



avec pythonTM

Variables de Classes
et héritage



Variables de classes

- Ici une classe Employe possédant uniquement trois attributs, "nom", "prenom" et "ID"

```
class Employe:....  
    def __init__(self,nom,prenom) -> None:  
        self.nom = nom  
        self.prenom = prenom  
        self.ID = random.randint(1000,9999)
```

Variables de classes



```
class Employe:
    ....liste_employe = []
    ....next_ID = 1000
    ....
    ....def __init__(self,nom,prenom) -> None:
    ....    ....self.nom = nom
    ....    ....self.prenom = prenom
    ....    ....self.ID = Employe.next_ID
    ....
    ....    ....Employe.next_ID += 1
    ....    ....Employe.liste_employe.append(self)
```

Variables appartenant à la classe Employe

Variables appartenant à l'objet créé à partir de la classe Employe

Ici on modifie les variables de classes. Toutes les instantiations accéderont aux mêmes variables de classe



Variables de classes

- › Une variable de classe est accessible directement à partir de la classe ou bien à partir d'une instantiation.
- › Il s'agit de la même variable ayant la même valeur.

```
employe1 = Employe("Anna", "Tremblay")  
employe2 = Employe("Bartholemy", "Duchamp")
```

```
for emp in Employe.liste_employe:·  
    ····print(f'{emp.prenom} {emp.nom} {emp.ID}')
```



```
for emp in employe1.liste_employe:·  
    ····print(f'{emp.prenom} {emp.nom} {emp.ID}')
```

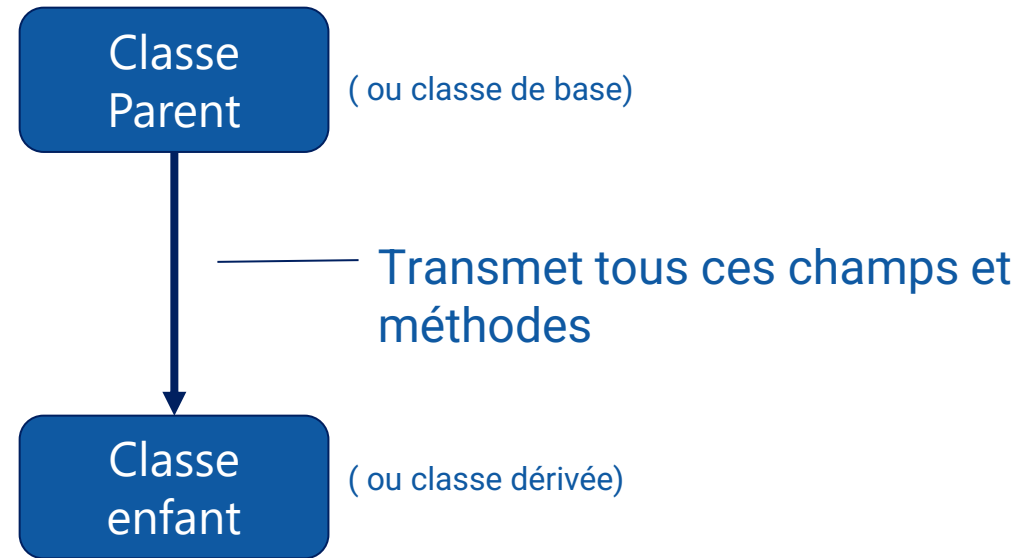
PROBLÈMES 1 SORTIE TERMINAL ...

```
Tremblay Anna 1000  
Duchamp Bartholemy 1001  
Tremblay Anna 1000  
Duchamp Bartholemy 1001
```

L'héritage des classes



- Permet de définir une classe à partir d'une classe déjà existante




- Permet d'hériter des champs et des méthodes de la classe parent.

Héritage



```
class Programmeur(Employe):  
    ...def __init__(self, nom, prenom, language_favori) -> None:  
    ...    super().__init__(nom, prenom)  
    ...    self.language_favori = language_favori
```

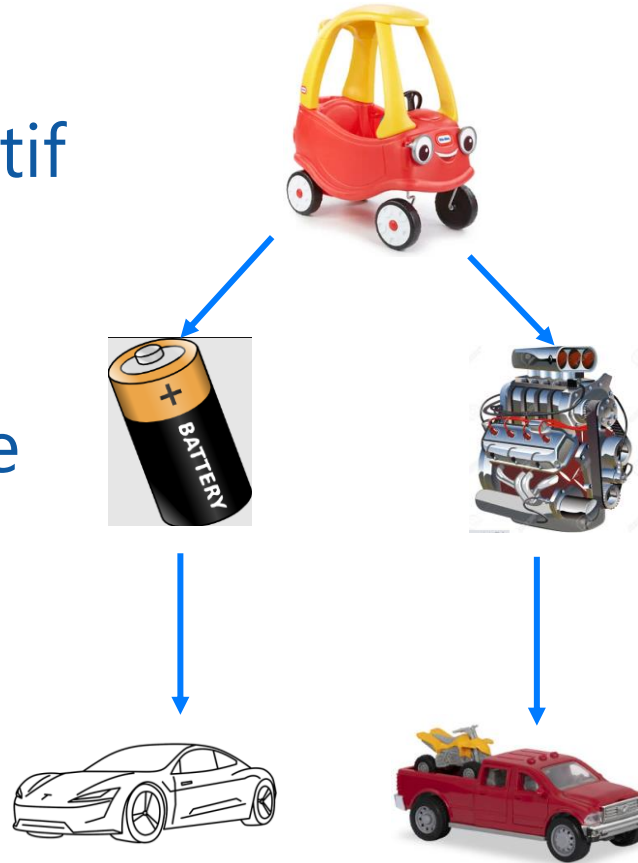


super() fait référence au parent
Employe. On utilise sa méthode `__init__()`
pour lui passer les valeurs de ses propriétés.

Héritage transitif



- > L'héritage est transitif en Python.
- > Une classe hérite de tous les attributs et méthodes de tous ses ancêtres.



```
class Voiture:  
    ...def __init__(self,marque) -> None:  
    ...    self.marque = marque
```

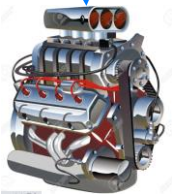
```
class Voiture_moteur(Voiture):  
    ...def __init__(self, marque,reservoir):  
    ...    super().__init__(marque)  
    ...    self.reservoir = reservoir
```

```
class Pickup(Voiture_moteur):  
    ...def __init__(self, marque, reservoir,puissance):  
    ...    super().__init__(marque, reservoir)  
    ...    self.puissance = puissance
```

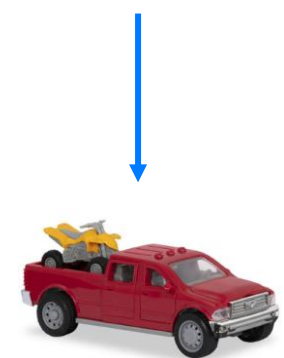
Héritage transitif



```
class Voiture:  
    ...def __init__(self, marque) -> None:  
    ...    self.marque = marque
```



```
class Voiture_moteur(Voiture):  
    ...def __init__(self, marque, reservoir):  
    ...    super().__init__(marque)  
    ...    self.reservoir = reservoir
```



```
class Pickup(Voiture_moteur):  
    ...def __init__(self, marque, reservoir, puissance):  
    ...    super().__init__(marque, reservoir)  
    ...    self.puissance = puissance
```

➤ Un Pickup hérite des attributs de tous ses ancêtres

```
16   remorque = Pickup("Ford", "60L", "1200hp")  
17  
18   print(f"J'aime mon pick {remorque.marque} avec  
      {remorque.puissance} et et un réservoir de  
      {remorque.reservoir}")
```

PROBLÈMES

1

TERMINAL

...

Python

+

✓

□

🗑

^

×

J'aime mon pick Ford avec 1200hp et et un réservoir de 60L

Héritage des méthodes



```
1 class Voiture:
2     ...def __init__(self,marque) -> None:
3     ...self.marque = marque
4     ...
5     ...def klaxon(self):
6     ...print("honk!")
7
```

```
15 class Pickup(Voiture_moteur):
16     ...def __init__(self, marque, reservoir,puissance):
17     ...super().__init__(marque, reservoir)
18     ...self.puissance = puissance
19
20 remorque = Pickup("Ford","60L","1200hp")
21 remorque.klaxon()
```

> La classe Pickup hérite de la méthode klaxon et les objets de la classe Pickup peuvent donc utiliser cette méthode

PROBLÈMES 1 TERMINAL ...

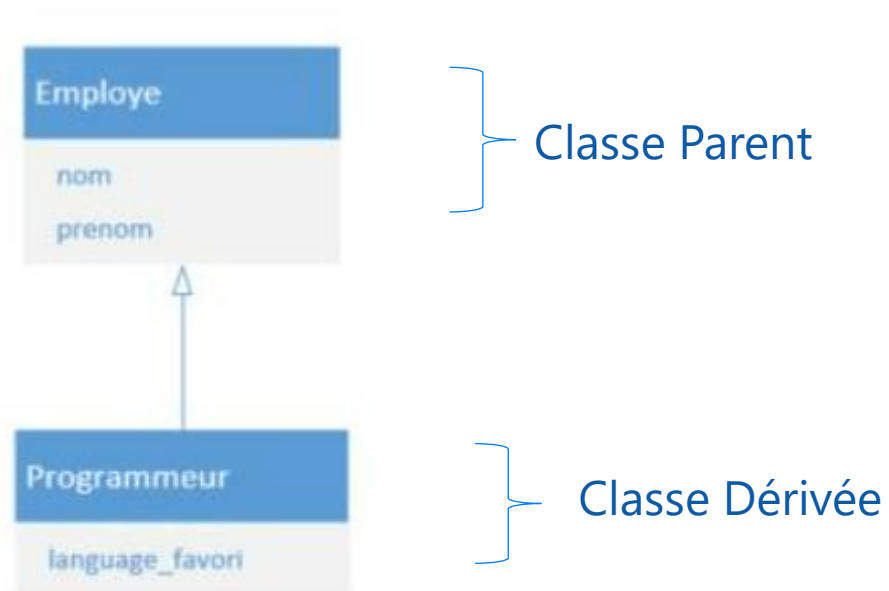
Python + - □ □ ^ ×

honk!

Modélisation UML



- › L'héritage est illustré ainsi dans UML



Un Programmeur EST un Employe