) Inserați elementul „8” la poziția 7.
 - e) Determinați numărul de elemente din listă.
 - f) Sortați lista.
 - g) Extindeți lista cu o altă listă
 - h) Eliminați primul element din listă
 - i) Eliminați ultimul element din listă.
 - j) Afișați lista finală.
3. Se dă următorul tuplu: `tuplu = (1, "xyz", 3.14, False, [4, 6])`
 - a) Afișați tuplul inițial.
 - b) Afișați lungimea tuplului.
 - c) Creați un nou tuplu care să conțină doar primele trei elemente.
 - d) Testați modificarea valorii unui element (exemplu: `tuplu[0] = 10`).
 - e) Convertiți tuplul într-o listă.
 - f) Afișați tuplul final.
4. Se dă următorul set: `set = {2, 4, 6, 8, 10}`
 - a) Afișați setul inițial.
 - b) Adăugați numărul 9 setului.
 - c) Adăugați numărul 2 setului. Ce observați?
 - d) Stergeți un element din set folosind metoda `remove()`.
 - e) Stergeți un element din set folosind metoda `pop()`.
 - f) Actualizați setul cu `elemente_aditionale = {11, 12, 13}`.
 - g) Afișați setul final.
5. Definiți un dicționar, numit `student`, cu următoarele chei: nume, prenume, vîrstă, an de facultate, universitate și alegeti valori pentru fiecare cheie.
 - a) Afișați dicționarul inițial.
 - b) Afișați lungimea dicționarului.
 - c) Afișați numele de familie al studentului.
 - d) Adăugați un element nou în dicționar, reprezentând țara în care studiază studentul.
 - e) Actualizați anul de facultate.
 - f) Stergeți vîrstă
 - g) Afișați dicționarul final.

6. Se dă următoarea listă: `lista_initiala = [27, 88, 46, 52 , 3, 1, 17, 15, 4, 2, 1, 0]`
- Afişați lista inițială.
 - Utilizați list comprehension pentru a crea o listă cu elementele impare.
 - Utilizați list comprehension pentru a crea o listă cu elementele pare.
 - Afişați lista cu elementele impare și elementele pare.
7. Creați un fișier text numit „`lista_cumparaturi.txt`” cu următoarele cumpărături pe fiecare linie: Lapte, Carne, Unt, Cafea, Fructe.
- Citiți și afișați conținutul fișierului.
 - Adăugați un element nou în listă cum ar fi Legume.
 - Scrieți lista actualizată în fișierul „`lista_cumparaturi_acualizata.txt`”.
 - Afişați un mesaj în consolă care să anunțe că lista a scrisă în fișier.
8. Să se definească următoarele funcții **Python**:
- `valoare_maxima` – primește ca parametru o listă sau un tuple și returnează valoarea maximă din acesta.
 - `suma` – primește ca parametri două liste (`lista1, lista2`) și returnează o altă listă (`lista3`) însumând elementele din `lista1` și `lista2` element cu element. Dacă lungimea celor două liste primite ca parametru nu este egală atunci se va returna o listă goală.
 - `factorial` – primește ca parametru un număr întreg și returnează factorialul acestuia.
 - `export_csv` – primește ca parametru o listă și un nume de fișier și adaugă listă, în format CSV, la sfârșitul fișierului.
 - `rezistente_parallel` – primește ca parametru o listă reprezentând valorile unor rezistențe puse în paralel și returnează rezistența echivalentă:
- $$R_e = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2} + \dots + \frac{1}{R_n}}$$
9. Definiți următoarele mulțimi matematice $A = \{1, 7, 2, 8, 19, 1, -1, -5, -7, 0\}$ și $B = \{0, 9, 1, -8, 7, 22, -5, -4, -3, 2\}$ ca seturi în **Python**. Salvați cele două seturi într-un fișier CSV.
Calculați și afișați:
- $A \cup B$
 - $A \cap B$
 - $A \setminus B$
 - $B \setminus A$
10. Fie fișierul `config.json`. Încărcați conținutul fișierului într-un dicționar, afișați-l pe ecran, modificați valoarea cheii "MAX_ERR" din sub-dicționarul "TESTING" la 0.01 și actualizați fișierul cu noile modificări.