Compte rendu TP2

1- Gestion de salariés :

On souhaite développer un programme de gestion de salaires des employés d'une entreprise. Un employé est caractérisé par son prénom, nom et âge. Nous distinguons entre deux catégories d'employés : les commerciaux et les techniciens. Parmi les commerciaux on distingue entre les vendeurs et les représentants. Les techniciens peuvent être des techniciens de production ou de manutention. On rappelle que le salaire minimum mensuel (SMIC) est de 1678€, et que le SMIC journalier est fixé à 60,4 € Les règles de calcul des salaires dépendent de la catégorie des employés.

I Le salaire d'un vendeur est égale au SMIC mensuel auquel on ajoute 10% du chiffre d'affaires réalisé par le vendeur I Le salaire d'un représentant est égal au SMIC mensuel auquel on ajoute 15% du chiffre d'affaires réalisé par le représentant. I Le salaire d'un technicien de production est à égale au SMIC journalier multiplié par le nombre de jours travaillés. On y ajoute 5 epar pièce produit. I Le salaire d'un technicien de manutention est égale au SMIC journalier multiplié par le nombre de jours travaillés. Donner les classes nécessaires pour la gestion des salaires des employés de cette entreprise.

J’ai commencé par la création de l’entrée des données identifiantes d’un employée utilisateur (nom, prénom, âge)

```
#definition de l'id de l'employé demandé
def employe():
    e["Nom"]=input("Donner le nom d'employé :") #entrée du nom
    e["prenom"]=input("Donner le Prénom d'employé :") #entrée de prénom
    e["age"]=input("Donner l'age d'employé :") # entrée de l'age

```

Donne en sortie

```
PS C:\Users\moham> & "C:/Program Files/WindowsApps/PythonSoftwareFoundation.Python.3.10
Donner le nom d'employé :mzoughi
Donner le Prénom d'employé :mohamed amine
Donner l'age d'employé :18
```

Ensuite j’ai défini les smic dans le système

```
#SMIC
SMIC=1678
#SMIC journalier
SMICj=60.4
#valeur initial de salaire
sal=0
```

Ensuite, j’ai créé un mécanisme avec lequel le programme, par le biais de quelques questions, choisit l’équation qui permet de calculer le bon salaire de l’employée

Voici le code :

```python
def salair(SMIC,SMICj,sal):
    t1=input("donner la catégorie d'employé :")
    while ((t1.upper()!="COMMERCIAL") and (t1.upper()!="TECHNICIEN")):
        t1=input("verifier la catégorie d'employé :")
    if (t1.upper()=="COMMERCIAL") :
        t2=input("est il un vendeur ou un représentant ? :")
        if (t2.upper()=="VENDEUR"):
            ca=float(input("donner le chiffre d'affaire :"))
            sal=((ca/100)*10)+sal+SMIC
            e["salaire"]=str(sal)+"€"
            print("la salaire de votre "+ t1+"  "+"  "+t2+" est :",sal,"€")
        elif(t2.upper()=="REPRESENTANT"):
            ca=float(input("donner le chiffre d'affaire :"))
            sal=sal+((ca/100)*15)+SMIC
            e["salaire"]=str(sal)+"€"
            print("la salaire de votre "+ t1+"  "+"  "+t2+" est :",sal,"€")

    elif (t1.upper()=="TECHNICIEN") :
        t2=input("est il un technicien de production ou de manutention ? :")
        if (t2.upper()=="PRODUCTION"):
            njt=float(input("donner le nombre de jours travaillé :"))
            pp=float(input("donner le nombre de piéces produits :"))
            sal=sal+(SMICj*njt)+(pp*5)
            e["salaire"]=str(sal)+"€"
            print("la salaire de votre "+ t1+"  "+"  "+t2+" est :",sal,"€")
        elif(t2.upper()=="MANUTENTION"):
            njt=float(input("donner le nombre de jours travaillé :"))
            sal=sal+(SMICj*njt)
            e["salaire"]=str(sal)+"€"
            print("la salaire de votre "+ t1 +"  "+"  "+ t2 +" est :",sal,"€")
        else:
            t2=input("Verifier votre saisie ! :")
#program principale
e={}
employe()
salair(SMIC,SMICj,sal)
print(e)
```

Test du code :

```
Donner le nom d'employé :mzoughi
Donner le Prénom d'employé :mohamed amine
Donner l'age d'employé :18
donner la catégorie d'employé :commercial
est il un vendeur ou un représentant ? :vendeur
donner le chiffre d'affaire :20000
la salaire de votre commercial    vendeur est : 3678.0 €
{'Nom': 'mzoughi', 'prenom': 'mohamed amine', 'age': '18', 'salaire': '3678.0€'}
PS C:\Users\moham>
```

```
PS C:\Users\moham> & C:/Program Files/WindowsApps/PythonSoftwareFoundation.Python.3.10
Donner le nom d'employé :mzoughi
Donner le Prénom d'employé :mohamed amine
Donner l'age d'employé :18
donner la catégorie d'employé :technicien
est il un technicien de production ou de manutention ? :production
donner le nombre de jours travaillé :25
donner le nombre de piéces produits :20
la salaire de votre technicien    production est : 1610.0 €
{'Nom': 'mzoughi', 'prenom': 'mohamed amine', 'age': '18', 'salaire': '1610.0€'}
PS C:\Users\moham>
```

2- Pile et file

Une pile (resp. File) est une structure de données qui permet de stocker des objets et de les retirer en mode dit LIFO : Last In First Out (resp. FIFO First in First Out).

On s'intéresse ici à la programmation de file ou de pile à capacité bornée.

Une pile offre les méthodes suivantes :

I) push(obj) permet d'empiler l'élément obj si la pile n'est pas pleine.

I) pop () permet de retourner l'élément en tête de la pile si elle n'est pas vide. .

1) Proposer une classe Pile qui modélise une pile bornée.

```python
from logging import exception
# identificatin des entrées avec test
class Pile(object):
    def __init__(self, size):
        assert isinstance(size, int) and size >0
        self.size=size
        self.pointer=size
        self.tab= [0]* size

# définition de la capacité vide
    def vide (self):
        return (self.pointer==self.size)
# définition de la capacité pleine
    def plein(self):
       return(self.pointer==0)

 #permet d'empiler l'objet
    def push(self, obj):
        if not self.full():
            self.pointer=self.pointer-1
            self.tab[self.pointer]=obj
        else:
            raise Exception

    #permet de retourner l'element en tête
    def pop(self):
        if not self.empty():
            res=self.tab[self.pointer]
            self.pointer=self.pointer +1
            return res
        else:
            #rejet de l'exception
            raise Exception
```

2) En utilisant la classe Pile proposer une classe File qui modélise une file d'attente bornée.

```
from queue import Empty


class File (object):
    
    def __init__(self,size):
        self.p1=pile(size)
        self.p2=pile(size)

    def vide (self):
        return self.p1.vide()

    def plein(self):
        return self.p1.plein()

    def ajouter(self, obj):
        return self.p1.push(obj)

```

3-Gestion d'un parking

Une société de gestion de parkings de voitures souhaite mettre en place un système automatique d'affichage en temps réel de nombre de places disponibles dans un parking.

Un parking est caractérisé par : son adresse, sa capacité d'accueil, et le nombre de portails.

Chaque portail est doté d'un écran affichant un message précisant les informations suivantes : — le numéro du portail,

— le nombre de places libres dans le parking (ou si le parking est complet), — le nombre d'entrées et le nombre de sorties effectuées par le portail.

Donner les classes nécessaires pour effectuer cette simulation.

J'ai commencé par la réflexion totale sur tout l'exercice pour bien le comprendre vu qu'on va le reprendre dans le TP3

Faire les vérifications sur les entrées pour commencer :

```python
class Parking:
    def __init__(self,adr, cap , nbPortail):
        assert isinstance(adr, str) and len(adr) >0
        assert isinstance(capp,int) and cap >0
        assert isinstance(nbPortail,int) and nbPortail >0
```

```python
        self.adr=adr
        self.cap=cap
        self.nbPlacesOccupe=0
```