Documentație Tema

## Prerequisites & Run

Pentru a rula codul din această temă, este necesar să aveți instalate următoarele module Python:

1. pandas – pentru manipularea datelor.
2. matplotlib – pentru generarea și afișarea graficelor.

### Instalarea modulelor necesare

Pentru a instala modulele necesare, rulați următoarele comenzi în terminal sau Command Prompt:

pip install pandas matplotlib

### Structura Fișierelor

Asigurați-vă că aveți următoarele fișiere în același director:

* employee.py – codul pentru gestionarea angajaților și managerilor.
* plot\_data.py – codul pentru vizualizarea datelor.
* data.csv – fișierul CSV care conține datele pentru vizualizare.

### Rularea Codului

Pentru a rula codurile, folosiți următoarele comenzi în terminal:

* Pentru employee.py:

python employee.py

* Pentru plot\_data.py:

python plot\_data.py

## Explicație

#### Fișierul employee.py

#### Introducere

Codul conține două clase: Employee și Manager. Clasa Employee este o clasă de bază pentru gestionarea angajaților, iar clasa Manager moștenește funcționalitățile din Employee și adaugă caracteristici suplimentare pentru manageri. Vom detalia fiecare metodă și vom răspunde la întrebările posibile pentru a oferi o înțelegere completă.

### Definirea Clasei Employee

class Employee:

"""Clasa de bază comună pentru toți angajații"""

empCount = 0 # Variabilă de clasă care ține evidența numărului total de angajați

* Explicație:
  + Variabila empCount este o variabilă de clasă de tip int, folosită pentru a număra câți angajați au fost creați.

#### Constructorul \_\_init\_\_

def \_\_init\_\_(self, name, salary):

# Constructor care inițializează un angajat cu nume, salariu și un dicționar de sarcini

self.name = name

self.salary = salary

self.tasks = {} # Dicționar pentru a stoca sarcinile și statusul acestora

Employee.empCount += 1 # Incrementăm numărul de angajați de fiecare dată când creăm un obiect Employee

* Întrebare: De ce tip este variabila self.tasks la linia 8?
  + self.tasks este de tip dict (dicționar), folosit pentru a asocia sarcinile cu statusul lor.
* Întrebare: Ce face Employee.empCount += 1 la linia 9?
  + Această linie incrementează numărul total de angajați de fiecare dată când un nou angajat este creat.

#### Metoda display\_emp\_count

def display\_emp\_count(self):

"""Afișează numărul de angajați"""

print(f"Total number of employee(s) is {Employee.empCount}")

* Întrebare: Ce afișează această metodă la linia 14?
  + Afișează numărul total de angajați din clasa Employee.

#### Metoda display\_employee

def display\_employee(self):

# Afișează numele și salariul angajatului

print("Name : ", self.name, ", Salary: ", self.salary)

* Explicație:
  + Această metodă afișează numele și salariul angajatului curent.

#### Destructorul \_\_del\_\_

def \_\_del\_\_(self):

# Destructor care decrementază numărul de angajați atunci când un obiect Employee este șters

Employee.empCount -= 1

* Întrebare: Ce face Employee.empCount -= 1 la linia 19?
  + Scade numărul total de angajați atunci când un obiect Employee este șters.

#### Metoda update\_salary

def update\_salary(self, new\_salary):

# Metodă care actualizează salariul angajatului

self.salary = new\_salary

* Explicație:
  + Această metodă permite actualizarea salariului unui angajat cu o valoare nouă.

#### Metoda modify\_task

def modify\_task(self, task\_name, status="New"):

# Metodă care adaugă sau modifică o sarcină pentru angajat, cu un status dat (implicit "New")

self.tasks[task\_name] = status

* Întrebare: Ce face self.tasks[task\_name] = status la linia 25?
  + Adaugă o sarcină nouă în dicționarul tasks sau actualizează statusul unei sarcini existente.

#### Metoda display\_task

def display\_task(self, status):

# Metodă care afișează toate sarcinile cu un status specificat

print(f"Taskuri cu statusul {status}")

for name in self.tasks.keys():

if self.tasks[name] == status:

print(name)

* Explicație:
  + Afișează toate sarcinile din dicționarul tasks care au statusul specificat.

### Definirea Clasei Manager

class Manager(Employee):

mgr\_count = 0 # Variabilă de clasă care ține evidența numărului total de manageri

#### Constructorul \_\_init\_\_

def \_\_init\_\_(self, name, salary, department):

super().\_\_init\_\_(name, salary) # Apelăm constructorul clasei de bază (Employee)

self.department = department # Adăugăm o nouă proprietate pentru departament

Manager.mgr\_count += 1 # Incrementăm numărul de manageri de fiecare dată când creăm un obiect Manager

* Întrebare: De ce folosim super().\_\_init\_\_(name, salary) la linia 36?
  + Apelăm constructorul clasei Employee pentru a inițializa numele și salariul managerului.

#### Metoda display\_employee Suprascrisă

def display\_employee(self, X):

if X % 3 == 0:

print("Name: ", self.name)

elif X % 3 == 1:

print("Salary: ", self.salary)

elif X % 3 == 2:

print("Department: ", self.department)

* Explicație:
  + Această metodă afișează diferite informații în funcție de valoarea lui X.

### Codul Principal (Funcții de Testare)

if \_\_name\_\_ == "\_\_main\_\_":

X = 5

Y = 12

# Crearea obiectelor Employee

emp1 = Employee("Ion", 3000)

emp2 = Employee("Maria", 3500)

# Crearea obiectelor Manager

mgr1 = Manager("Andrei", 5000, "DevTeam\_A")

mgr2 = Manager("Elena", 5500, "DevTeam\_A")

mgr3 = Manager("Radu", 6000, "DevTeam\_A")

mgr4 = Manager("Mihai", 6200, "DevTeam\_A")

# Apelarea metodei display\_employee

emp1.display\_employee()

emp2.display\_employee()

mgr1.display\_employee(X)

mgr2.display\_employee(X)

mgr3.display\_employee(X)

mgr4.display\_employee(X)

# Afișarea numărului total de angajați

print("\nTotal Employees:")

emp1.display\_emp\_count()

print("Total Managers: ", Manager.mgr\_count)

### Explicatie

1. De ce este folosit if \_\_name\_\_ == "\_\_main\_\_":?
   * Pentru a rula codul de testare doar când fișierul este executat direct, nu când este importat.
2. Ce tipuri de date sunt X și Y?
   * Ambele sunt de tip int.
3. Ce face metoda display\_emp\_count?
   * Afișează numărul total de angajați existenți.

#### Fișierul plot\_data.py

#### Introducere

Acest script Python utilizează bibliotecile pandas și matplotlib pentru a manipula și vizualiza date dintr-un fișier CSV (data.csv). Codul afișează toate datele, primele X linii și ultimele Y linii din coloanele „Durata” și „Puls”. Mai jos vei găsi explicații detaliate pentru fiecare secțiune a codului, împreună cu răspunsuri la întrebări frecvente.

### Importarea Bibliotecilor

import pandas as pd # Importă biblioteca pandas, utilizată pentru manipularea datelor

import matplotlib.pyplot as plt # Importă biblioteca matplotlib pentru crearea de grafice

* Întrebare: Ce face import pandas as pd la linia 1?
  + Importă biblioteca pandas sub aliasul pd, ceea ce facilitează utilizarea funcțiilor din bibliotecă.
* Întrebare: De ce este folosit import matplotlib.pyplot as plt?
  + Importă modulul pyplot din biblioteca matplotlib, utilizat pentru a crea și afișa grafice.

### Citirea Datelor din Fișierul CSV

df = pd.read\_csv('data.csv') # Citește fișierul CSV 'data.csv' și îl încarcă într-un DataFrame numit df

* Explicație:
  + Funcția pd.read\_csv citește datele din fișierul data.csv și le încarcă într-un DataFrame, o structură de date utilizată pentru analiza datelor în pandas.
* Întrebare: Ce tip de variabilă este df?
  + df este de tip DataFrame.

### Afișarea Tuturor Datelor

print(df) # Afișează toate datele din DataFrame

df.plot(title="Toate Valorile") # Creează un grafic cu toate datele din df și adaugă un titlu

plt.show() # Afișează graficul generat

* Explicație:
  + print(df) afișează toate datele din DataFrame în consola de execuție.
  + df.plot() generează un grafic pentru toate coloanele numerice din DataFrame.
  + plt.show() afișează graficul creat.
* Întrebare: Ce face df.plot(title="Toate Valorile")?
  + Creează un grafic pentru toate datele din df și adaugă titlul „Toate Valorile”.

### Definirea Variabilelor X și Y

X = 5

Y = 12

* Explicație:
  + X și Y sunt două variabile de tip int utilizate pentru a specifica numărul de linii care vor fi afișate ulterior.

### Afișarea Primelor X Valori

print(df.head(X)) # Afișează primele X linii (primele 5 linii) din DataFrame

df.head(X).plot(title=f"Primele {X} Valori") # Creează un grafic cu primele X linii din df și adaugă un titlu

plt.show() # Afișează graficul generat

* Explicație:
  + df.head(X) returnează primele X linii din DataFrame.
  + df.head(X).plot() generează un grafic pentru aceste linii.
  + plt.show() afișează graficul.
* Întrebare: Ce face df.head(X) la linia 17?
  + Afișează primele X (5) linii din DataFrame.

### Afișarea Ultimelor Y Valori pentru Coloanele „Durata” și „Puls”

print(df.tail(Y)[['Durata', 'Puls']]) # Afișează ultimele Y (12) linii din df doar pentru coloanele 'Durata' și 'Puls'

df.tail(Y)[['Durata', 'Puls']].plot(title=f"Ultimele {Y} Valori pentru Durata și Puls") # Creează un grafic cu ultimele Y valori pentru 'Durata' și 'Puls'

plt.show() # Afișează graficul generat

* Explicație:
  + df.tail(Y) returnează ultimele Y (12) linii din DataFrame.
  + [['Durata', 'Puls']] selectează doar coloanele „Durata” și „Puls”.
  + df.tail(Y)[['Durata', 'Puls']].plot() generează un grafic pentru aceste date.
  + plt.show() afișează graficul.
* Întrebare: Ce face df.tail(Y)[['Durata', 'Puls']] la linia 22?
  + Afișează ultimele Y (12) linii din DataFrame pentru coloanele „Durata” și „Puls”.

### Explicatie

1. Ce este un DataFrame?
   * Un DataFrame este o structură de date bidimensională din pandas, similară cu un tabel Excel.
2. De ce folosim plt.show()?
   * Pentru a afișa graficele create cu matplotlib.
3. Ce se întâmplă dacă data.csv nu există?
   * Programul va arunca o eroare FileNotFoundError. Este necesar să te asiguri că fișierul data.csv există în directorul curent.

# Surse:

1. Pandas Documentation: https://pandas.pydata.org/pandas-docs/stable/  
2. Matplotlib Documentation: https://matplotlib.org/stable/contents.html  
3. Python Official Documentation: https://docs.python.org/3/  
4. ChatGPT: https://chat.openai.com  
5. W3Schools Python Functions: https://www.w3schools.com/python/