# La_POO

July 16, 2025

## 1 Introduction

## 2 Les classes

En Python, une classe est un modèle ou une structure qui permet de définir des objets. Elle regroupe des attributs (données) et des méthodes (fonctions) qui décrivent le comportement et les propriétés des objets créés à partir de cette classe.

- **Définition d'une classe :** Une classe est définie à l'aide du mot-clé **class**. Elle sert de plan pour créer des instances (ou objets), qui sont des entités concrètes basées sur cette classe.

- **Structure d'une classe :** Voici les éléments principaux d'une classe :

1. Attributs :

Ce sont des variables qui stockent les données ou les propriétés d'un objet. Ils peuvent être définis dans la méthode spéciale **init** (le constructeur) ou directement dans la classe.

2. Méthodes :

Ce sont des fonctions définies dans une classe qui décrivent les comportements ou actions que les objets peuvent effectuer. La méthode spéciale **init** est appelée automatiquement lors de la création d'un objet pour initialiser ses attributs.

3. Objet :

Une instance d'une classe. Chaque objet possède ses propres valeurs pour les attributs définis dans la classe.

### 2.0.1 Exemple d'une classe simple

[1]:
```python
# Création d'une classe Livre

class Livre:
    def __init__(self,Titre, Auteur, An_pub):
        self.titre=Titre
        self.author=Auteur
```

```
        self.year=An_pub
    def afficher(self):
        print(f"Le titre du livre est {self.titre} dont l'auteur est {self.
↪author}. Il est sorti en {self.year}.")
```

[2]:
```
# Instantation : creation d'objets de la classe Livre
# lv est le nom de l'objet

lv1 = Livre("Louca", "Bruno Dequier", 2014)
lv2 = Livre("La légende Final fansy XII","Rémi Lopez", 2024)

lv1.afficher()
lv2.afficher()
```

```
Le titre du livre est Louca dont l'auteur est Bruno Dequier. Il est sorti en
2014.
Le titre du livre est La légende Final fansy XII dont l'auteur est Rémi Lopez.
Il est sorti en 2024.
```

### 2.0.2 La classe Bachelor

[3]:
```python
# Création de la classe Bachelor

class Bachelor:
    #attributs
    def __init__(self,nom,age,note):
        self.__nom=nom
        self.age=age
        self.note=note

    #méthodes
    def afficher(self):
        print(f"Nom : {self.getnom()}")
        print(f"Age : {self.age}")
        print(f"Note : {self.note}")

    #fonction  "get" -> récupérer la valeur de l'attribut
    def getnom(self):
        return self.__nom

    #fonction "set" -> modifier la valeur de l'attribut
    def setnom(self,nom):
        self.__nom=nom


    def ajouter(self,points):
        print(f"Nouvelle note: {self.note + points}\n")
        self.note += points
```

[4]:
```python
# Instantation : creation d'un objet d'un classe
# Appel de la classe Bachelor() pour créer un objet etd1

bchlr1 = Bachelor("Zoléni", 20, 17)
bchlr1.setnom("Killian")
print (bchlr1.getnom())

bchlr1.ajouter(2)
bchlr1.afficher()
```

```
Killian
Nouvelle note: 19

Nom : Killian
Age : 20
Note : 19
```

### 2.0.3 Un rectangle

[5]:
```
# Création de la classe Rectangle
# La classe Rectangle est une classe qui permet de créer des objets de type␣
 ↪rectangle

class Rectangle:
    """Ceci est la classe rectangle"""
    def __init__(self,longueur=0.0, largeur=0.0,couleur="blanc"):
        """Initialisation d'un objet.

        Définition des attributs avec des valeur par défaut.
        """
        self.longueur=longueur
        self.largeur=largeur
        self.couleur=couleur

    def calcule_surface(self):
        """Méthode qui calcule la surface."""
        return   self.longueur*self.largeur

    def change_carre(self,cote):
        """Méthode transforme un rectangle en carré"""
        self.longueur=cote
        self.largeur=cote

    def ligne_vide(self):
        print(" " *20)
        """Méthode qui affiche une ligne vide"""

    def afficher(self):
        print (f"Longueur: {self.longueur}\nLargeur: {self.largeur}\nCouleur:␣
 ↪{self.couleur}\n"
               f"Surface: {self.calcule_surface()}")
```

[6]:
```
if __name__=="__main__":
    rectangle = Rectangle(2,3,"noir")
    rectangle.afficher()
    rectangle.ligne_vide()
    rectangle.change_carre(5)
    rectangle.afficher()
```

```
Longueur: 2
Largeur: 3
Couleur: noir
Surface: 6

Longueur: 5
```

```
Largeur: 5
Couleur: noir
Surface: 25
```

#### 2.0.4 Un dernier exemple pour la route

[7]:
```python
class Citron:
    couleur="jaune"
    def afficher(self):
        print(self.couleur)
    def recuperer_saveur(self):
        return "acide"

if __name__=="__main__":
    citron1 = Citron()
    citron1.afficher()
    saveur=citron1.recuperer_saveur()
    print(saveur)
```

```
jaune
acide
```

### 2.1 Le Polymorphisme

Le Polymorphisme permet de : Redéfinir des méthodes - Surchager des méthodes - Dans la même classe, j'ai 2 méthodes qui ont même nom mais avec des comp différents:

- même nom
- différence (nombre,type)

- Surcharger des opérateurs

**Exemple :**

[8]:
```python
class employé:
    def __init__(self, salaire):
        self.salaire=salaire

    def calcul_salaire(self, prime=None):  #"None" = soit la prime est là, soit␣
 ↪elle n'est pas là...
          if prime==None:
               print(f"Salaire: {self.salaire}")
          else:
            print(f"Salaire avec prime: {self.salaire+prime}")

    def __add__(self,e):
        return self.salaire+e.salaire
    def __mul__(self,e):
        return self.salaire*e.salaire
```

```
s1=employé(20)
s1.calcul_salaire(1000)

s2=employé(3000)

print(s1+s2)
print(s1*s2)
```

```
Salaire avec prime: 1020
3020
60000
```

## 2.2 Le principe d'hérédité

L'hérédité est un concept fondamental en programmation orientée objet (POO), y compris en Python.

Elle permet à une classe (appelée classe dérivée ou classe enfant) d'hériter des attributs et des méthodes d'une autre classe (appelée classe de base ou classe parent).

Cela favorise la réutilisation du code et facilite l'extension des fonctionnalités.

Principe de l'hérédité : - Une classe parent contient des attributs et des méthodes de base. - Une classe enfant hérite de ces attributs et méthodes, mais peut également : Ajouter de nouveaux attributs ou méthodes. Redéfinir (ou surcharger) les méthodes de la classe parent.

### 2.2.1 Exemple

[9]:
```
# Le principe d'héritage

class person:
    def __init__(self, nom, age):
        self.nom=nom
        self.age=age
    def afficher(self):
        print("Le nom :",self.nom)

 # Définir une sous-classe ou bien classe fille
class etudiant(person):
    def __init__(self, nom,age,note):
        person.__init__(self,nom,age)
        self.note=note
    def afficher(self):
         super().afficher()

 # Définir une sous-classe ou bien classe fille
class professeur(person):
    def __init__(self,nom,age,salaire):
        person.__init__(self,nom,age)
```

```
        self.salaire=salaire
    def afficher(self):
        super().afficher()

# Instanciation = création d'un objet à partir d'une classe
etd1=etudiant("Zoléni", 19,17)
etd1.afficher()
pr1=professeur("Xavier", 26, 1300)
pr1.afficher()

# print(f"nom : {etd1.nom}, age: {etd1.age}")
```

```
Le nom : Zoléni
Le nom : Xavier
```

La classe fille peut également hériter de deux classes mères.

### 2.2.2 Exercice effectué à l'ENSEA

[1]:
```
# Script python effectué à l'ENSEA (cf Classes_Exo_1.py)

from scripts.Classes_Exo_1 import professor
pr1=professor("DuChandelier","Xavier",2500,"médecin généraliste")
pr1.afficher()
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[1], line 3
      1 # Script python effectué à l'ENSEA (cf Classes_Exo_1.py)
----> 3 from scripts.Classes_Exo_1 import professor
      4 pr1=professor("DuChandelier","Xavier",2500,"médecin généraliste")
      5 pr1.afficher()

ModuleNotFoundError: No module named 'scripts.Classes_Exo_1'
```

## 2.3 3. Savoir lier les objets de classes différentes entre eux

On peut ainsi lier les objets entre eux et ils seront programmés selon leurs méthodes de leur **class** respective

Programme liant 2 classes

[ ]:
```
class Membre:
    """Les infos de chaque membre de la famille (nom, prénom, age,...)"""
    def __init__(self, prenom, nom):
        self.nom = nom
        self.prenom = prenom
```

```
    def afficher_infos(self):
        print(f"Prénom: {self.prenom}\nNom: {self.nom}\n\n")
```

[ ]:
```
class Famille:
    """Cette classe permet de créer une famille\n"""

    def __init__(self, nom):
        self.nom = nom
        self.famille = []

    def afficher(self):
        print(f"La Famille {self.nom}\n")

    def ajouter(self, membre):
        self.famille.append(membre)

    def afficher_famille(self):
        if not self.famille:
            print("La famille est vide.")
        else:
            for membre in self.famille:
                membre.afficher_infos()    # ----> Activation de la méthode␣
 ↪afficher_infos() programmé dans la classe Membre
```

[ ]:
```
membre_1 = Membre("Zoléni", "Kokolo Zassi")
membre_2 = Membre("Daphné", "Kokolo Zassi")
membre_3 = Membre("Johéda", "Kokolo Zassi")

famille = Famille("Kokolo Zassi")
famille.ajouter(membre_1)
famille.ajouter(membre_2)
famille.ajouter(membre_3)
famille.afficher()
famille.afficher_famille()
```

```
La Famille Kokolo Zassi

Prénom: Zoléni
Nom: Kokolo Zassi


Prénom: Daphné
Nom: Kokolo Zassi


Prénom: Johéda
Nom: Kokolo Zassi
```

# 3 L'encapsulation

- 3 types de visibilité :
  - Public = tt le monde peut y accéder
  - Protected = seul la class qui possède l'élément et la class fille peut y accéder
  - Private = seul la classe poosédant l'élément peut y accéder

[ ]:
```
# Revoir le concept car pas bien compris
```

# Les classes Abstraites

- Une classe abstraite est une classe possédant une méthode abstraite.
- La spécificité est qu'on ne peut pas créer d'objets avec une méthode abstraite.

### 3.0.1 Exemple 1: les Véhicules

[ ]:
```
# Les classes Abstraites

from abc import ABC, abstractmethod

# Création  d'une classe abstraite
class Vehicule(ABC):
    def __init__(self, vitesse):
        self.vitesse=vitesse

    @abstractmethod        # ----> Argument permettant d'indiquer à Python que
 ↪c'est une classe abstraite
    def printInfo(self):
        pass

# Création d'une classe héritant d'une classe mère
class moto(Vehicule):
    def __init__(self, vitesse):
        Vehicule.__init__(self,vitesse)
    def printInfo(self):
        print ("Im a moto with speed", self.vitesse)
    def augmanter_vitesse(self,x):
        self.vitesse+=x
```

[ ]:
```
roues_2 = moto(20)
roues_2.printInfo()
roues_2.augmanter_vitesse(20)
roues_2.printInfo()
```

```
Im a moto with speed 20
Im a moto with speed 40
```

[ ]:
```
# On ne peut pas créer des objets à partir de classes abstraites
# Faisons le test...

ve=Vehicule(20) # ----> Objet venant de la classe Véhicule
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[17], line 4
      1 # On ne peut pas créer des objets à partir de classes abstraites
      2 # Faisons le test…
----> 4 ve=Vehicule(20) # ----> Objet venant de la classe Véhicule

TypeError: Can't instantiate abstract class Vehicule without an implementation␣
 ↪for abstract method 'printInfo'
```

**Exemple 2: le système de paiement**

[ ]:
```
# Système de paiement

from abc import ABC, abstractmethod

# Création de la classe abstraite ModePaiement
class ModePaiement(ABC):
    def __init__(self,montant):
        self.montant=montant
    @abstractmethod
    def effectuer_paiement():
        pass


# Création de la classe Cartebancaire
# 1er mode de paiement
class Cartebancaire(ModePaiement):
    def __init__(self, montant,num):
        ModePaiement.__init__(self,montant)
        self.num=num
    def effectuer_paiement(self):
        self.num=input("Mettez votre numéro de carte [16 chiffres]: ")
        if len(self.num)==16:
            print(f"Paiement effectué avec la carte bancaire avec un montant de␣
 ↪{self.montant} €.")
        else:
            print("Ce n'est pas le numéro d'une carte bancaire. Echec de␣
 ↪l'opération.")


# Création de la classe PayPal
```

```python
# 2ème mode de paiement
class PayPal(ModePaiement):
    def __init__(self, montant,adress):
        ModePaiement.__init__(self,montant)
        self.adress=adress
    def effectuer_paiement(self):
        self.adress=str(input("Mettez votre adresse mail :"))
        if ("@gmail.com" or "@yahoo.fr"or "@outlook.com") in self.adress:
            print(f"Paiement de {self.adress} effectué avec Paypal avec un␣
 ↪montant de {self.montant} €.")
        elif ("@ensea.fr" or "@societe.com") in self.adress:
            print(f"Paiement effectué avec Paypal via une société avec un␣
 ↪montant de {self.montant} €.")
        else:
            print("Ce n'est pas une adresse mail. Echec de l'opération.")
```

[ ]:
```python
# Le programme en lui-même

montant=int(input("Mettez votre montant: "))
mode=str(input("Choisissez le mode de paiement : "))
if mode=="CB"or mode=="carte bancaire":
    mode=Cartebancaire(montant,8)
    mode.effectuer_paiement()
elif mode=="Paypal" or mode=="PP":
    mode=PayPal(montant,8)
    mode.effectuer_paiement()
else:
     print("\nJe ne connais pas ce mode de paiement. Choisissez les modes de␣
 ↪paiements suivants:\n",
      "['CB','carte bancaire','PP','Paypal']. \n",
     "Echec de l'opération...")
```

```
Paiement effectué avec la carte bancaire avec un montant de 564 €.
```