Documentație

Informații generale despre temă:

- Nume student: Iosu Ștefan Lucian
- Variabile utilizate:
  - X =4;
  - Y = 12 ;

Exercițiul 1: Implementarea clasei Employee și Manager

1. prerequisites & run

Pentru a rula acest cod este necesar:
- Instalarea Python . Se poate descărca de pe python.org.
- Utilizarea unui editor de cod precum Visual Studio Code. Configurare:
  1. Instalează extensia Python din secțiunea "Extensions" (iconița cu 4 pătrate).
  2. Creează un fișier numit employee.py.
  3. Inserează codul sursă în acest fișier.

Pentru rulare:
1. Deschide Visual Studio Code, navighează la locația fișierului, apoi apasă pe "Run".
2. Alternativ, în linia de comandă:
   cd C:\Calea\Catre\Proiect
   python employee.py

2. Explicație detaliată a codului sursă

```
class Employee:
   """Common base class for all employees"""
   empCount = 0
```

- Se definește o clasă `Employee`, care reprezintă o bază comună pentru toți angajații.
- Atributul de clasă `empCount` este utilizat pentru a ține evidența numărului total de angajați creați.

```
def __init__(self, name, salary):
   self.name = name
   self.salary = salary
   self.tasks = {}
   Employee.empCount += 1
```

- Constructorul (`__init__`) inițializează obiectele clasei `Employee` cu:
  - `name`: numele angajatului.
  - `salary`: salariul angajatului.
  - `tasks`: un dicționar pentru gestionarea sarcinilor atribuite.

- Crearea unui obiect incrementează contorul empCount.

```
def display_emp_count(self):
  "Displays the number of employees"
  print(f"Total number of employee(s) is {Employee.empCount}")
```

- Metoda afișează numărul total de angajați folosind atributul empCount.

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

- Afișează detalii despre un angajat: numele și salariul.

```
def __del__(self):
  Employee.empCount -= 1
```

- Destructorul (__del__) decrementează contorul empCount când un obiect este distrus.

```
def update_salary(self, new_salary):
  self.salary = new_salary
def modify_task(self, task_name, status="New"):
  self.tasks[task_name] = status
```

- Adaugă sau actualizează o sarcină în dicționarul tasks, cu un status implicit „New”.

```
def display_task(self, status):
  print(f"Taskuri cu statusul {status}")
  for name in self.tasks.keys():
    if self.tasks[name] == status:
      print(name)
```

- Afișează sarcinile unui angajat care au un anumit status.

```
class Manager(Employee):
    mgr_count = 0
```

- Clasa `Manager` moștenește `Employee` și adaugă funcționalități specifice managerilor.
- Atributul `mgr_count` ține evidența numărului total de manageri.

```
def __init__(self, name, salary, department):
    super().__init__(name, salary)
    self.department = "F04" + department
    Manager.mgr_count += 1
```

- Constructorul clasei `Manager` inițializează:
  - `name` și `salary` prin apelarea constructorului clasei părinte (`super().__init__`).

- `department`: se prefixează „F04” înainte de numele departamentului.
- Incrementează contorul `mgr_count`.

```
def display_employee(self):
    X = 4
    if X % 3 == 0:
        print("Name: ", self.name)
    elif X % 3 == 1:
        print("Salary: ", self.salary)
    elif X % 3 == 2:
        print("Department: ", self.department)
```

- Suprascrie metoda `display_employee` din `Employee`.
- Afişează diferite informaţii în funcţie de valoarea lui `X` (4):
  - Dacă `X % 3 == 0`: afişează numele.
  - Dacă `X % 3 == 1`: afişează salariul.
  - Dacă `X % 3 == 2`: afişează departamentul.

```
employee1 = Employee("Tudor", 5000)
employee2 = Employee("Ion", 6000)
```

- Creează două obiecte `Employee` cu numele „Tudor” şi „Ion” şi salariile 5000 şi 6000.

```
manager1 = Manager("Stefan", 7000, "F04")
manager2 = Manager("Teo", 8000, "F04")
manager3 = Manager("Marius", 9000, "F04")
manager4 = Manager("Sofiia", 7500, "F04")
```

- Creează patru obiecte `Manager` cu numele, salariile şi departamentele specificate.

```
print("Employee1:")
employee1.display_employee()
```

- Afişează detaliile angajatului „Tudor”.

```
print("\nManager1:")
manager1.display_employee()
```

- Afişează detalii despre managerul „Stefan” în funcţie de valoarea lui `X`.

```
print("\nTotal Employees: ", Employee.empCount)
print("Total Managers: ", Manager.mgr_count)
```

- Afişează numărul total de angajaţi şi manageri folosind `empCount` şi `mgr_count`.

Rezultate obținute

1. Afișarea informațiilor despre angajați și manageri.
2. Numărul total de angajați este afișat cu empCount.
3. Numărul total de manageri este afișat cu mgr_count.

Exercițiul 2: Vizualizarea datelor dintr-un fișier CSV cu Pandas și Matplotlib

1. prerequisites & run

Pentru rulare:

1. Instalează modulele necesare:
   pip install pandas matplotlib
2. Asigurare că fișierul data.csv este disponibil în locația specificată.

2. Explicație detaliată a codului

```
import matplotlib.pyplot as plt
import pandas as pd
```

- **`matplotlib.pyplot`**: Biblioteca utilizată pentru generarea și personalizarea graficelor.
- **`pandas`**: Biblioteca utilizată pentru manipularea și analiza datelor.

Citirea fișierului CSV

```
df = pd.read_csv('C:/Users/iosus/Downloads/data.csv')
```

- Se citește fișierul `data.csv` utilizând funcția `pd.read_csv` din Pandas.
- Rezultatul este un `DataFrame`, o structură de date bidimensională asemănătoare unui tabel.

Graficul tuturor valorilor

```
df.plot()
plt.title("Toate valorile")
plt.ylabel("Valori")
plt.legend(title="Parametrii")
plt.show()
```

1. **`df.plot()`**: Creează un grafic care afișează toate coloanele din `DataFrame` pe axa Y, iar indexul rândurilor pe axa X.
2. **`plt.title("Toate valorile")`**: Adaugă titlul graficului: „Toate valorile”.
3. **`plt.ylabel("Valori")`**: Etichetează axa Y cu textul „Valori”.
4. **`plt.legend(title="Parametrii")`**: Adaugă o legendă pentru fiecare coloană, cu titlul „Parametrii”.
5. **`plt.show()`**: Afișează graficul.

Graficul primelor 4 valori

```
df[:4].plot()
plt.title("Primele 4 valori")
plt.ylabel("Valori")
plt.legend(title="Parametrii")
plt.show()
```

1. **`df[:4]`**: Selectează primele 4 rânduri din `DataFrame` (folosind slicing: `0:4`).
2. Restul codului este similar cu secțiunea anterioară, dar titlul indică faptul că se afișează doar primele 4 valori.

Graficul ultimelor 12 valori

```
df[-12:].plot()
plt.title("Ultimele 12 valori")
plt.ylabel("Valori")
plt.legend(title="Parametrii")
plt.show()
```

1. **`df[-12:]`**: Selectează ultimele 12 rânduri din `DataFrame` (slicing negativ: de la al 12-lea ultim rând până la final).
2. Restul codului este identic, dar titlul specifică afișarea ultimelor 12 valori.

Referințe bibliografice

- Documentația disciplinei.
  https://www.w3schools.com/python/pandas/default.asp
- *https://www.w3schools.com/python/matplotlib_intro.asp*
- *calp.python.extras de la Files > Class.Materials*
- *https://code.visualstudio.com/docs/python/tutorial-flask*
- *https://github.com/microsoft/python-sample-vscode-flask-tutorial*
- *https://code.visualstudio.com/docs/python/testing*
- https://www.w3schools.com/python/pandas/pandas_csv.asp
- https://www.w3schools.com/python/python_classes.asp
- https://www.w3schools.com/python/matplotlib_pyplot.asp
- https://www.youtube.com/watch?v=ZDa-Z5JzLYM