Programmation Python

CHAPITRE II

Classes
Héritage et Polymorphisme

Younes Lakhrissi
Professeur à l'ENSA de Fès
Université Sidi Mohamed Ben Abdellah

Objectif du chapitre

- **Cours**
  - Classe et sa structure
  - Droits d'accès aux membres
  - Méthodes essentielles : constructeur, setters, getters et str
  - Héritage
  - Polymorphisme

- **Travaux dirigés et pratiques**
  - TP 7 : Gestion d'une famille
  - TP 8 : Gestion d'un championnat de football

POO – Classes

- La programmation objet permet de structurer les applications sous forme d'objets qui interagissent entre eux et avec le monde extérieur.
- Parmi les avantages de la POO c'est de pouvoir développer l'application par partie à travers les classes.
- Les classes permettent d'éviter au programmeur d'utiliser des variables globales. Les variables globales rendent les applications peu lisibles, surtout dans le cas des grandes applications.
- Une classe représente un ensemble d'objets ayant les mêmes propriétés
- Une classe est une encapsulation qui possède :
  - Une structure : la liste des attributs.
  - Un comportement : la liste des méthodes.

POO – Classes

- Un objet est une instance d'une classe :
  - Il possède les mêmes caractéristiques de sa classe.
  - Il concrétise les attributs par des valeurs spécifiques.
- Le principe de réutilisation constitue un grand avantage de la POO. Il permet de créer des classes à partir de classes déjà existantes.
- La réutilisation est atteinte de deux manières :
  - Composition
  - Héritage
  - On discutera en détails ces deux principes plus tard

Particularité de Python

- Toutes les classes héritent implicitement d'une classe mère appelée **Object**.
- Le mot-clé **self** permet de désigner l'objet courant. L'équivalent de **this** en Java ou en C++
- Visibilité : droit d'accès aux membres de la classe
  - Le mot-clé **private** n'existe pas : On préfixe les attributs par **deux underscores**
  - Le mot-clé **protected** n'existe pas : On préfixe les attributs par **un underscore**
  - Le mot-clé **public** n'existe pas : par défaut **tout est public**
- Pas de mot-clé **static** à la différence de la plupart des langages
  - Un attribut qui n'est pas déclaré dans le constructeur est un attribut statique

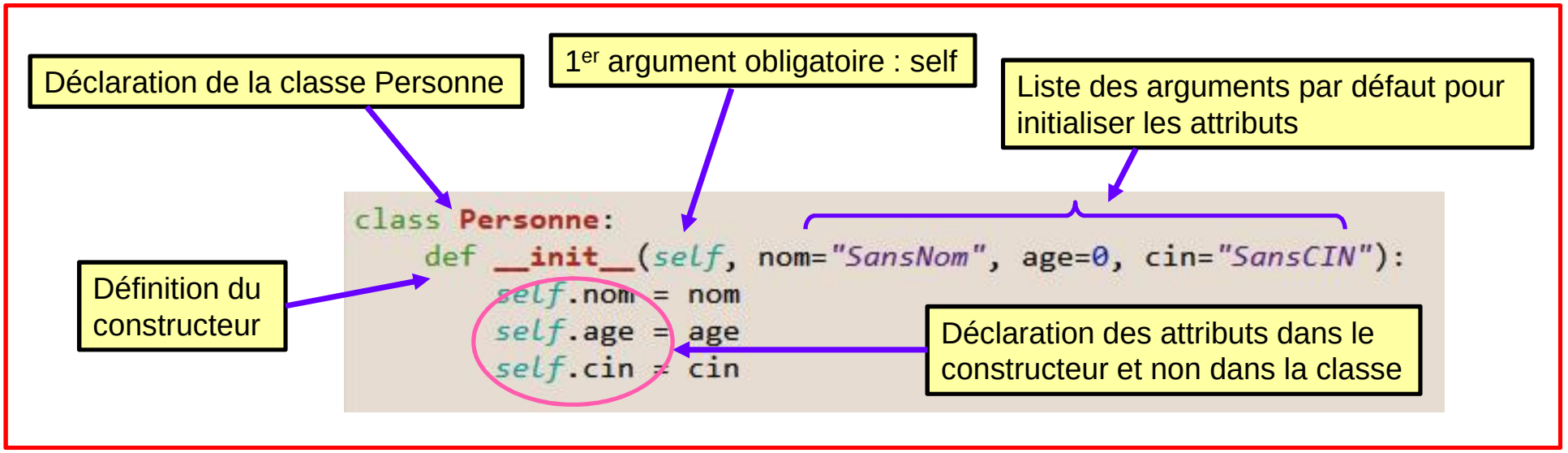
Particularité de Python

- **Pas de surcharge** de fonction et de méthodes en Python !
- Python ne supporte **pas** la déclaration des **constantes**.
  - Pour différencier une constante d'une variable normale, toutes les lettres du nom à attribuer à la constante doivent être en majuscule.
  - Mais c'est juste une convention … définir une constante n'est pas possible.

Constructeur

- **Déclaration du constructeur**
  - La méthode constructeur porte le nom : **__init__( )**
  - Le premier argument doit être **self**, puis suivi de la liste des valeurs pour initialiser les attributs de l'objet.
- **Rôle du constructeur**
  - Le constructeur est le responsable de la construction des objets. Son rôle est d'initialiser les attributs de l'objet en création.
  - Le constructeur donne la forme autorisée et obligatoire pour la création des objets.
  - Il faut alors développer plusieurs formes de constructeurs pour s'adapter au différentes manières de création.
  - Par défaut, toute classe en Python a un constructeur par défaut sans paramètres
- **En Python la surcharge n'est pas permise. Comment faire alors ?**
  - On peut utiliser les valeurs par défaut

Constructeur – classe Personne



Toutes ces créations d'objets sont acceptables

```
#---- Programme Principal --------#
p1 = Personne()
p2 = Personne('Nadia')
p3 = Personne('Nadia', 19)
p4 = Personne('Nadia', 19, 'D1010')
```

Méthode d'affichage __str__()

- **Objectif :**
  - Pour afficher les détails d'un objet, il faut développer la méthode **__str__(self)**
  - C'est l'équivalent de toString() en Java ou l'opérateur << en C++
  - **__str__()** prend **self** comme unique argument
- **Exemple :**
  - Pour la classe Personne, voici une version possible de la méthode __str__()

```
def __str__(self):
    return "Personne[{} - {} - {}]".format(self.nom, self.age, self.cin)
```

  - Pour afficher les informations de l'objet « p », l'appel à __str__() est automatique

```
p = Personne('Nadia', 19, 'D1010')
print(p)
```

```
Personne[Nadia - 19 - D1010]
```

Destructeur

- **Déclaration du destructeur :**
  - Il porte le nom **__del__( )**
  - Doit avoir un seul argument **self**.
- **Rôle du destructeur :**
  - Méthode appelée lors de la destruction d'un objet
  - Son objectif est d'exécuter le code nécessaire pour que l'application puisse continuer son exécution sans l'objet à détruire.
  - L'appel au destructeur est implicite lorsque l'objet n'est plus référencé
  - L'appel peut être explicite en utilisant le mot clé **del**
- **Exemple :**
  - rendre les ressources déjà réservées par l'objet,
  - modifier certains paramètres de l'application, …

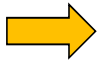
Destructeur – classe Personne

- **Déclaration du destructeur :**

```
class Personne:
    def __init__(self, nom="SansNom", age=0, cin="SansCIN"):
        self.__nom = nom ;  self.__age = age ; self.__cin = cin
    def __del__(self):
        print("destruction de :",self)
```

- **L'appel peut être explicite en utilisant le mot clé del**

```
p1 = Personne("Hicham",33,"AB210")
print("--> avant del")
del p1
print("--> Apres del")
```

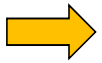


```
--> avant del
destruction de : [Hicham - 33 - AB210]
--> Apres del
```

- **L'appel peut être implicite lorsque l'objet n'est plus référencé**

```
def f():
    p = Personne("Siham",25,"CD123")

print("--> avant f()")
f()
print("--> apres f()")
```



```
--> avant f()
destruction de : [Siham - 25 - CD123]
--> apres f()
```

Contrôle d'accès aux attributs de la classe

- **Problème**
  - L'âge d'une personne doit être positif. Comment faire pour ne pas accepter une valeur négative ?
  - Comment récupérer les attributs (privés) de la classe Personne ?

- **Solution**
  - Bloquer l'accès direct aux attributs (mettre la visibilité à private)
  - Définir des méthodes qui contrôlent l'affectation de valeurs aux attributs

- **A faire**
  - Rendez **private** les 3 attributs de la classe Personne
  - Définissez un **setter** pour chaque attribut
  - Définissez un **getter** pour chaque attribut

Attributs statiques

- Attributs d'objets
  - Attributs déclarés dans le constructeur
  - Attributs propres à chaque objet.
  - L'objet possède la réservation mémoire de ses attributs et peut les manipuler et leur attribuer ses propres valeurs.

- Attributs de classe
  - Attributs déclarés dans la classe
  - Attributs partagés par tous les objets de la classe.
  - Il n' y a pas de réservation par objet mais une seule réservation partagée et utilisée par tous les objets.

Classe Personne – Version finale

Constructeur

Destructeur

Méthode str

Setters

Getters

```
class Personne:
    def __init__(self, nom="SansNom", age=0, cin="SansCIN"):
        self.__nom = nom ;  self.__age = age ; self.__cin = cin
    def __del__(self):
        print("destruction de :",self)
    def __str__(self):
        return "[{} - {} - {}]".format(self.__nom, self.__age, self.__cin)
    def set_nom(self, nom):
        self.__nom = nom
    def set_age(self, age):
        self.__age = age
    def set_cin(self, cin):
        self.__cin = cin
    def get_nom(self):
        return self.__nom
    def get_age(self):
        return self.__age
    def get_cin(self):
        return self.__cin
```

```
#---- Programme Principal --------#
p1 = Personne()
p2 = Personne('Nadia', 19, 'D1010')
print(p1)
print(p2)
```

```
Personne[SansNom - 0 - SansCIN]
Personne[Nadia - 19 - D1010]
```

TP2 – Gestion d'une Famille

- En se basant sur la classe Personne déjà développée, créez la classe Famille.
  - Une famille possède les attributs suivants:
    - Un père
    - Une mère
    - 0 à plusieurs enfants
  - Développez les méthodes suivantes :
    - Le constructeur : __init__()
    - La méthode d'affichage : __str__()
    - La méthode ajouterEnfant(Personne)
    - La méthode enfantPlusGrand() qui retourne l'enfant le plus grand de la famille
    - La méthode enfantPlusPetit() qui retourne l'enfant le plus petit de la famille

TP7 – Exemple d'exécution

Programme principal

```
#------------ Programme principal ------------
pere = Personne("Amine", 44,"A1200")  # Père
mere = Personne("Amina", 40,"C1650")  # Mère
e1 = Personne("Aymane",18,"C3333")  # enfant 1
e2 = Personne("Imane",10)           # enfant 2
e3 = Personne("Imad",3)             # enfant 3

enfants = [e1,e2,e3]

F = Famille(pere,mere,enfants)

e4 = Personne("Manal",20,"C4444")   # enfant 4
F.ajouterEnfant(e4)

print(F)
print("l'enfant le plus grand : ", F.enfantPlusGrand())
print("l'enfant le plus petit : ", F.enfantPlusPetit())
```

Résultat de l'exécution

```
Famille de Amine et Amina :
- Père : [Amine - 44 - A1200]
- Mère : [Amina - 40 - C1650]
- Enfants :
    - [Aymane - 18 - C3333]
    - [Imane - 10 - SansCIN]
    - [Imad - 3 - SansCIN]
    - [Manal - 20 - C4444]
l'enfant le plus grand :  [Manal - 20 - C4444]
l'enfant le plus petit :  [Imad - 3 - SansCIN]
```

TP7 – Solution

```python
def __init__(self, pere, mere, enfants=None):
    self.pere = pere
    self.mere = mere
    if enfants==None:
        self.enfants = []
    else:
        self.enfants = enfants
```

```python
def __str__(self):
    ch="Famille de "+self.pere.get_nom()+" et "+self.mere.get_nom()+" :\n"
    ch+="- Père : "+self.pere.__str__()+"\n"
    ch+="- Mère : "+self.mere.__str__()+"\n"
    ch+="- Enfants :\n"
    for e in self.enfants:
        ch+="\t- "+e.__str__()+"\n"
    return ch
```

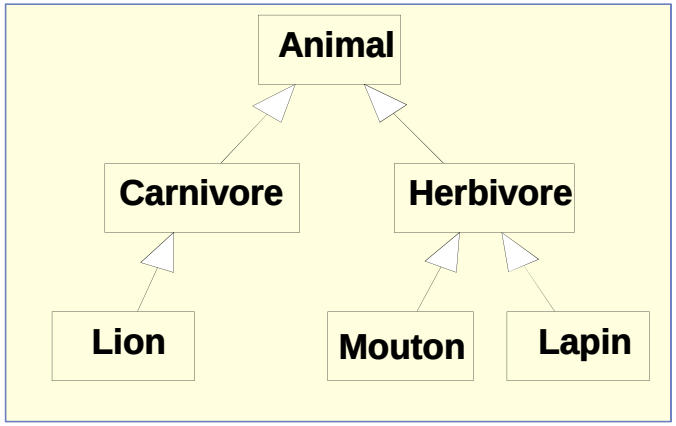
```python
def ajouterEnfant(self, enfant):
    self.enfants.append(enfant)
```

```python
def enfantPlusGrand(self):
    if len(self.enfants)==0:
        return None
    max = self.enfants[0]
    for i in range(1, len(self.enfants)):
        if self.enfants[i].get_age() > max.get_age():
            max = self.enfants[i]
    return max
```

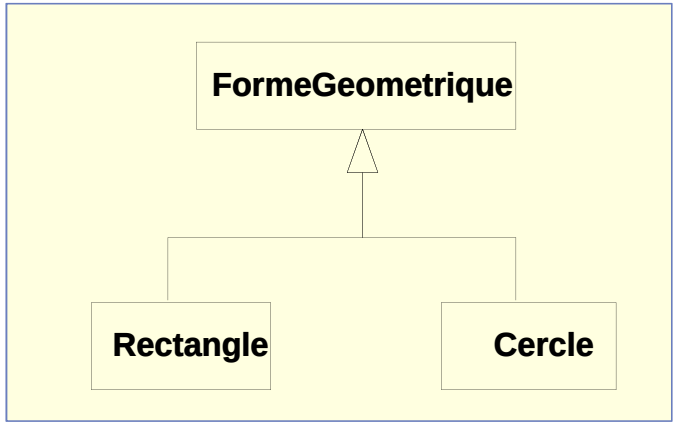
```python
def enfantPlusPetit(self):
    if len(self.enfants)==0:
        return None
    min = self.enfants[0]
    for i in range(1, len(self.enfants)):
        if self.enfants[i].get_age() < min.get_age():
            min = self.enfants[i]
    return min
```

Héritage

- Objectif :
  - Un des piliers de la programmation objet
  - Un mécanisme puissant qui :
    - Crée une hiérarchie entre les classes
    - Favorise la réutilisation du code
    - Favorise l'évolution de l'application



- C'est une relation entre :
  - Un élément général
    - Classe mère :
    - Classe générale,
    - Classe de base,
    - Super-classe
  - Un élément spécifique :
    - Classe fille,
    - Classe spécialisée,
    - Classe dérivée,
    - Sous-classe,



Structure d'une classe dérivée

- Une classe dérivée (spécialisation de la classe mère)
  - Attributs
    - Possède automatiquement les attributs de la classe de base
    - peut en ajouter d'autres
  - Méthodes
    - possède les méthodes de la classe de base
    - peut ajouter de nouvelles méthodes
    - peut redéfinir certaines méthodes
- Forme générale

```
class ClasseFille (ClasseMère):
    # code
```

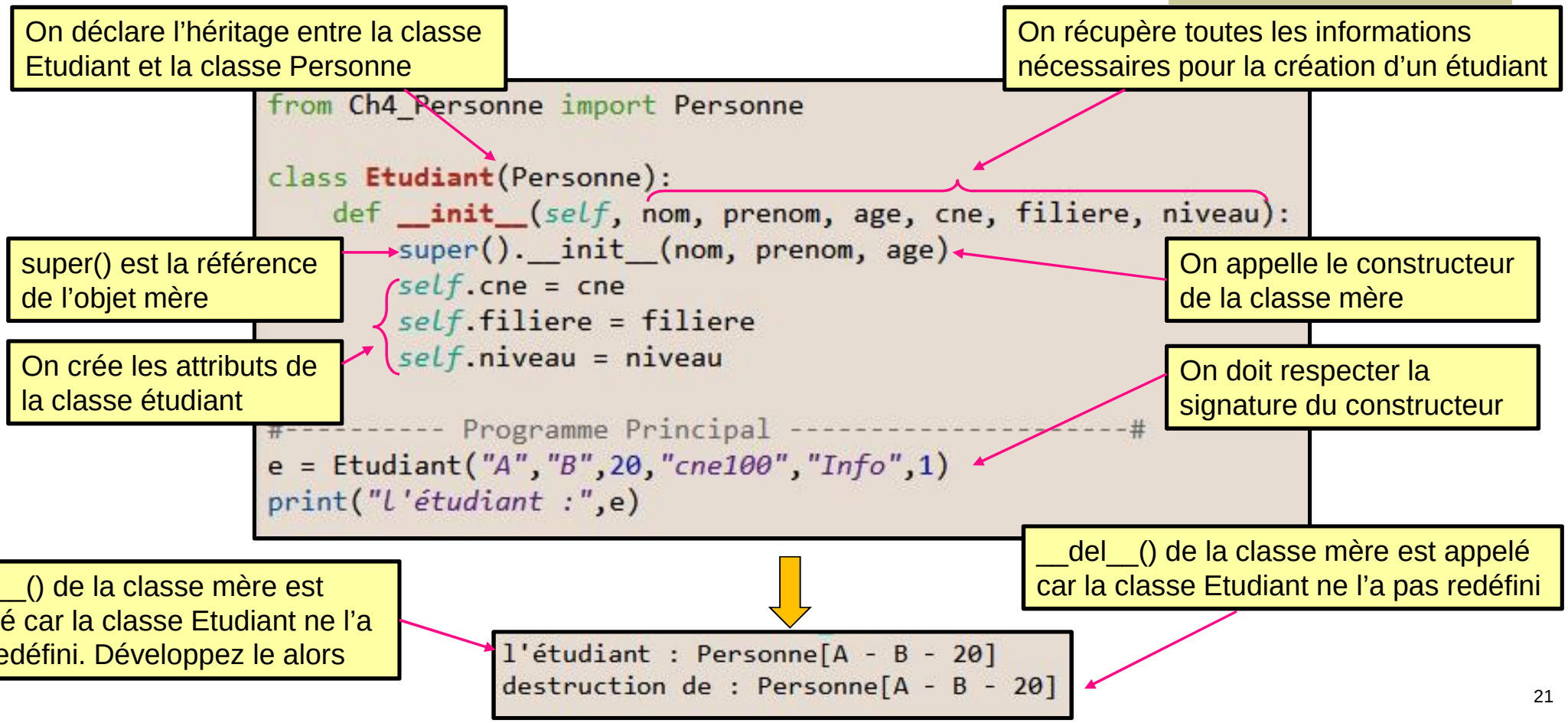
Composition vs Héritage

- **Composition**
  - Une classe est composée si certains de ses membres sont eux-mêmes des objets
  - permet de créer une classe plus complexe à partir d'une ou de plusieurs classes.
  - EXEMPLE : (Personne, DateNaissance), (Chambre, mur), …

- **Héritage**
  - Une classe est dérivée si elle est une spécialisation (une « sorte ») de la classe de base
  - EXEMPLE : (Etudiant , Personne) (Lion, Animal), …

- **Indicateur !**
  - Si on arrive à utiliser le verbe « être », il s'agit de l'héritage
  - Sinon, si on utilise le verbe « avoir », il s'agit d'une composition

TP3(partie1) – Etudiant et Personne



Attention

- **super().__init__()**
  - Constructeur de la classe fille doit appeler le constructeur de la classe mère explicitement, sinon l'initialisation de l'objet mère est non faite.
  - Quand une classe hérite d'une autre classe, elle a la responsabilité de s'assurer que le constructeur de la classe parente est appelé pour garantir l'initialisation.

- **La classe object**
  - Python définit la classe object. Toutes les classes hérite directement ou indirectement de cette classe. Si une classe ne déclare aucune classe parente alors sa classe parente est object.
  - Il est possible de créer des instances de la classe object. Cela reste d'un usage limité car cette classe n'offre aucune méthode particulière et il n'est pas possible d'ajouter dynamiquement des attributs à ses instances.

isinstance()

- La fonction **isinstance()** renvoie True si l'objet donné comme premier argument est du type spécifié comme 2ème argument, sinon False.
- Si le paramètre de type est un tuple, cette fonction renverra True si l'objet est l'un des types du tuple.
- Exemples :
  - Teste si 5 est un entier :
    - x = **isinstance (5 , int)**
  - Teste si "Bonjour" est un des types suivants : float, str, list, ou Personne :
    - x = **isinstance("Bonjour" , (float, str, list, Personne))**
  - Sachant que la classe Voiture hérite de Véhicule, après l'instanciation v = Voiture()
  - isinstance(v, Voiture)
    - True
  - isinstance(v, Vehicule)
    - True

TP3(partie2) – Classe Professeur

- De la même manière du développement de la classe Etudiant, développez la classe Professeur qui hérite de la classe Personne et qui possède les attributs suivants :
  - Spécialité
  - Listes des matières enseignées
  - Établissement d'attache

Polymorphisme

- **L'objectif de l'Héritage**
  - Le polymorphisme est lié à l'héritage :
  - Nouveau code(classe fille) qui réutilise un code existant(mère)
  - Liaison montante de la classe dérivée vers la classe de base
- **L'objectif du Polymorphisme**
  - Ancien code qui utilise un code nouveau
  - Liaison descendante de la classe de base vers la classe dérivée
  - Moyen d'accès uniforme aux objets avec des implémentations différentes
  - C'est un avantage de la programmation orientée objet, car il permet d'étendre ou d'améliorer un système sans modifier le code existant.
- **Exemples :**
  - Client et les articles achetés
  - Gestion de différents types de formes géométriques

TP3(partie3) – Gestion des Equipes de Football

- **En se basant sur les classes Personne Etudiant et Professeur déjà développées, réalisez une application qui répond aux exigences suivantes :**
  - Une équipe possède un **nom** et composée de **6 joueurs**.
  - Un joueur peut être soit un professeur soit un étudiant.
  - Un joueur possède un numéro de maillot et occupe un poste dans son équipe : attaquant, défenseur, gardien, …
  - Développez la classe **Joueur** et ses services de base : constructeur, str, setters, …
  - Développez la classe **Equipe** avec ses services de base, et spécifiez aussi :
    - la méthode « ajouterJoueur ».
    - la méthode « supprimerJoueur ».
    - la méthode « afficherMembres » qui affiche les informations des 6 membres de l'équipe.

TP9 – Formes Géométriques (optionnel)