Rapport

Ayoub Faradi

Abdelatif elhamaoui

SOMMAIRE

Tp n°1 :

- ***Exercice 4***

Tp n°2 :

- ***Exercice 1***
- ***Exercice 2***
- ***Exercice 3***
- ***Exercice 4***

Tp n°3 :

- ***Exercice 1***
- ***Exercice 2***

Tp n°4 :

- ***Exercice 1***
- ***Exercice 2***

Tp n°1

Exercice 1: Compréhension des Composantes d'une Classe

- **Définition d'une Classe et Attributs :**

• Créez une classe Personne avec les attributs nom et age.

• Ajoutez un constructeur pour initialiser ces attributs.

• Implémentez une méthode pour afficher les détails de la personne.

2. Types de Données et Attributs de Classe :

• Ajoutez un attribut de classe population pour suivre le nombre total d'instances de la classe Personne.

• Mettez à jour le constructeur pour incrémenter le compteur de population à chaque nouvelle instance. • Implémentez une méthode de classe pour afficher le nombre total de personnes

Correction

Conception

Personne
nom : chaine âge : entier salaire : entier population : entier
Affichere_detail() Note_total()

CODE PYTHON

L’affichage

```
nom:Ali,age:20,salaire5000
nom:Ahmed,age:23,salaire6000
2
```

Exercice 2:

Encapsulation et Modificateurs

- **Principe de l'Encapsulation :**

• Ajoutez un attribut privé « **__salaire** » à la classe Personne.

• Implémentez des méthodes getters et setters pour accéder et modifier cet attribut.

- **Modificateurs et Accesseurs :**

• Ajoutez une méthode « **Augmenter_salaire** » qui permet d'augmenter le salaire d'une personne de manière sécurisée.

Correction

Conception

Personne
nom : chaine **âge : entier** **(-) salaire : entier**

population : entier
Get_salaire() Set_salaire() Augmenter_salaire()

CODE PYTHON

```
class Personne:
    population=0
    def __init__(self, nom, age, salaire):
        self.nom = nom
        self.age = age
        self.__salaire = salaire
        Personne.population+=1

    def afficher(self):
        print(f'Nom:{self.nom},Age:{self.age}
              ')
```

```
    @classmethod
    def nbr_personnes(cls):
        print(f'Nomber totale de personnes
                :{cls.population}')

    def _getters(self):
        print(f"Salaire de {self.nom} est
                {self.__salaire} DH")
    def _setters(self,n):
        self.__salaire = int(n)

    def Augmenter_salaire(self,n):
        self.__salaire = self.__salaire +
       (self.__salaire*(n/100))

p1 = Personne("Ali",19,3000)
p2 = Personne("Ayoub",20,4000)
p1.Augmenter_salaire(10)
p1._getters()
p2._setters(6000)
p2.Augmenter_salaire(15)
p2._getters()
```

L’affichage

```
Salaire de Ali est 3300.0 DH
Salaire de Ayoub est 6900.0 DH
```

Exercice 3:

Définition et Destruction d'un Objet

1.Définition d'un Objet et Instanciation :

- Créez une nouvelle classe Cours avec des attributs tels que nom, **professeur** (instance de Personne), et **étudiants** (une liste d'instances de Personne).
- Ajoutez un constructeur pour initialiser ces attributs.

2.Destruction Explicite d'un Objet :

• Ajoutez une méthode **« quitter_cours »** à la classe Personne qui permet à un étudiant de quitter un cours.

Correction

Conception

Personne
nom : chaine **âge : entier** **(-) salaire : entier** **population : entier**
Get_salaire() **Set_salaire()** **Augmenter_salaire()**

Cours
nom : chaine professeur : personne étudiant : liste de personne
Quitter_cours()

CODE PYTHON

```
class Cours(Personne):
    def __init__(self,nom, professeur,
etudiants = None):
        self.nom = nom
        self.professeur = professeur
        self.etudiants = etudiants if not
                                None else []
```

```
    def quitter_cours(self,etudiant):
        self.etudiants.remove(etudiant)
```

```
prefesseur = Personne("Ahmed",28,9000)

cour1 = Cours("java script", prefesseur, [p1,p2])
cour2 = Cours("Python", prefesseur, [p1,p2,p3,p4])
cour3 = Cours("HTML & CSS", prefesseur, [p1,p2,p3])

cour2.quitter_cours(p1)
```

Exercice 4: Méthodes et Visibilité

1.Définition d'une Méthode et Visibilité :

- Ajoutez une méthode **annoncer_cours** à la classe Cours qui affiche les détails du cours, du professeur, et des étudiants.

2.Paramètres d'une Méthode et Méthodes de Classe :

- Modifiez la méthode **annoncer_cours** pour accepter un paramètre facultatif permettant de filtrer les étudiants par âge.
- Ajoutez une méthode de classe **cours_populaire** à la classe Cours qui renvoie le cours avec le plus d'étudiants.

3.Fonctions Imbriquées :

- Ajoutez une fonction imbriquée à la méthode **annoncer_cours** qui affiche des informations spécifiques lorsque le cours est populaire (par exemple, plus de 10 étudiants).

Correction

Conception

Personne

nom : chaine âge : entier (-) salaire : entier population : entier
Get_salaire() Set_salaire() Augmenter_salaire()

Cours

nom : chaine professeur : personne étudiant : liste de personne
Quitter_cours() Annoncer_cours() Cours_populaire()

CODE PYTHON

```
class Cours(Personne):
    nbr_etudiant = 0
    population = ""
```

```
    def
__init__(self,nom,professeur,etudiants=None):
        self.nom = nom
        self.professeur = professeur
        self.etudiants = etudiants if not
                            None else []
        if self.etudiants.__len__() >
            Cours.nbr_etudiant:
             Cours.nbr_etudiant =
             self.etudiants.__len__()
             Cours.population = self.nom

    def quitter_cours(self,etudiant):
        self.etudiants.remove(etudiant)

    def annocer_cours(self,etudiantAge=19):
        print(f"Cours : {self.nom}")
        print("Proffesseur :")
        self.professeur.afficher()
        print("Etudiants : ")
        for etudiant in self.etudiants:
            if etudiant.age == etudiantAge:
                etudiant.afficher()
        def imbriquée():
            if self.etudiants.__len__() > 10:
                print(f"Plus de 10
```

```
                        etudiants")
            elif self.etudiants.__len__() <
                    10:
                print(f"Moin de 10
                        etudiants")
            else:
                print(f"10 etudiants")
            moyenneAge = 0
            for ageEtud in self.etudiants:
                moyenneAge += ageEtud.age
            moyenneAge /=
            self.etudiants.__len__()
            print(f"Moyenne d'age des
                    etudiants est {moyenneAge}")
        imbriquée()
    @classmethod
    def cours_populaire(cls):
        print(f"Cours avec le plus
             d'étudiants est
             {Cours.population} de
             {Cours.nbr_etudiant} étudiants")

prefesseur = Personne("Ahmed",28,9000)
cour1 = Cours("java script", prefesseur,
                [p1,p2])
cour2 = Cours("Python", prefesseur,
```

```
                [p1,p2,p3,p4])
cour3 = Cours("HTML & CSS", prefesseur,
                [p1,p2,p3])
cour2.quitter_cours(p1)
cour1.annocer_cours(20)
Cours.cours_populaire()
cour2.annocer_cours()
```

L’affichage

```
Cours : java script
Proffesseur :
Nom: Ahmed, Age: 28
Etudiants :
Nom: Ayoub, Age: 20
Moin de 10 etudiants
Moyenne d'age des etudiants est
19.5
Cours avec le plus d'etudiants est
Python de 4 etudiants
Cours : Python
```

```
Proffesseur :
Nom: Ahmed, Age: 28
Etudiants :
Moin de 10 etudiants
Moyenne d'age des etudiants est
19.666666666666668
```

Tp n°2

Exercice 1:

Modélisation d'une Classe

1. Définissez une classe ***Produit*** avec les attributs **nom**, **prix**, et **quantite_stock**.

2. Implémentez un constructeur pour initialiser ces attributs lors de la création d'une instance.

3. Définissez une méthode **afficher_details** pour afficher les détails du produit.

Correction

Conception

Produit
nom : chaine **prix : réel** **quantite_stock :**

entier

Afficher_details()

CODE PYTHON

```
class Produit:

    def __init__(self, nom, prix,
quantite_stack):
        self.nom = nom
        self.prix = prix
        self.quantite = quantite_stack

    def afficher_details(self):
        print(f"Nom : {self.nom}, Prix :
{self.prix}, Quantité Stock : {self.quantite}")


p1 = Produit("pc",5000,12)
p1.afficher_details()
```

L’affichage

```
Nom : pc, Prix : 5000,
Quantite Stock : 12
```

Exercice 2:

Principe de l'Encapsulation

1. Encapsulez les attributs de la classe **Produit** en les définissant comme des **attributs privés**.

2. Implémentez des méthodes **getters** et **setters** pour accéder et modifier les attributs de manière sécurisée.

Correction

Conception

Produit
(-) nom : chaine (-) prix : réel (-) quantite_stock : entier
Afficher_details() get_nom(),set_nom()

get_prix(),set_prix()
get_quantite(),set_quantite()
SortList()
printStar()

CODE PYTHON

```
class Produit:
    def __init__(self, nom, prix,
                 quantite_stack):
        self.__nom = nom
        self.__prix = prix
        self.__quantite = quantite_stack

    def afficher_details(self):
        print(f"Nom : {self.__nom}, Prix :
            {self.__prix}, Quantité Stock :
            {self.__quantite}")
```

```
    def get_nom(self):
        return self.__nom
    def get_prix(self):
        return self.__prix
    def get_quantité(self):
        return self.__quantite
    
    def set_nom(self,nom):
        self.__nom = nom
    def set_prix(self,prix):
        self.__prix = prix
    def set_nom(self,nom):
        self.__nom = nom

    @staticmethod
    def sortList(list):
        list_dec = []
        while list != []:
            max = list[0]
            i = 1
            for i in range(len(list)):
                if list[i]>max:
                    max = list[i]
            list_dec.append(max)
            list.remove(max)
```

```
        return list_dec

    @staticmethod
    def printStar(num):
        i=1
        j=1
        for i in range(num):
            for j in range(i):
                print('*',end='')
            print('\n')

import random as r

l = r.sample(range(0,100),10)
p2 = Produit("pc",5000,12)
p2.set_nom('labtop')
print(p2.get_nom())

l2 = Produit.sortList(l)
print(l2)

star1 = Produit.printStar(6)
```

L’affichage

```
labtop
[95, 85, 82, 69, 66, 60, 46, 34, 9, 1]

*

**

***

****

*****
```

Exercice 3:

Principe de l'Héritage

1. Définissez une classe **Aliment** qui hérite de la classe Produit.

2. Ajoutez un attribut **date_peremption** à la classe Aliment.

3. Implémentez un constructeur pour initialiser cet attribut lors de la création d'une instance.

Correction

Conception

Produit
(#) nom : chaine (#) prix : réel (#) quantite_stock : entier
Afficher_details()

Aliment
(#) date_permption : date
Afficher_details() Nbr_aliment()

CODE PYTHON

```
class Produit:
    def __init__(self, nom, prix,
                 quantite_stack):
        self._nom = nom
        self._prix = prix
        self._quantite =
        quantite_stack

    def afficher_details(self):
        print(f"Nom : {self._nom},
         Prix : {self._prix},
         Quantité Stock :
         {self._quantite}")
```

```
    def
calculer_prix_total(self,quantité):
        return f"Prix Total
                {self._prix * quantité}"

class Aliment(Produit):
    population  = 0
    def __init__(self, nom, prix,
     quantite_stack, date_peremption):
        super().__init__(nom, prix,
                         quantite_stack)
        self._date = date_peremption
        Aliment.population+=1

    def afficher_details(self):
        print(f"Nom : {self._nom},
        Prix : {self._prix}, Quantité
        Stock : {self._quantite}, Date
        de Peremption {self._date}")
    @classmethod
```

```
    def nbr_aliment(cls):
        print(f'Nomber totale de
        aliments : {cls.population}')

a1 = Aliment("pc",5000,200,"20-02-
                2024")
a1.afficher_details()
a2 = Aliment("mobile",2100,400,"04-
    01-2024")
a2.afficher_details()
Aliment.nbr_aliment()
```

L’affichage

```
Nom : pc, Prix : 5000,
Quantite Stock : 200, Date de
Peremption 20-02-2024

Nom : mobile, Prix : 2100,
Quantite Stock : 400, Date de
Peremption 04-01-2024

Nomber totale de aliments : 2
```

Exercice 4:

Principe du Polymorphisme

1. Ajoutez une méthode **calculer_prix_total** à la classe Produit qui prend la quantité en paramètre et renvoie le prix total.

2. Redéfinissez cette méthode dans la classe Aliment pour appliquer une réduction si la date de péremption est proche.

Correction

Conception

Produit
(#) nom : chaine (#) prix : réel (#) quantite_stock : entier
Afficher_details() Calculer_prix_total()
Aliment
(#) date_permption : date
Afficher_details() nbr_aliment() Calculer_prix_total()

CODE PYTHON

```
class Aliment(Produit):
    population  = 0
```

```
    def __init__(self, nom, prix,
    quantite_stack, date_peremption):
        super().__init__(nom, prix,
                         quantite_stack)
        self._date = date_peremption
        Aliment.population+=1

    def afficher_details(self):
        print(f"Nom : {self._nom},
        Prix : {self._prix}, Quantité
        Stock : {self._quantite}, Date
        de Peremption {self._date}")

    @classmethod
    def nbr_aliment(cls):
        print(f'Nomber totale de
        aliments : {cls.population}')

    def calculer_prix_total
            (self,quantité,date_per):
        date_p = []
```

```
        date_p = date_per.split('-')
        date_x = []
        date_x = self._date.split('-')
        if int(date_x[0])-1 ==
            int(date_p[0]):
            return(self._prix*quantité)
             -(self._prix*quantité*0.1)
        Elif int(date_x[0])-2 ==
              int(date_p[0]):
            return(self._prix*quantité)
            -(self._prix*quantité*0.05)
        else:
            return(self._prix*quantité)

p3 = Aliment("book", 20,200,'2024-02-
                                  12')
p4 = Aliment("clavier", 100,30,"2024-
                                 01-14")

Aliment.nbr_aliment()
```

```
print(p3.calculer_prix_total(10,'2023
                             -05-15'))
```

L’affichage

```
Nomber totale de
aliments : 2
```

```
180.0
```

Tp n°1

Exercice 1:

Polymorphisme et Redéfinition des Méthodes

1. Définissez une classe abstraite **Vehicule** avec une méthode abstraite ***deplacer***().

2. Créez des classes dérivées **Voiture** et **Avion** qui héritent de **Vehicule**.

3. Redéfinissez la méthode ***deplacer***() dans chaque classe dérivée pour afficher un message spécifique à chaque type de véhicule.

4. Créez une méthode statique ***prime***() qui prends en paramêtre un nombre

Correction

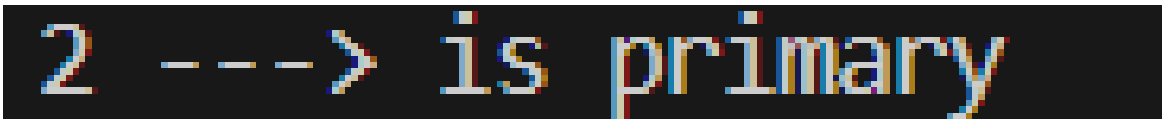
Conception

Vehicule
deplacer() **prime()**

CODE PYTHON

```python
from abc import ABC , abstractmethod
from math import pi

class vehicule(ABC):
    @abstractmethod
    def deplacer(self):
        pass

class voiture(vehicule):
    def deplacer(self):
        print("est une voiture")

    @staticmethod
    def prime(n):
        if n>=0 :
            i = 2
            b = True
            while b:
                if n%i ==0:
                    b = False
                    return f'{n} ---> is primary'
                else:
                    i = i + 1
                    if i == n//2:
                        b = False
                        return f'{n} ---> is primary'

class avion(vehicule):
    def deplacer(self):
        print("est une avion")

objv = voiture()
print(objv.prime(2))
```

L’affichage



Exercice 2: Surcharge des Opérateurs et Utilisation des Méthodes Abstraites

1. Définissez une classe abstraite **Forme** avec une méthode abstraite ***calculer_surface() .***

2. Créez des classes dérivées **Rectangle** et **Cercle** qui héritent de **Forme**.

3. Redéfinissez la méthode ***calculer_surface()*** dans chaque classe dérivée pour calculer la surface spécifique à chaque forme.

4. Créez une méthode statique ***somme_aires()*** dans chaque class dérivée que renvoie la somme des surfaces de 2 formes.

Correction

Conception

Form
calculer_surface()

CODE PYTHON

```python
class forme(ABC):
    @abstractmethod
    def calculer_surface(self):
        pass
class rectangle(forme):
    def __init__(self,langeur,largeur):
        self.langeur = langeur
        self.largeur = largeur

    def calculer_surface(self):
        return self.largeur * self.largeur

    @staticmethod
    def sommes_aires(forme1,forme2):
        return forme1.calculer_surface() + forme2.calculer_surface()


class Cercle(forme):
    def __init__(self,rayon):
        self.rayon = rayon

    def calculer_surface(self):
        return self.rayon**2 * pi
    @staticmethod
    def sommes_aires(forme1,forme2):
        return forme1.calculer_surface() + forme2.calculer_surface()
R = rectangle(6,4)
C = Cercle(3)
print(C.calculer_surface())
print(R.calculer_surface())
print(Cercle.sommes_aires(R,C))
```

L’affichage

```
28.274333882308138
16
44.27433388230814
```