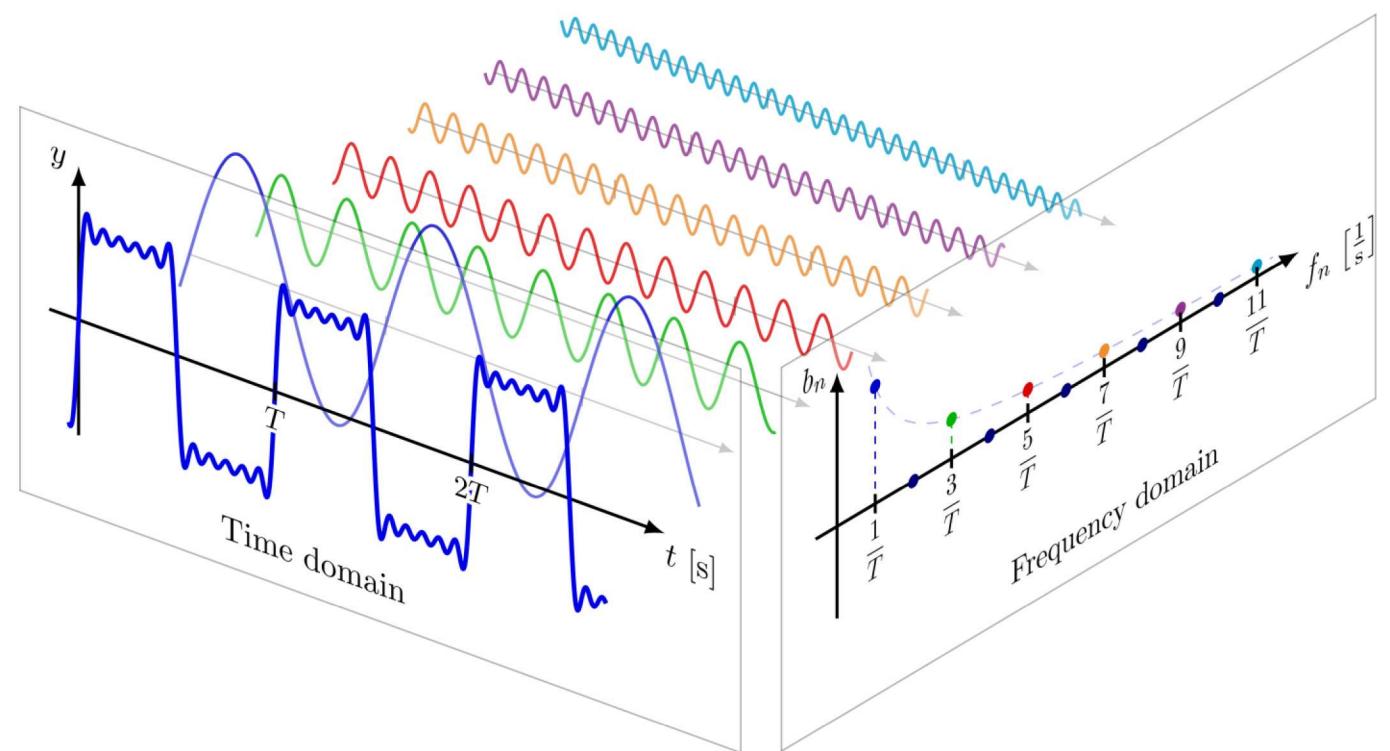


CURS 2 PD

Introducere în limbajul Python

Functii. Structuri de date



Funcții

Structuri de date

Salvarea datelor

1. Funcții

2. Structuri de date

3. Salvarea datelor

Funcții

Structuri de date

Salvarea datelor

Definiție

Funcțiile sunt o metodă de împărțire a codului în secvențe ce au anumite funcționalități și care pot fi apoi refoosite, 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

main.py

```
def afiseaza_titlu():
    # continutul functiei
    print("Procesarea Datelor")

    # apelarea functiei
afiseaza_titlu()
```

rezultat

Procesarea Datelor

Definiție

Funcțiile sunt o metodă de împărțire a codului în secvențe ce au anumite funcționalități și care pot fi apoi refoosite, 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).

- Funcțiile pot avea și parametri de intrare (unul sau mai mulți) – variabile ce pot fi transmise funcției în momentul apelării acesteia.

Exemplu

main.py

```
def numara(x, y):
    for i in range(x, y + 1):
        print(i)

# apelarea functiei
numara(2, 5)
```

rezultat

2
3
4
5

Definiție

Funcțiile sunt o metodă de împărțire a codului în secvențe ce au anumite funcționalități și care pot fi apoi refoosite, 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).

- 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
```

1. Funcții

2. Structuri de date

3. Salvarea datelor

Funcții

Structuri de date

Salvarea datelor

1. Funcții

2. Structuri de date

3. Salvarea datelor

Funcții

Structuri de date

Salvarea datelor

- **Lista** este un vector unidimensional ce poate conține orice număr de elemente de orice tip.
- Valoarea unei liste se initializează enumerând valorile elementelor acesteia, separate de virgulă între un set de paranteze drepte.
- 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 indecsări negativi, caz în care elementele vor fi numărate de la sfârșitul listei spre început (ex.: indexul **-1** reprezintă ultimul element al listei, indexul **-2** reprezintă penultimul element etc)
- Pentru a determina numărul de elemente dintr-o listă se va folosi funcția **len** dându-i ca parametru lista.

Exemplu	
main.py	rezultat
<pre>lista = ["triunghi", "patrat", "cerc"] print(lista[1]) # elementul cu indexul 1 print(lista[2]) # elementul cu indexul 2 print(lista[-1]) # ultimul element lista[0] = "hexagon" # modificare primul element print(lista) print(lista[0:2]) # elementele cu indecsări 0 și 1 print(len(lista)) # numarul de elemente din lista</pre>	<p>patrat cerc Cerc</p> <p>['hexagon', 'patrat', 'cerc']</p> <p>['hexagon', 'patrat']</p> <p>3</p>

- 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.
- Metode pentru alterarea listelor:

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

Exemplu	
main.py	rezultat
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)	[]

- **Tuplul** este un vector unidimensional ce poate conține orice număr de elemente de orice tip.
- Valoarea unui tuplu se initializează enumerând valorile elementelor acestuia, separate de virgulă între un set de paranteze **rotunde**.
- 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 negativei.
- 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 **IMUTABILE**.
- Conversia unui tuplu într-o listă se poate face utilizând funcția **list** căreia îi dam ca parametru tuplul.

Exemplu	
main.py	rezultat
<pre>tuplu = (1, 2, 3, 4, 5) print(tuplu) print(tuplu[1]) # elementul cu indexul 1 print(tuplu[-2]) # penultimul element din tuplu lista = list(tuplu) print(lista) tuplu2 = (10,) # tuplu cu un singur element print(tuplu2)</pre>	<pre>(1, 2, 3, 4, 5) 2 4 [1, 2, 3, 4, 5] (10,)</pre>

- **Setul** este un vector unidimensional ce poate conține orice număr de elemente de orice tip.
- Valoarea unui set se initializează enumerând valorile elementelor acestuia, separate de virgulă între un set de **acolade**.
- Diferența dintre un set și o listă este că elementele setului nu au o anumită ordine, deci nu pot fi indexate.
- Deoarece elementele nu pot fi indexate, nu se pot modifica valorile elementelor deja existente.
- Un set nu poate avea două elemente cu aceeași valoare

Exemplu	
main.py	rezultat
<pre>my_set = {7, 4, 6, 2, 3, 4} print(my_set)</pre>	{2, 3, 4, 6, 7}

- Se pot adăuga elemente noi, sau se pot șterge elemente dintr-un set folosind o serie de metode:

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

Exemplu	
main.py	rezultat
<pre>my_set = {7, 4, 6, 2, 3, 7} print(my_set)</pre>	{2, 3, 4, 6, 7}
<pre>my_set.add(-1) # adăugăm element cu valoarea -1 print(my_set)</pre>	{2, 3, 4, 6, 7, -1}
<pre>my_set.remove(7) # stergem elementul cu valoarea 7 print(my_set)</pre>	{2, 3, 4, 6, -1}

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

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

Exemplu	
main.py	rezultat
<pre>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))</pre>	<pre>{3, 4} {1, 2, 3, 4, 5, 6} {1, 2}</pre>

- **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
- 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.
- Ștergerea unei perechi din dicționar se face folosind instrucțiunea **del**

Exemplu	
main.py	rezultat
<pre>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)</pre>	<pre>{'Colocviu': 3, 'Prezenta': 1, 'Examen': 6} 6 {'Colocviu': 3, 'Prezenta': 1, 'Examen': 6, 'Tema': 1} {'Colocviu': 3, 'Examen': 6, 'Tema': 1}</pre>

Definiție

„**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**

Exemplu

Se dă o listă **lista1** 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 **lista1**.

Rezolvare fără List Comprehension**main.py**

```
lista1 = [-5, -2, 0, 1, 12, -1, 5, 7]
lista2 = []

for elem in lista1:
    if elem >= 0:
        lista2.append(elem ** (1/2))

print(lista2)
```

rezultat

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

**Rezolvare cu List Comprehension****main.py**

```
lista1 = [-5, -2, 0, 1, 12, -1, 5, 7]
lista2 = [elem ** (1/2) for elem in lista1 if elem >= 0]

print(lista2)
```

rezultat

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

1. Funcții

2. Structuri de date

3. Salvarea datelor

Funcții

Structuri de date

Salvarea datelor

1. Funcții

2. Structuri de date

3. Salvarea datelor

Funcții

Salvarea datelor

- Pentru a lucra cu fișiere în **Python** (citire / salvare date) există mai multe metode, una din ele fiind utilizarea structurii **with**.
- În exemplul anterior, în interiorul structurii **with** vom avea o variabilă `my_file` ce va face referire la fișierul deschis cu funcția `open`.
- Funcția `open` primește doi parametri:
 - un sir de caractere reprezentând calea spre fișier (în exemplu se deschide fișierul **exemplu.txt**).
 - un sir de caractere conținând un singur caracter reprezentând modul de deschidere a fișierului.
- Pentru citirea din fișier și scrierea în fișier, se vor folosi o serie de metode disponibile pentru variabila `my_file`.

main.py

```
with open("exemplu.txt", "r") as my_file:
    # scriem / citim fisierul in interiorul structurii with
    ...
# in afara structurii with nu mai avem acces la fisier
```

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

Metode pentru scrierea și citirea 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 șirului de caractere text în fișier

- Pentru citirea din fișier și scrierea în fișier, se vor folosi o serie de metode disponibile pentru variabila `my_file`.

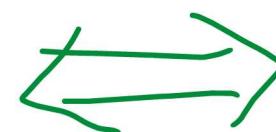
Metode pentru scrierea și citirea fișierelor		
Mod	Metoda	Descriere
Citire	<code>my_file.read()</code>	va returna tot conținutul fișierului
	<code>my_file.read(x)</code>	va returna primele x caractere din fișier
	<code>my_file.readline()</code>	va returna o linie din fișier; dacă se va mai apela din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

Exemplu

main.py	rezultat
<pre>with open("test.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())</pre>	Prima linie din fisier A doua linie din fisier Ultima linie din fisier

- **CSV** (*Comma Separated Values – Valori Separate de Virgulă*) este unul din cele mai cunoscute formate pentru importare și exportarea datelor sub formă de tabel.
- Fișierele CSV au extensia **.csv**
- Fiecare rând al fișierului CSV reprezintă un rând al tabelului, în timp ce valorile fiecărui rând sunt separate utilizând un separator (de obicei virgula).

11	12	13
21	22	23
31	32	33
41	42	43



Fișier CSV

11, 12, 13
21, 22, 23
31, 32, 33
41, 42, 43

- **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:
 - **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.

Exemplu	
main.py	rezultat
<pre>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: # scriem fisier CSV writer_object = csv.writer(my_file, delimiter=',') writer_object.writerow(x) writer_object.writerow(y) with open("test.csv", "r") as my_file: # citim fisier CSV reader_object = csv.reader(my_file, delimiter=',') for row in reader_object: print(row)</pre>	<pre>['1', '7', '4', '5', '2', '3', '2', '2'] ['1', '3', '2', '2', '1', '2', '1', '1']</pre>

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

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) # scriem fișier JSON

with open("test.json", "r") as my_file:
    bd_studenti = json.load(my_file) # citim fișier JSON
    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

1. Funcții

2. Structuri de date

3. Salvarea datelor

Funcții

Salvarea datelor

1. Funcții

2. Structuri de date

3. Salvarea datelor

Funcții

Structuri de date

Salvarea datelor