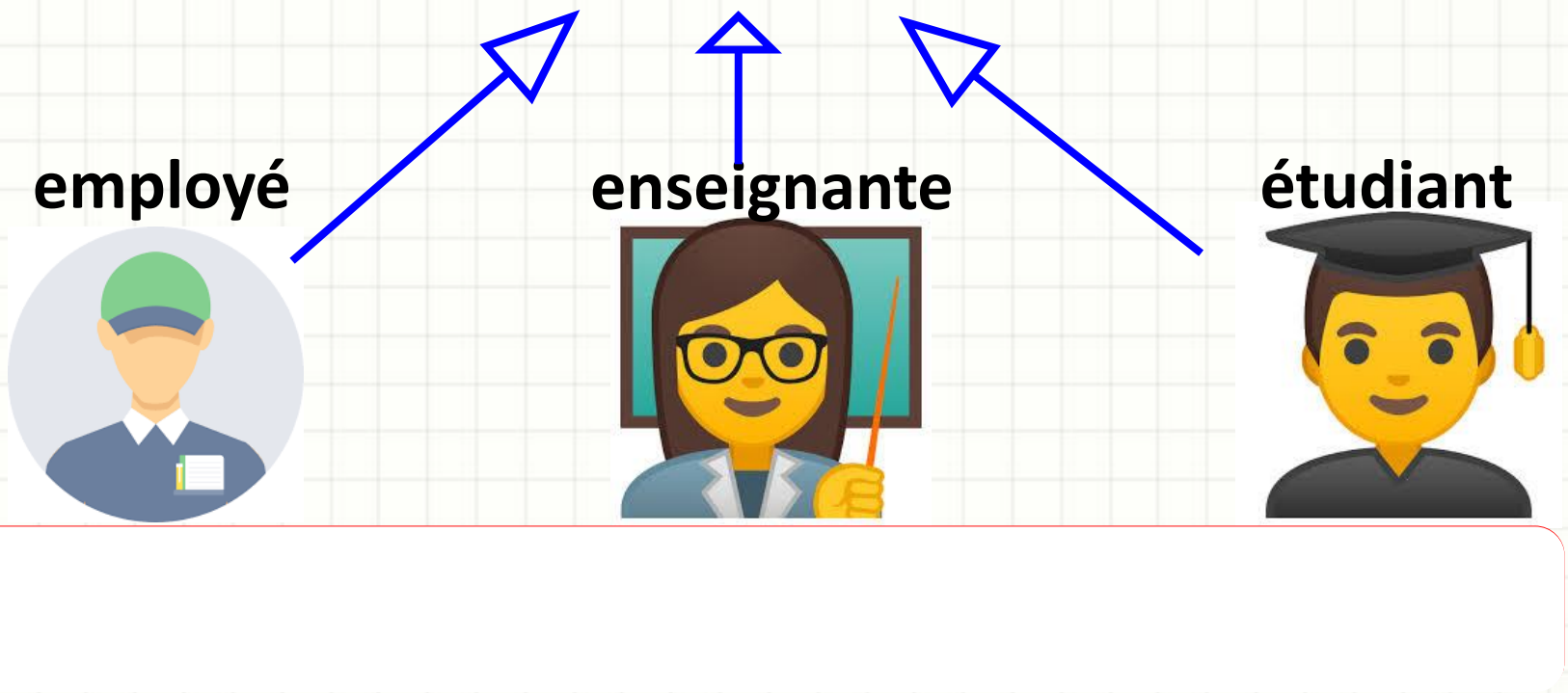
FORMATION PYTON
HÉRITAGE EN PYTHON

Mustapha HAIN
ENSAM-Casablanca
Infohain@gmail.com

POO offre plusieurs relations, à savoir:

- **Héritage,**
- **Agrégation,**
- **Composition**
- **Association avec cardinalités.**

L'HÉRITAGE SIMPLE



L'HÉRITAGE SIMPLE

À partir d'une classe A, on peut créer une classe B qui possède toutes les caractéristiques de la classe A, à laquelle on ajoute un certain nombre de méthodes qui sont spécifiques à B. On dit que :

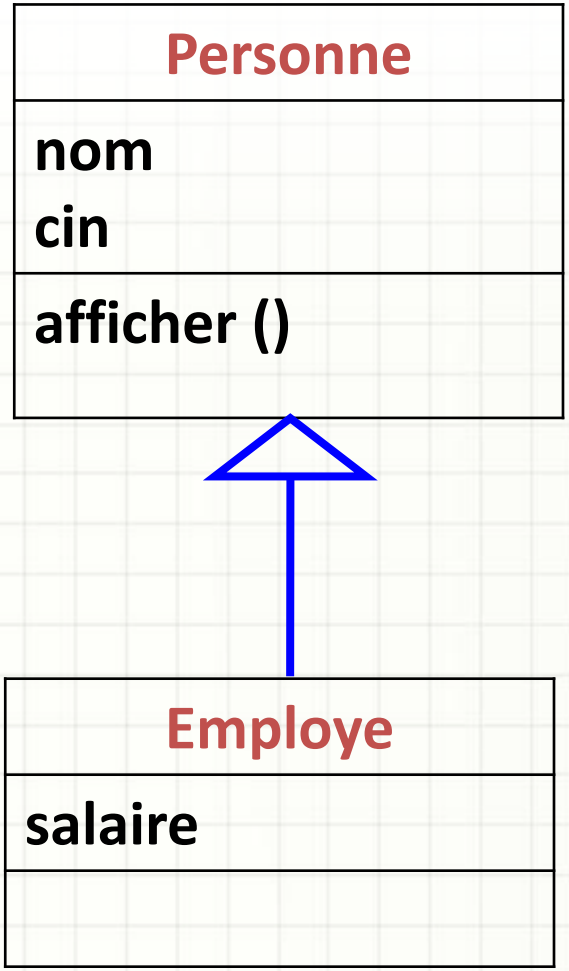
- **la classe B hérite de la classe A ;**
- **la classe B est une sous-classe de la classe A ;**
- **la classe B spécialise la classe A ;**
- **la classe B étend la classe A ;**
- **la classe A est une super-classe de la classe B ;**
- **la classe A généralise la classe B.**

On dit aussi que:

- **la classe A est une classe de base**
- **la classe B est une classe dérivée**

Syntaxe

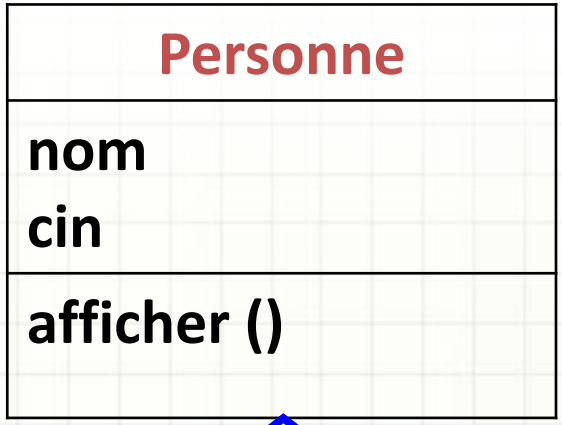
```
class mere:
    # corps de la classe mère

class enfant(mere):
    # corps de la classe enfant
```



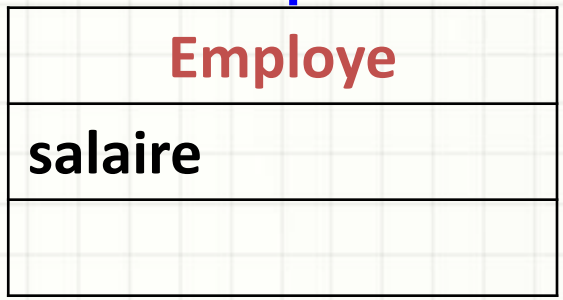
L'HÉRITAGE SIMPLE

```
class Personne :
    # Constructeur
    def __init__(self, nom, cin):
        self.nom = nom
        self.cin = cin
    def afficher(self):
        print("Nom : ", self.nom)
        print("CIN : ", self.cin)
```

#Constructeur




```
class Employe(Personne):
    def __init__(self, nom, cin, salaire):
        self.salaire = salaire
        # appeler __init__ de la classe mère (Personne)
        Personne.__init__(self, nom, cin)
```




```
# création d'une variable d'instance
p = Employe("Ismail", "EE4567", 7000)
p.afficher()
```

L'HÉRITAGE SIMPLE

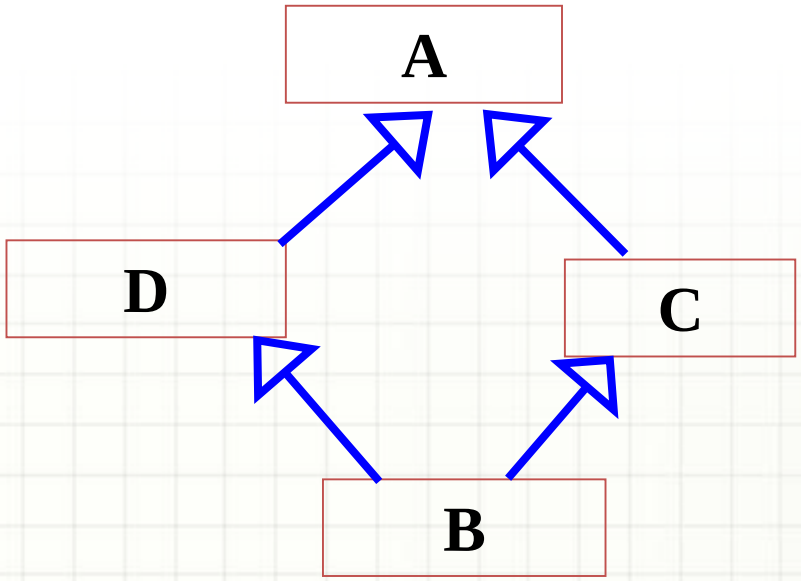
```
class Personne():
  # Constructeur
   def __init__(self, nom, cin):
      self.nom = nom
      self.cin = cin
    def afficher(self):
        print("Nom : ",self.nom)
        print("CIN : ",self.cin)

class Employe( Personne ):
          def __init__(self, nom, cin, salaire):
                    self.salaire = salaire
# appeler __init__ de la classe mère (Personne)
                   Personne.__init__(self, nom, cin)

p=Employe("Ismail", "EE4567", 7000)
p.afficher()
```

```
Nom : Ismail
CIN : EE4567
```

L'HÉRITAGE SIMPLE

HÉRITAGE MULTIPLE

Python permet de dériver une classe à partir de plusieurs classes à la fois, cela s'appelle HÉRITAGE MULTIPLE.




```
Class mere_1:
   # corps de mere_1
Class mere_2:
   # corps de mere_2
Class mere_3:
   # corps de mere_3
Class enfant(mere_1, mere_2, mere_3):
   # corps de la classe enfant
```

La classe enfant est dérivée de trois classes mere_1, mere_2, mere_3. En conséquence, il héritera des attributs et des méthodes des trois classes.

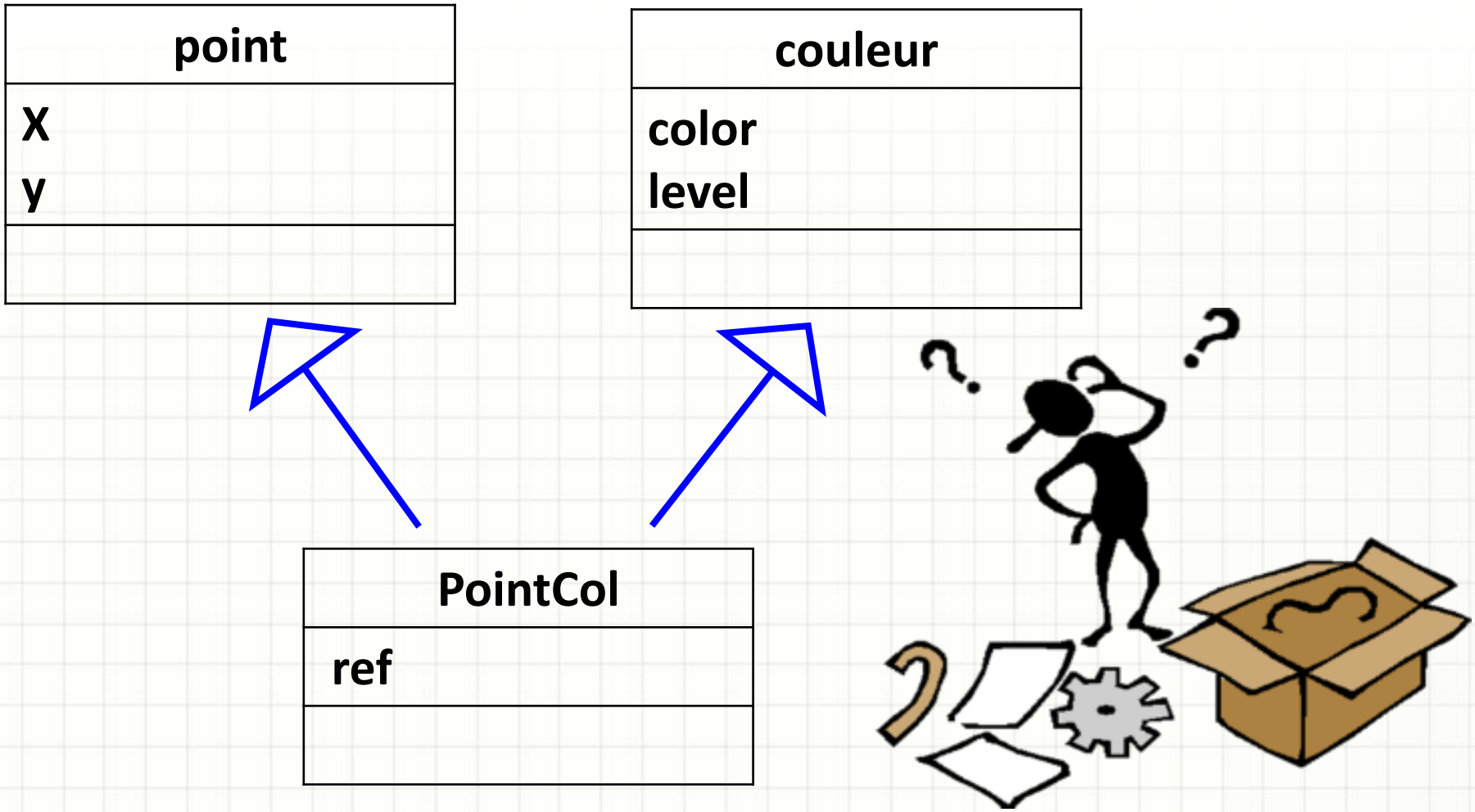
HÉRITAGE MULTIPLE

```
class Person(object):
    def __init__(self, name,
idnumber):
        self.name = name
        self.idnumber = idnumber
    def afficher(self):
        print(self.name,self.idnumber)
class Employee(object):
    def __init__(self, salary, post):
        self.salary = salary
        self.post = post
    def afficher(self):
        print(self.salary, self.post)
```

```
class Leader(Person, Employee):
    def __init__(self, name,
idnumber, salary, post, points):
        self.points = points
        Person.__init__(self, name, idnumber)
        Employee.__init__(self, salary, post)
    def afficher(self):
        Person.afficher(self)
        Employee.afficher(self)
        print(self.points)
```

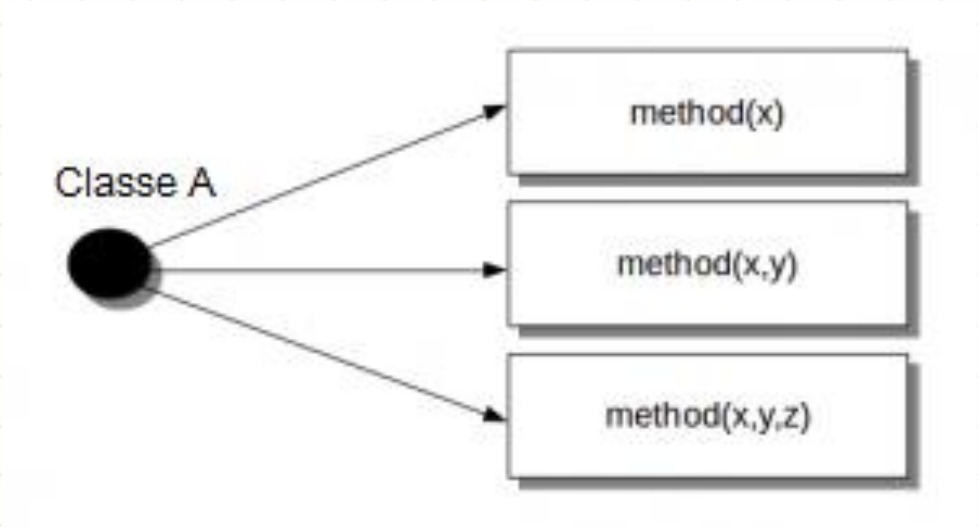
```
p1=Person("Sara",10)
p2=Employee(7000,"Manager")
p3=Leader("Karim",11,6500,"Sales","DG"
)
p1.afficher()
p2.afficher()
p3.afficher()
```

HÉRITAGE MULTIPLE

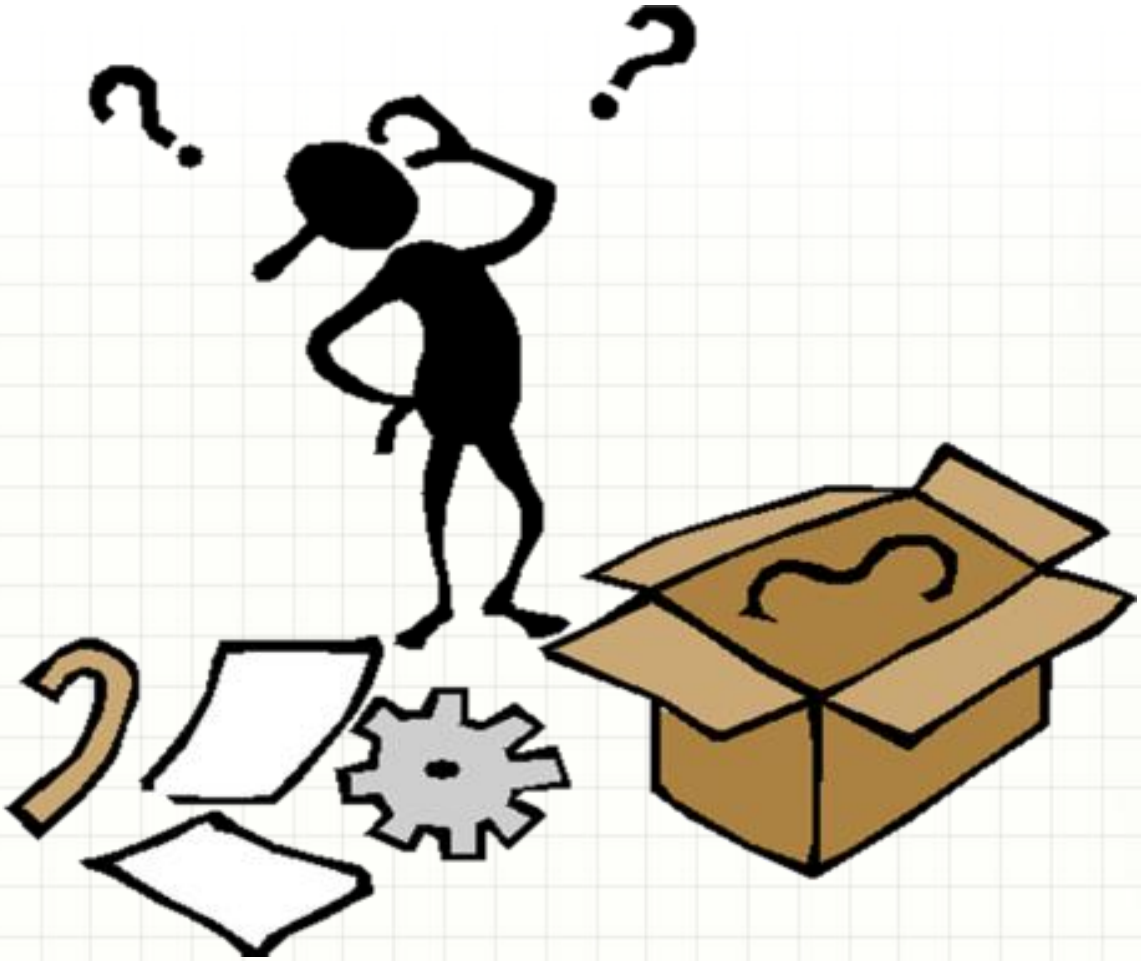


SURCHARGE(OVERLOADING)

Le surcharge(Overloading) est de définir plusieurs méthodes de signatures différentes pour un même nom. C'est-à-dire définir, dans la même classe, plusieurs fonctions de même nom mais de différent nombre d'arguments.



En python pas de surcharge, La solution donc est d'utiliser les paramètres avec valeur par défaut.



Travaillons ensemble- Activité 1