Pentru tema a fost necesara folosirea variabilelor X si Y care in cazul meu au fost X=5 si Y=17;

## EXERCITIUL 1

### PREREQUISITES & RUN

Pentru a implementa acest cod este necesara si preferabil recomanda versiunea Python 3.10,insa orice versiune 3.8+ este buna.De asemenea,este necesar si un terminal unde se pot executa fisiere .py precum VScode(folosit si de mine la rezolvarea temei).

Pentru rulare se creaza un fisier nou cu extensia exercitiul 1 tema de parcurs.py(in cazul meu) in care se introduce codul sursa iar apoi rularea codului se poate realiza cu comanda python exercitiul 1 tema de parcurs.py

### EXPLICATIA CODULUI

Voi incepe cu explicarea codului sursa:

Clasa Employee este clasa de baza ce are ca scop reprezentarea tuturor angajatilor.

In interiorul clasei Employee regasim:

Atributul de clasa empCount=0 folosit pentru a tine evidenta tuturor angajatilor creati.

Constructorul

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

        self.name = name

        self.salary = salary

        self.tasks = {}

**Employee**.empCount += 1

Care are rolul de a initializa un angajat cu nume,salariu si totodata creeaza un dictionar care stocheaza sarcini de lucru dar si incrementeaza numarul angajatilor;

Metodele folosite in clasa Employee sunt urmatoarele:

def **display\_emp\_count**(self):

        "Displays the number of employees"

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

folosita cu scopul de a afisa numarul total al angajatilor

def **display\_employee**(self):

**print**("Name :", self.name, ", Salary:", self.

Folosita cu scopul de a afisa detaliile despre angajat,adica numele respective salariul.

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

**Employee**.empCount -= 1

Destructorul folosit pentru a decrementa numar de angajati atunci cand unul din obiectele create(angajati) este distrus(sters)

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

        self.salary = new\_salary

folosita cu scopul de a putea schimba valoarea salariului angajatului

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

        self.tasks[task\_name] = status

folosita in scopul de a permite modificarea statusului unui task deja existent sau adaugarea altui task cu statusul implicit de “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)

permite afisarea taskurilor fiecarui angajat impreuna cu statusul acestuia

Clasa Manager

class **Manager**(**Employee**):

este o clasa mostenita de la clasa Employee

mgr\_count = 0

aceasta este o variabila de clasa ce initializeaza numarul mangerilor cu 0 si tine evidenta obiectelor de tip manger create;

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

**super**().**\_\_init\_\_**(name, salary)

        Nume\_Echipa="F33 "

        self.department=Nume\_Echipa+departament

Manager.mgr\_count += 1

constructorul clasei Manager care apeleaza constructorul clase de baza Employee pentru a accesa atributele name si salary si adauga un alt atribut “departament” care adauga un departament managerului create apoi incrementeaza numarul mangerilor.

    def **display\_employee**(self):

**print**(f"Department:{self.department}")

este o metoda suprascrisa metodei clasei de baza ce afiseaza doar departamentul managerului.(deoarece 5%3==2)

manager1=**Manager**("Stefania",3900,"HR")

manager2=**Manager**("Diana",3950,"IT")

manager3=**Manager**("Amalia",3800,"Marketing")

manager4=**Manager**("Tudor",3500,"Finance")

manager5=**Manager**("Vlad",4000,"Sales")

aici am creat 5 obiecte de tip manger (deoarece 17%3==5)

manager1.**display\_employee**()

manager2.**display\_employee**()

manager3.**display\_employee**()

manager4.**display\_employee**()

manager5.**display\_employee**()

afisarea celor 5 obiecte de tip Manager cu metoda display.employee suprascrisa ce va afisa doar departamentul acestora;

employee1=**Employee**("Stefan",4300)

employee2=**Employee**("Rares",2256)

employee1.**display\_employee**()

employee2.**display\_employee**()

crearea si afisarea a doua obiecte de tip Employee pentru a vedea diferenta dintre metode

**print**(f"Număr total angajați (Employee.empCount): {**Employee**.empCount}")

**print**(f"Număr total manageri (Manager.mgr\_count): {**Manager**.mgr\_count}")

afisarea numarului total de manageri si de angajati pentru instantele mai sus create.

## EXERCITIUL 2

### PREREQUISITES & RUN

In plus fata de exercitiul anterior este necesara instalarea modulelor necesare si anume Pandas si Matplotlib.Pentru instalare se va deschide un nou terminal in care se vor rula comenzile “pip install Pandas” respective “pip install Matplotlib” Apoi va trebui sa ne asiguram ca fisierele se regasesc in locatiile potrivite.Dupa rulare codul trebuie sa ne genereze 3 grafice.

### EXPLICATIA CODULUI

import **matplotlib**.**pyplot** as **plt**

import **pandas** as **pd**

acestea sunt bibliotecile folosite,mai exact pentru a crea grafice respective pentru a manipula si a analiza datele.

df = **pd**.**read\_csv**(r'C:\Users\Stefi Manda\Desktop\Python 2024\data.csv')

aici se citeste fisierul CSV care va duce la df ce este un Data Frame ce are o structura asemanatoare unei matrice.

df.plot()

**plt**.**title**("Toate valorile")

**plt**.**xlabel**("Contor")

**plt**.**ylabel**("Valori")

**plt**.**legend**(title="Parametrii")

**plt**.**grid**()

**plt**.**show**()

acesta reprezinta graficul pentru toate valorile din fisierul csv. df[['Durata', 'Puls','MaxPuls', 'Calorii']].plot() aceasta comanda creaza un grafic folosindu se de toate valorile si elementele din fisier iar comana **plt**.**show**() afiseaza graficul.Restul comenzilor sunt folosite strict pentru diverse caracteristici ale graficului precum titlu,abscisa,ordonata,legenda.

df.**head**(5).plot()

**plt**.**title**("Primele 5 valori")

**plt**.**xlabel**("Contor")

**plt**.**ylabel**("Valori")

**plt**.**legend**(title="Parametrii")

**plt**.**grid**()

**plt**.**show**()

Diferenta fata de graficul anterior este ca acesta va afisa graficul doar pentru primele 5 valori ale fisierului folosind comanda head().

df[['Durata','Puls']].**tail**(17).plot()

**plt**.**title**("Ultimele 17 valori")

**plt**.**xlabel**("Contor")

**plt**.**ylabel**("Valori")

**plt**.**legend**(title="Parametrii")

**plt**.**grid**()

**plt**.**show**()

Acest grafic va fi afisat doar pentru valorile “Durata” respective “Puls” regasite in fisierul csv.Totodata acesta va fi facut pentru ultimele 17 valori ale fisierului lucru realizat prin comanda tail().

<https://github.com/dianavlad04/tema_proiect/tree/stefaniamanda> linkul de github

REFERINTE BIBLIOGRAFICE

<https://www.w3schools.com/python/default.asp>

[Python Full Course for Beginners](https://www.youtube.com/watch?v=_uQrJ0TkZlc)

[shared – Google Drive](https://drive.google.com/drive/folders/10v9epSZp2e0Q6gyLV9C4QLQbYHdivZww)-documentatia de pe teams