DÉCOUVREZ
LES FORMATIONS
DE L'OFPPT

OFPPT

FILIÈRES
CONDITIONS D'ACCÈS
PROGRAMME
DIPLÔMES

Rapport des exercices de programmation orienté objet (python)

❖ ***Realise par :***

✓ *Ilham barakat*

✓ *Aya taki*

Annee scolaire ***2023 2024***

Formateur *othmane haqay*

Filiere *devloppement digital 104*

Centre de formation *ntic 2 sidi maarouf*

OFPPT

INTRODUCTION :

- La programmation orientée objet est l'une des méthodologies récentes de programmation, couramment utilisée par les langages de programmation les plus répandus (python, JavaScript, C#.Net, ...). Cette méthodologie succède à la programmation impérative en lui ajoutant les notions d'objets et de classes.

L'orientation objet présente de nombreux avantages, sinon elle ne serait pas le paradigme dominant du développement logiciel moderne. L'un des principaux avantages est l'encapsulation de la logique et des données dans des classes individuelles. Cela améliore la maintenabilité et rend le code plus facile à étendre.

Plan : *TP1*

TP2

TP3

Exercice 1: polymorphisme et redéfinition des Méthodes

Exercice 2: Surcharge des opérateurs et utilisation des méthodes abstraites

TP1

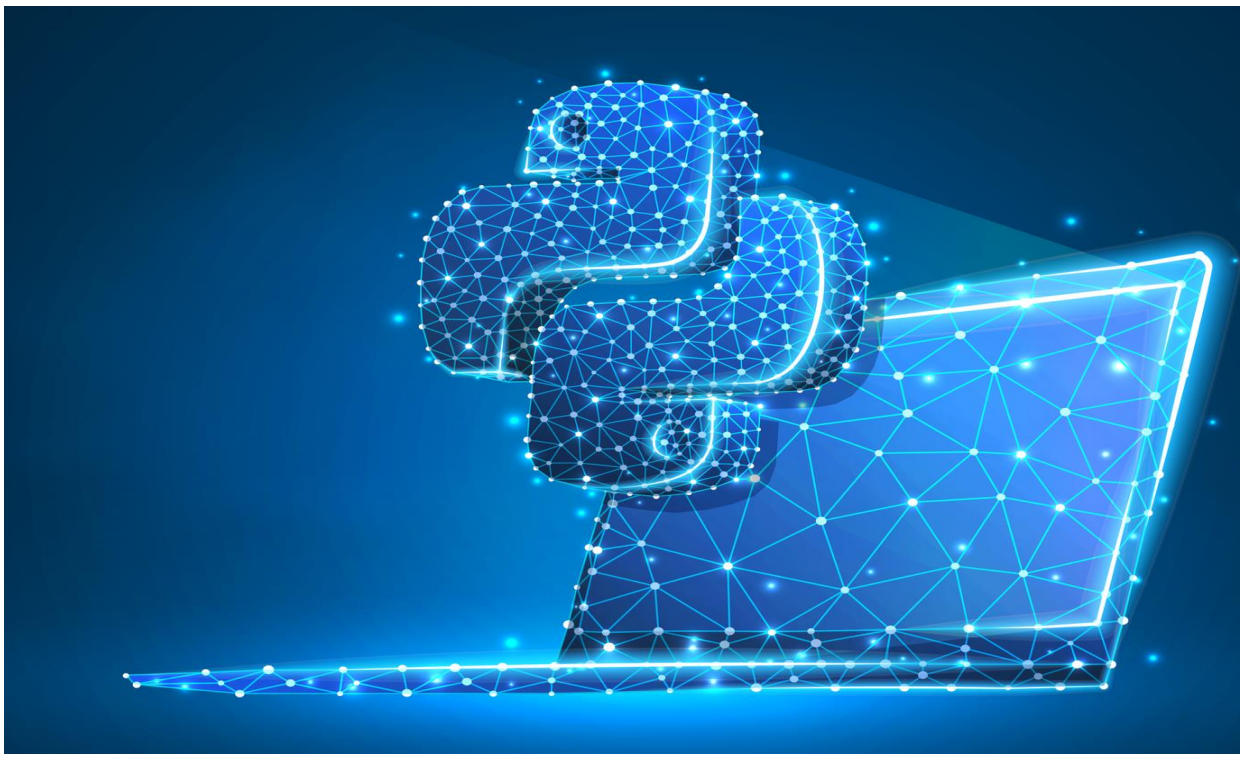
Exercice 1: Compréhension des Composantes d'une Classe

A .Énonce

1. ***Définition d'une Classe et Attributs :***
   - Créez une classe Personne avec les attributs nom et âge.
   - 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.

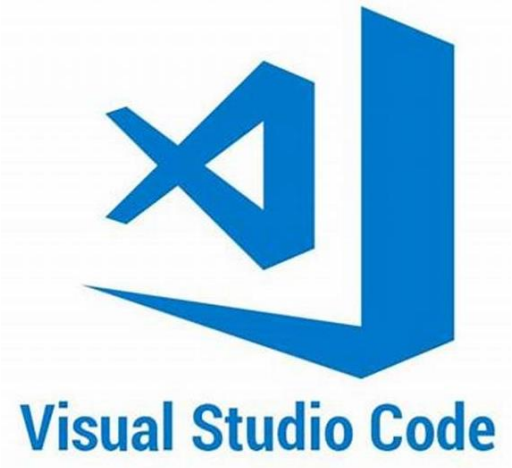
B.CONCEPTION

Personne
Nom : chaine de caractère Age : entier Salaire : entier Population : entier
Afficher_détails() Note_totale()



C.solution

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

    population = 0
    def __init__(self, nom, age,salaire ):
        self.nom = nom
        self.age = age
        self.salaire = salaire

        Personne.population += 1

    def afficher_details(self):
        print(f"nom: {self.nom}, Age: {self.age} ")
    @classmethod
    def nbr_personne (cls):
        print (f"nombre des personnes est:{cls.population}")
pr1=Personne(aya,20,35000)
pr2=Personne(adam,33,7800)
pr1.afficher_details()
pr2.afficher_details()
```



Exercice 2: Encapsulation et Modificateurs

A .Énonce

1-Principe de l'Encapsulation :

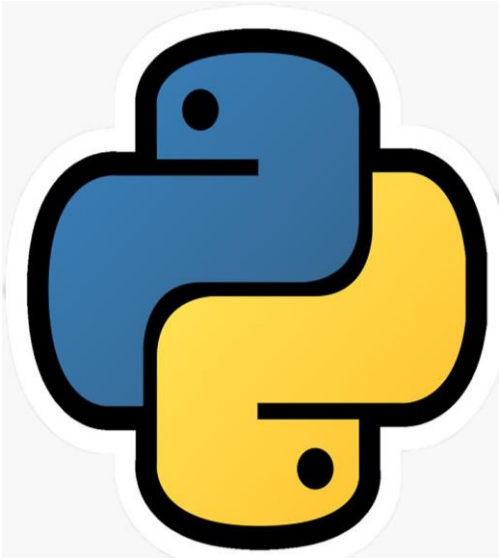
• Ajoutez un attribut privé « __salaire » à la classe Personne.

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

2-Modificateurs etAccesseurs :

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

B.CONCEPTION



Personne
Nom :chaine de caractère Age :entier (-)Salaire :entier Population :entier
GET-salaire() SET-salaire() Augmenter-salaire()

C.solution

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

    def afficher_details(self):
        print(f"nom: {self.nom}, Age: {self.age}, salaire: {self.__salaire}")

    def get_salaire(self):
        return self.__salaire

    def set_salaire(self, salaire):
        if salaire >= 0:
            self.__salaire = salaire
        else:
            print("Invalid salaire")

    @classmethod
    def display_population(cls):
        print(f"Total population: {cls.population}")

    def aug_salaire(self, amount):
        self.__salaire += amount
```

Exercice 3: Définition et Destruction d'un Objet

A .Énonce

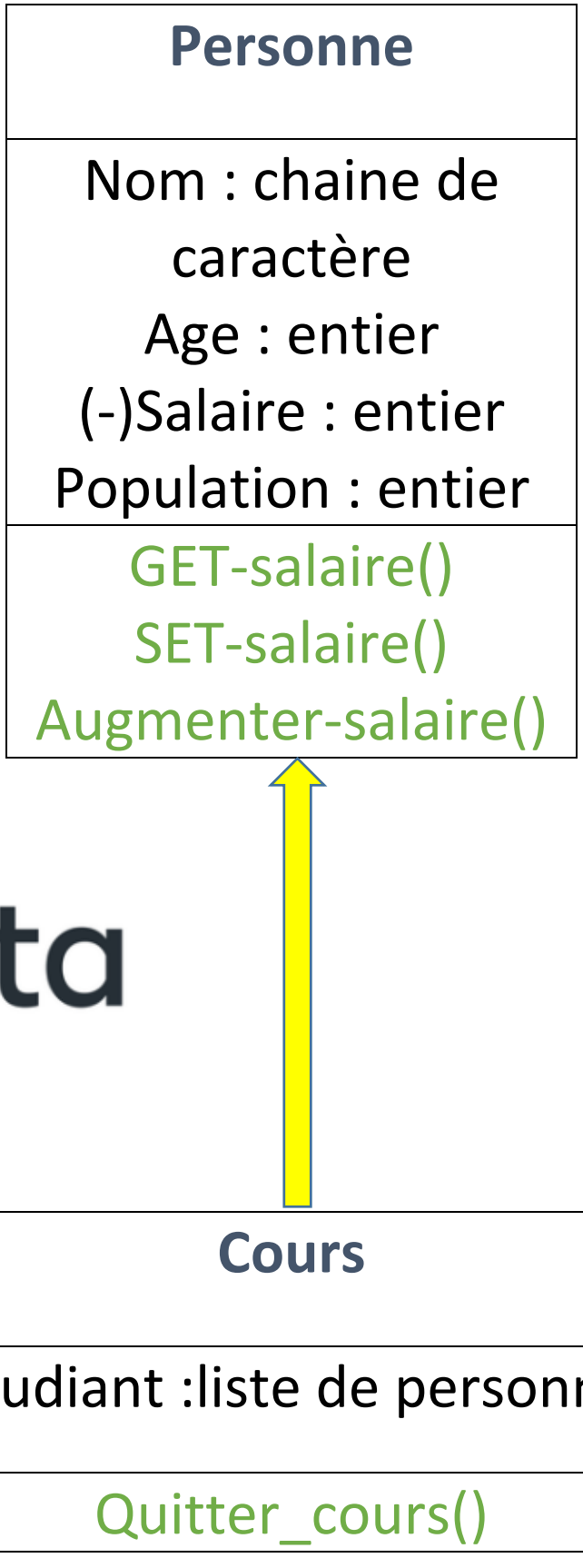
1. Définition d'un Objet et Instanciation :

- 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.

B.CONCEPTION



C.solution

```
class Cours:

    def __init__(self, nom, professeur, etudiant):

        self.nom = nom

        self.professeur = professeur

        self.etudiant = etudiant if not none else[]

    def quitter_cours(self, etudiant):

        self.etudiant.remove(etudiant)
```

Exercice 4: Méthodes et Visibilité

A .Énonce

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).

B.CONCEPTION

Cours
Nom : chaine de caractère personne: entier étudiant :liste de personne
Quitter_cours() Annoncer_cours() Cours_population()

C.solution

```python
class Cours:
    def __init__(self, nom, professeur, etudiant):
        self.nom = nom
        self.professeur = professeur
        self.etudiant = etudiant
    def announce_cours(self):
        print(f"Cours: {self.nom}, professeur: {self.professeur.nom}, etudiant:
{[etudiant.nom for etudiant in self.etudiant]}")
    def quitter_cours(self, etudiant):
        self.etudiant.remove(etudiant)
    def announce_cours(self, age_filter=None):
        print(f"Cours: {self.nom}, professeur: {self.professeur.nom}")
        if age_filter is None:
            print("etudiant: {[etudiant.nom for etudiant in self.etudiant]}")
        else:
            print("etudiant (filtered by age): {[etudiant.nom for etudiant in
self.etudiant if etudiant.age > age_filter]}")
class Personne:
    population = 0
    def __init__(self, nom, age, salaire=0):
        self.nom = nom
        self.age = age
        self.__salaire = salaire
        Personne.population += 1
    def dafficher_details(self):
        print(f"nom: {self.nom}, Age: {self.age}, salaire: {self.__salaire}")
    def get_salaire(self):
        return self.__salaire
    def set_salaire(self, salaire):
        if salaire >= 0:
            self.__salaire = salaire
        else:
            print("Invalid salaire")
    @classmethod
    def display_population(cls):
        print(f"Total population: {cls.population}")
    def aug_salaire(self, amount):
        self.__salaire += amount

professeur = Personne(nom="professeur3", age='35', salaire='10000')

etudiant1 = Personne(nom = "hafsa", age = 18, salaire = 8000)
cours = Cours(nom="POO", professeur="professeur1", etudiant="etudiant2")
```

TP2

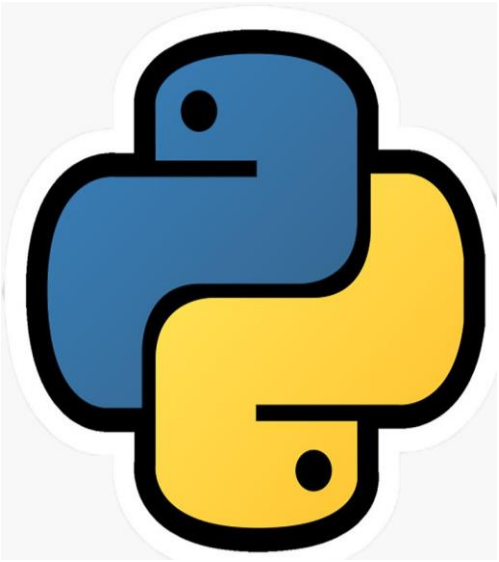
Exercice 1: Modélisation d'une Classe

A .Énonce

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.

B.CONCEPTION

Produit
Nom : chaine Prix : réel Quantite_stock : entier
afficher_details



C.solution

```
class produits :
    def _init_(self,nom,prix,quantite_stock):
        self.nom=nom
        self.prix=prix
        self.quantite_stock=quantite_stock
    def afficher_details(self):
        print(f"le nom du produits:{self.nom},le prix est:{self.prix},la quantité
disponible:{self.quantite_stock}")
P1=produits("majix", 5, 150)
P2=produits("maped",20,1)
print(P1.afficher_details()+P2.afficher_details())
```

Exercice 2: Principe de l'Encapsulation

A .Énonce

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.

B.CONCEPTION

Produit
(-)Nom : chaine (-)Prix :réel (-) Quantite_stock : entier
afficher_details get_nom(), set_nom(), get_ Prix (), set_ Prix (), get_ Quantite_stock (), set_ Quantite_stock (), sortList(), print(L2)

C.solution

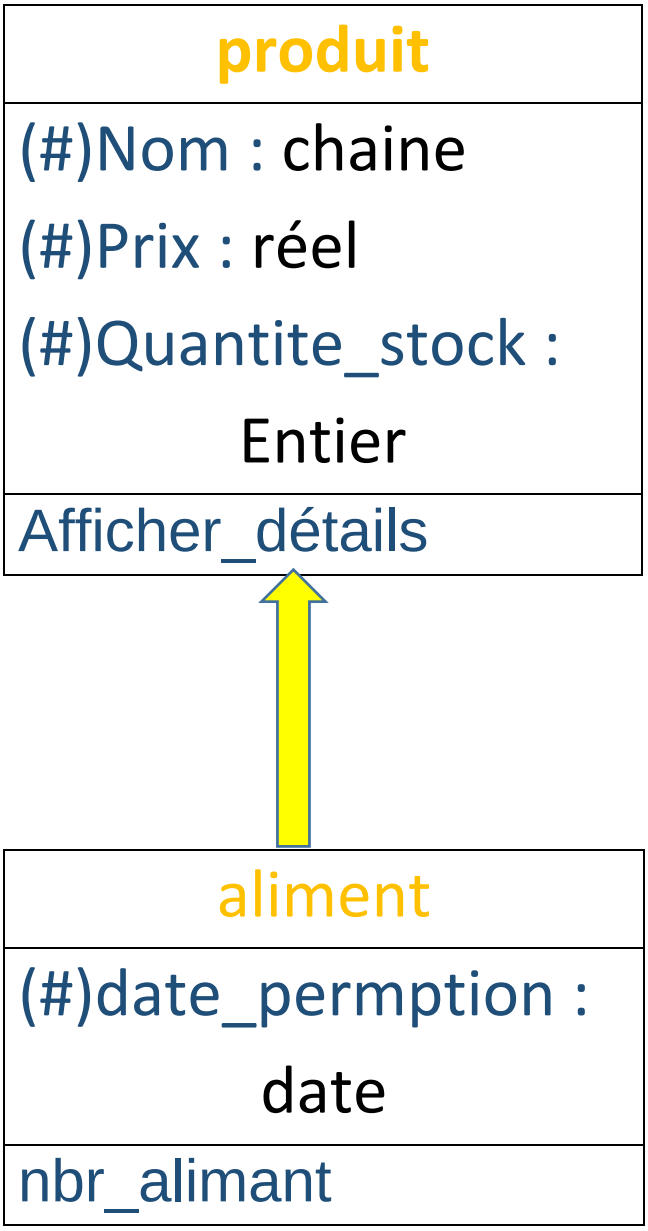
```
class produits :
    def _init_(self,nom,prix,quantite_stock):
        self.__nom=nom
        self.__prix=prix
        self.__quantite_stock = quantite_stock
    def get_nom(self):
        return self.__nom
    def set_nom(self,nv_nom):
         self.nom  = nv_nom
     def get_prix(self):
        return self.__prix
    def set_prix(self,nv_prix):
        self.prix= nv_prix
    def get_quantite_stock(self):
        return self.__quantite_stock
    def set_quantite_stock(self,nv_quantite_stock):
         self.__quantite_stock = nv_quantite_stock
    def afficher_details(self):
        print(f"le nom du produits:{self.nom},le prix est:{self.prix},
              la quantité disponible:{self.quantite_stock}")
p1=produits("majix", 5, 150)
print(p1.get_nom())
print(p1.get_prix())
print(p1.get_quantite_stock()) =
p1.setnom = "oni"
p1.setprix = 13
p1.setquantite_stock = 100
print(p1.get_nom())
print (p1.get__prix())
print(p1.get__quantite_stock())
p1.afficher_details()
@staticmethod
def sort (list):
    listdec = []
    while list !=[] :
     max = list[0]
    for i in range (len(liste)):
        if liste[i]>max:
            max=liste[i]
    listdec.append(max)
    list.remove(max)
    liste=listdec
    return liste
l1 = [28, 56, 98, 23]
l2=sort (l1)
print(l2)
```

Exercice 3: Principe de l'héritage

A .Énonce

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.

B.CONCEPTION



C.solution

```
class aliment(produits):
```

```
    count = 0

    def _init(self,nom,prix,quantite_stock
,date_pert):

     super().init(nom,prix,quantite_stock)

     self.date_pert = date_pert

     aliment.count += 1

@classmethod

def nbr_aliment(cls):
```

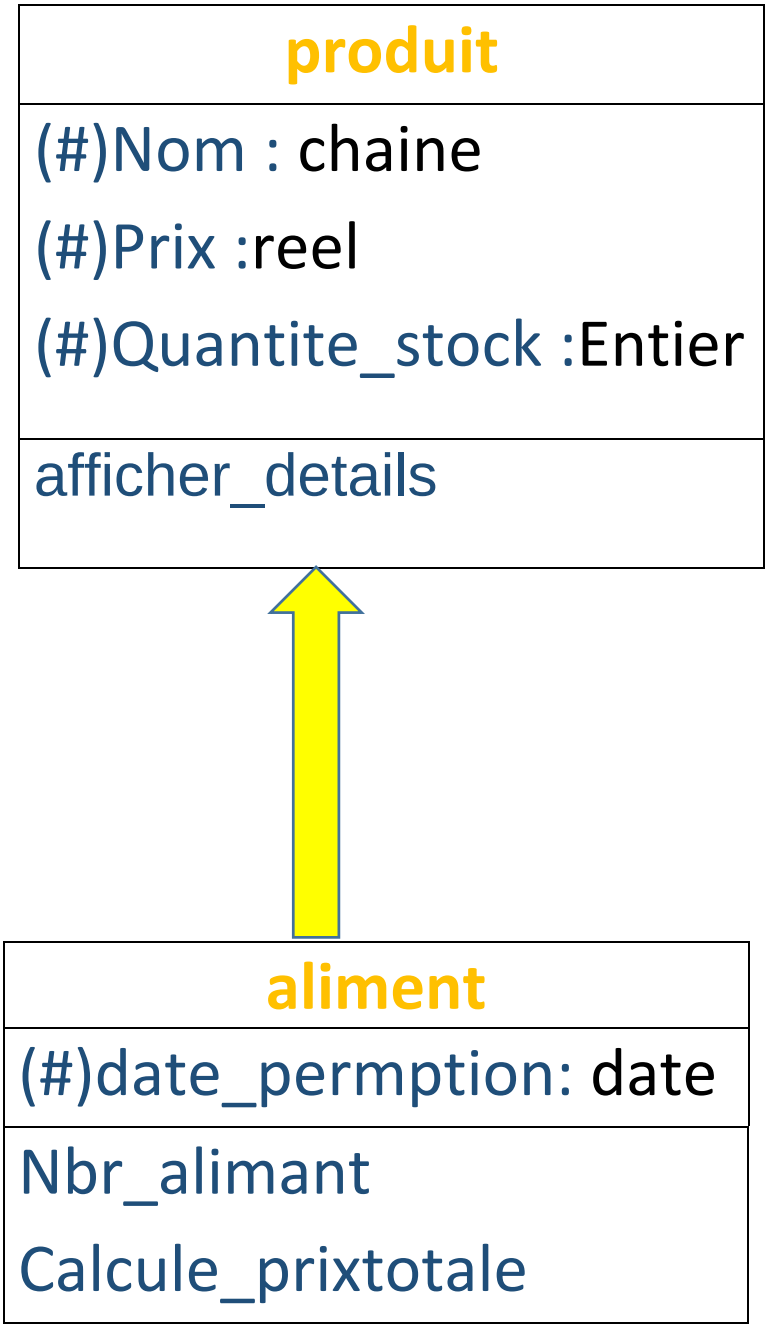
```
        print (f"nbr de totale {cls.count}")

p3 = aliment ("tomate",45,233,2025)

p5 = aliment ("serise",4,2345,2027)

print (aliment.nbr_aliment())
```

Exercice 4: Principe du polymorphisme

A .Énonce

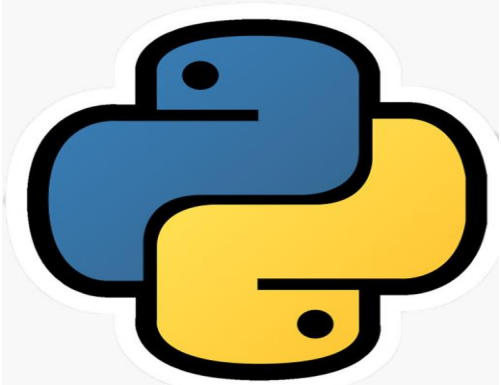
1. Ajoutez une méthode calculer_prix_total à la classe Produit qui prend la quantité en paramètre et renvoie le prix total.
2. Redéfinissez cette méthode dans la classe Aliment pour appliquer une Réduction si la date de péremption est proche.

B.CONCEPTION



C.solution

```python
#1
class produits :
    def _init_(self,nom,prix,quantite_stock):
        self.nom=nom
        self.prix=prix
        self.quantite_stock=quantite_stock
    def affichage (self):
        print(f"le nom du produits:{self.nom},le prix est:{self.prix},
              la quantité disponible:{self.quantite_stock}")
    def calculer_prixtotat(self,q):
        return self.prix * q
P1=produits("majix", 5, 150)
P2=produits("maped",20,13)
p1.affichage()
p1.calculer_prixtotat(20)
#2
class aliment(produits):
    def _init(self,nom,prix,quantite_stock ,date_pert):
         super().init(nom,prix,quantite_stock)
         self.date_pert = date_pert
    def calculer_prixtotat(self, q,date_act):
        if date_act == date_pert - 1:
            return((self.prix - self.prix * 0,5) * q )
all = aliment ("cookies",20,2900,"2024-03-28")
al1.calculer_prixtotat (45,"2023-03-20")
```



TP3

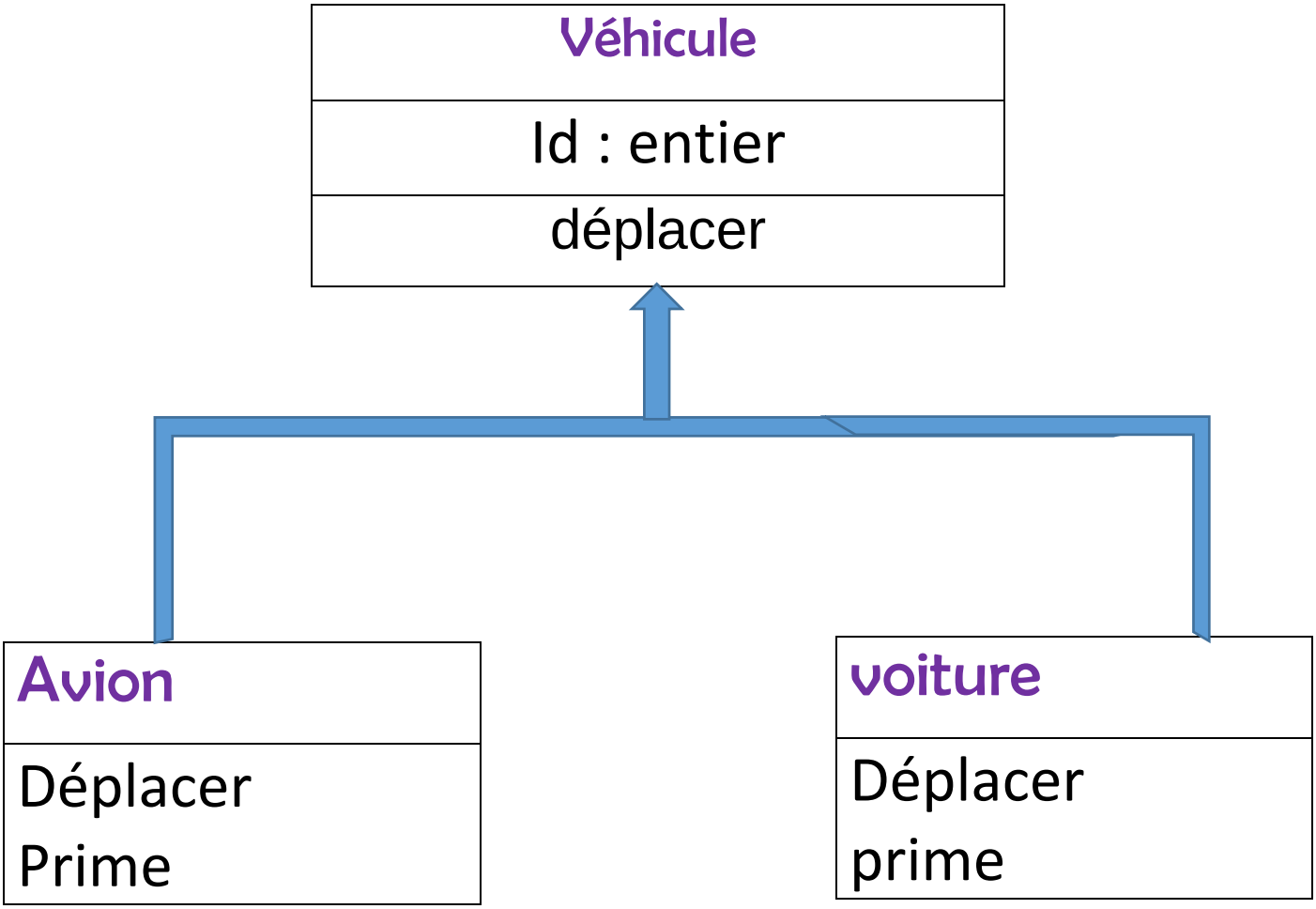
Exercice 1:

Polymorphisme et redéfinition des Méthodes

A .Énonce

1. Définissez une classe abstraite Véhicule avec une méthode abstraite déplacer ().
2. Créez des classes dérivées Voiture et Avion qui héritent de Véhicule.
3. Redéfinissez la méthode déplacer () dans chaque classe dérivée pour afficher un message spécifique à chaque type de véhicule.
4. Créez une méthode statique prime () qui prends en paramètre un nombre entier positif et qui détermine s'il s'agit d'un nombre premier ou non.

B.CONCEPTION



C.solution

```python
from abc import ABC, abstractmethod
class Vehicule(ABC):
    def __init__(self,id):
        self.id=id
    @abstractmethod
    def deplacer (self):
        pass
    @staticmethod
    def prime(n):
        if n>=0:
            i=2
            b = True
            while b:
                if n%i ==0:
                    b = False
                    return f'{n} is not primary'
                else:
                    i =i+1
                    if i ==n//2:
                        b = False
                        return f'{n} is primary'


class avion(Vehicule):
    def __init__(self,id):
        super().__init__(id)
    def deplacer (self):
      return ("hello, avion")

class voiture (Vehicule):
    def __init__(self,id):
        super().__init__(id)
    def deplacer (self):
      return("heyy, voiture")

v2=voiture(5)
print(v2.prime(13))
```

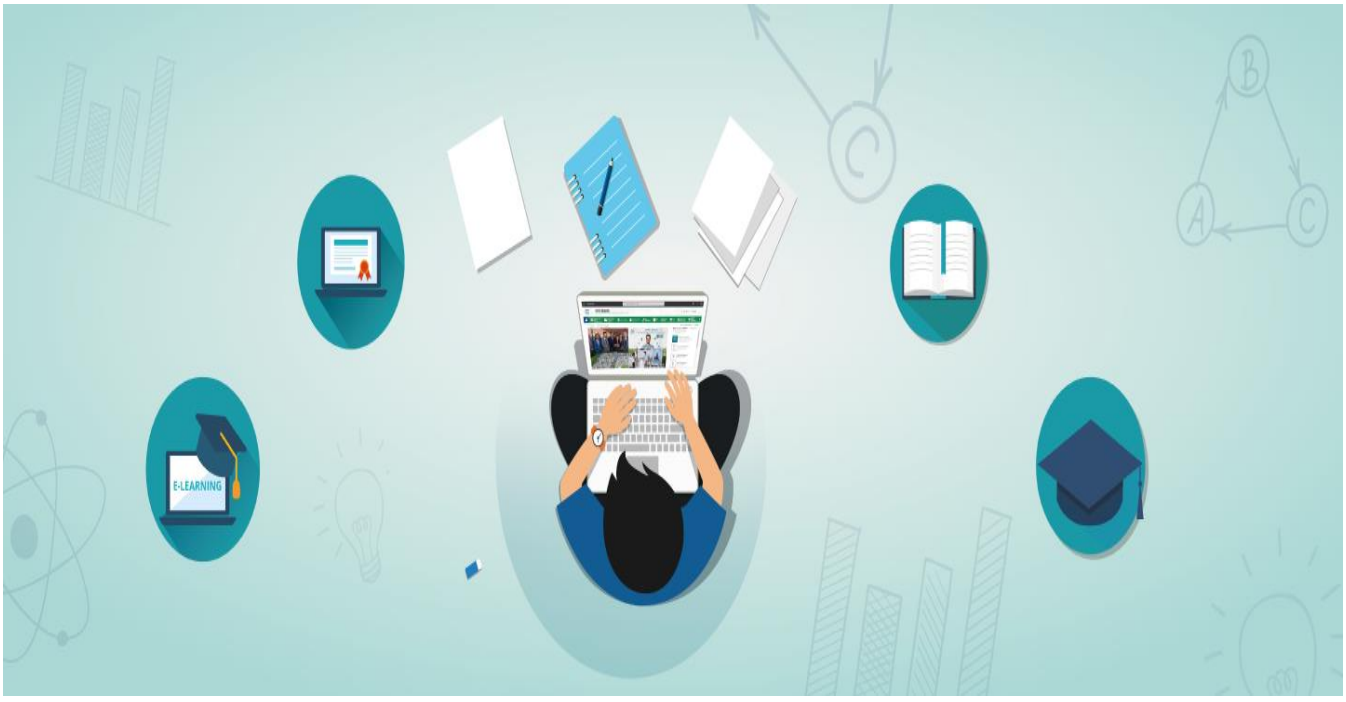
résultat sur Visual studio code

Exercice 2:

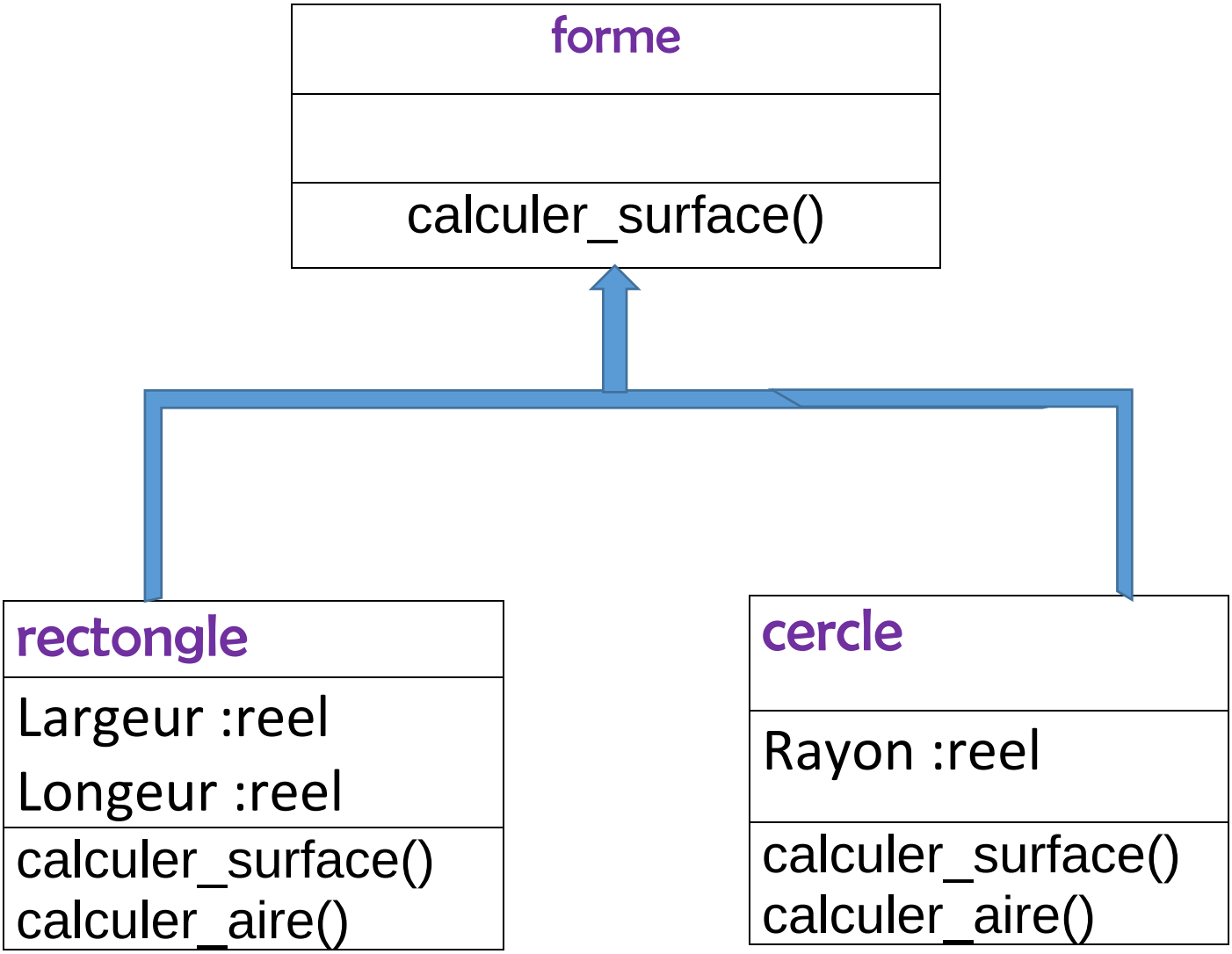
Polymorphisme et redéfinition des Méthodes

A .Énonce

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 classe dérivée qui renvoie la somme des surfaces de 2 formes.



B.CONCEPTION



C.solution

```
class Forme(ABC):
    @abstractmethod
    def calcule_surface (self):
        pass

class Rectongle(Forme):
    def __init__(self,id,lr,lg):
            super().__init__()
            self.largeur =lr
            self.longeur =lg
    def calcule_surface(self):
        return self.longeur*self.largeur

    def somme_aire(self,Forme1):
        return self.calcule_surface() + Forme1.calcule_surface()

class Cercle (Forme):
    def __init__(self,rayon,):
          super().__init__()
          self.rayon=rayon
    def calcule_surface(self):
           return math.pi * self.rayon**2

    def somme_aire(self,Forme1):
        return self.calcule_surface() + Forme1.calcule_surface()
    def __add__(self,o):
         return self. calculer_surface

ily=Cercle(4)
print (ily.calcule_surface())
aya=Cercle(5)
print(aya.calcule_surface())
print(Cercle.somme_aire(ily,aya))
```

résultat sur Visual studio code

```
TERMINAL ···                                Python + ··· ^ ×

ppData/Local/Programs/Python/Python312/python.exe "c:/Use
rs/DELL/Desktop/pratique html/poo/tp.py/tp3.py"
50.26548245743669
78.53981633974483
128.80529879718154
PS C:\Users\DELL\Desktop\pratique html>
```

merci

– BEACOUP –

POUR VOTRE ATTENTION