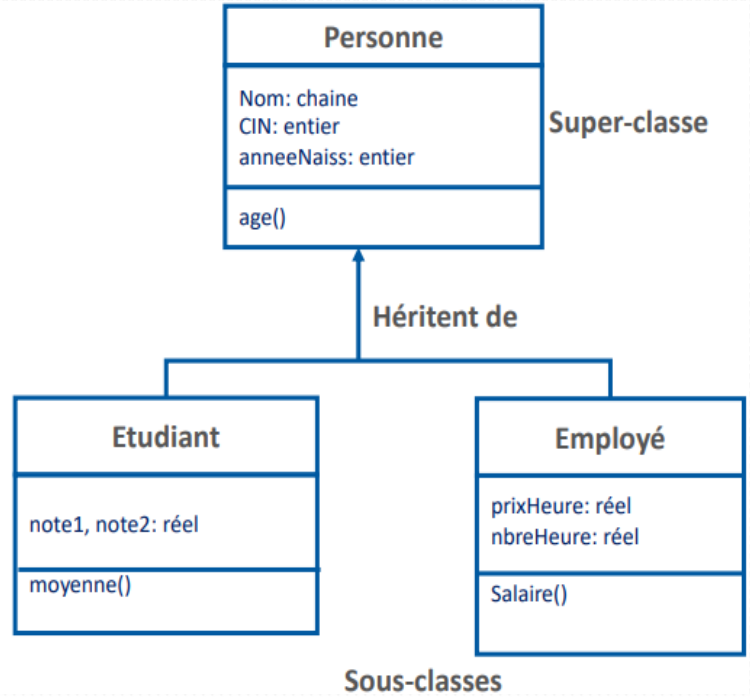
L'héritage simple

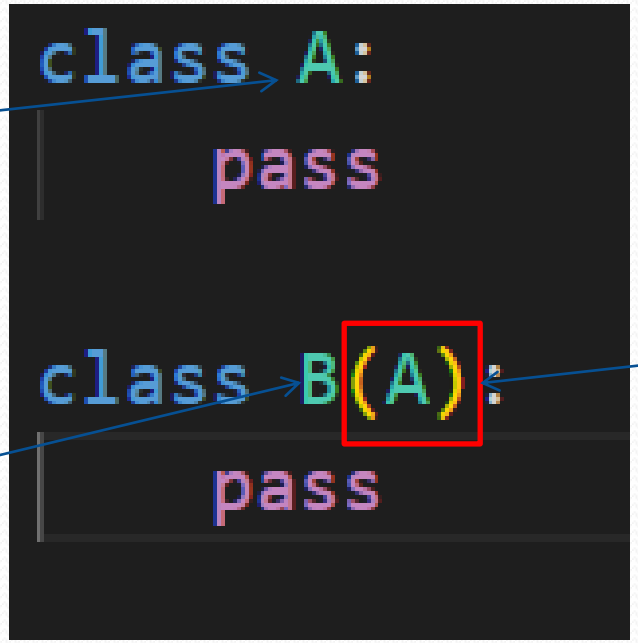
Héritage simple

- Lorsque une classe hérite d'une seule classe, on dit que l'héritage est simple. Exemple : la classe Etudiant hérite d'une seule classe , qui est la classe personne.



Comment appliquer l'héritage

- En python, pour dire qu'une classe B hérite d'une classe A, on utilise des **parenthèses** ()



Exemple

- La classe voiture hérite de la classe véhicule:

```
#classe mère
class vehicule:
    def __init__(self,a,b,c) :
        self.matricule=a
        self.marque=b
        self.prix=c

    def Afficher(self):
        print(f"Matricule:{self.matricule} \t Marque:{self.marque} \t

#classe fille
class voiture(vehicule):
    pass

veh1=vehicule("AB12345","ford",120000)
veh1.Afficher()
voit1=voiture("BX14785","Renault",150000)
voit1.Afficher()
```

Commentaire exemple 1

- La classe voiture hérite de la classe véhicule et ne contient aucun code, aucun attribut et aucune méthode.
- La classe voiture a hérité de la classe véhicule le constructeur d'initialisation (donc les attributs) et la méthode Afficher

  - On pu créer un nouveau objet voiture à l'aide du constructeur d'initialisation de la classe mère

  - On a pu Afficher les informations de la voiture grâce à la méthode Afficher() de la classe mère

Ajouter des fonctionnalités à la classe fille

- Exemple

```
#classe fille
class voiture(vehicule):
    def __init__(self,a,b,c,options):
        self.matricule=a
        self.marque=b
        self.prix=c
        self.options=options

    def ListeOptions(self):
        for x in self.options:
            print(x)
```

Ajouter des fonctionnalités à la classe fille

- Dans la classe fille **voiture**, on a ajouté un nouvel attribut qui est **options** qui représente les options qu'on peut avoir dans une voiture, cet attribut est sous forme d'une liste de valeurs. Donc on a définit le constructeur de la classe voiture

- On a ajouté une nouvelle méthode dans la classe fille **ListeOptions()**, permettant d'afficher les options dans une voiture

Héritage du constructeur méthode1

- Vous remarquez que le constructeur de la classe voiture c'est le même que celui de la classe mère, sauf l'attribut options qui est nouveau.

- Pour bénéficier des apports du principe de l'héritage, c'est mieux d'éviter la répétition en héritant du constructeur de la classe mère comme suit:

Appel du constructeur de la classe mère

```
class voiture(vehicule):
    def __init__(self,a,b,c,options):
        vehicule.__init__(self,a,b,c)
        self.options=options
```

Héritage du constructeur méthode2

- La deuxième méthode consiste à utiliser le mot **super()** qui signifie super-classe ou bien la classe mère. Dans ce cas on a pas besoin de préciser le paramètre self.

```
class voiture(vehicule):
    def __init__(self,a,b,c,options):
        super().__init__(a,b,c)
        self.options=options
```

Redéfinition d'une méthode de la classe mère

- On peut redéfinir la méthode afficher dans la classe fille de manière à ce qu'elle qui affiche toutes informations de la classe voiture , y inclus les options de cette voiture.

```
def Afficher(self):
    print(f"Matricule:{self.matricule} \t Marque:{self.marque} \t Prix:{s
    print("Les options de la voiture :")
    for x in self.options:
        print(x)
```

- Remarquez que l'affiche de la matricule, la marque et le prix se répète car il existe déjà dans la classe mère.

Redéfinition d'une méthode de la classe mère

- Pour éviter la répétition dans la méthode Afficher on peut écrire:

```
def Afficher(self):
    vehicule.Afficher(self)
    print("Les options de la voiture :")
    for x in self.options:
        print(x)
```

- Ou bien :

```
def Afficher(self):
    super().Afficher()
    print("Les options de la voiture :")
    for x in self.options:
        print(x)
```

Les modificateurs d'accès et l'héritage

- Lorsque un élément (attribut ou méthode) est privée dans une classe , c'est impossible d'y accéder par d'autres classes même s'il s'agit d'une classe fille
- Pour permettre à une classe d'utiliser les membres de la classe mère, il faut obligatoirement les mettre en mode protégés (**proteceted**)
- Un élément **protected** (attribut ou méthode) est utilisable uniquement par la classe elle-même et par la classe dérivée.

Protected en python

- Pour rendre un élément protected en python, il faut précéder son nom par un soulignement (_)

```
class vehicule:
    def __init__(self,a,b,c):
        self._matricule=a
        self._marque=b
        self._prix=c

    def afficher(self):
        print(f"{self._matricule}\t {self._marque}\t {self._prix}")

class voiture(vehicule):

    def afficher(self):
        print(f"{self._matricule}\t {self._marque}\t {self._prix}")
```