



avec pythonTM

Attributs privés,
Propriétés,
Méthodes privées

Attributs privés.

Propriétés, getters, setters

- > Attributs privés :
 - > Restreins l'accès à certaines valeurs hors de la classe.
 - > Nom commence par un "_" (un seul underscore)
 - > C'est un **standard** et non une règle
- > Propriétés :
 - > Permet de contrôler comment on interagit avec certains attributs.

Encapsulation



- Concept fondamental en programmation objet.
- Consiste en le regroupement de données, valeurs, et fonctions dans un bloc pour permettre la lecture et manipulation de ces données.
- Les classes et instances permettent un premier niveau d'encapsulation.
- Les attributs privés et les propriétés permettent un niveau supplémentaire d'encapsulation

Attributs privés



- On peut vouloir restreindre l'accès à certaines valeurs hors de la classe.
- En Python, la désignation d'un attribut comme étant privé est fait simplement en commençant son nom par un "_" (un seul underscore)

```
class Employe:  
    ... def __init__(self, nom, prenom, salaire):  
    ...     self.nom = nom  
    ...     self.prenom = prenom  
    ...     self._salaire = salaire
```

Attributs publics

Attribut privé



Attributs privés

Par **standard**, on n'appelle jamais les attributs commençant par un "_" hors de la classe.

(rien ne nous empêche d'appeler un attribut privé si on veut)

```
class Employe:
    ...def __init__(self,nom,prenom,salaire):
    ...    self.nom = nom
    ...    self.prenom = prenom
    ...    self._salaire = salaire

chimiste = Employe("Belatekallim","Tapputi","45k")

print(chimiste.nom)
print(chimiste.prenom)
print(chimiste.salaire)
```

PROBLÈMES

1

SORTIE

CONSOLE DE DÉBOGAGE

TERMINAL

.NET

Belatekallim

Tapputi

Traceback (most recent call last):

File "c:\Users\pierre-paul.gallant\Cégep Édouard-Montpe

*Dans la plupart des langages, on peut indiquer qu'un attribut est privé avec un mot-clef.

Cet attribut n'est alors aucunement accessible hors de la classe.



Propriétés (getters)

- > Si on veut quand même avoir accès au salaire de l'employé, on va devoir utiliser un décorateur pour créer une propriété.
- > Une propriété se comporte généralement comme un attribut, mais est définie à l'aide d'une méthode.

```
....@property  
....def salaire(self):  
....|....return self._salaire
```

```
print(chimiste.nom)  
print(chimiste.prenom)  
print(chimiste.salaire)
```

PROBLÈMES

SORTIE

CONSOLE DE DÉBOGAGE

TERMINAL

.NE

```
Belatekallim  
Tapputi  
45k
```



Propriétés (getters)

- > Dans cet exemple, la différence entre l'attribut et la propriété est:
- > nom et prénom sont des attributs et on peut changer leur valeur.
- > salaire est une propriété et sa valeur ne peut pas être changée.

```
chimiste.nom = "Tapi"  
print(chimiste.nom)
```

```
chimiste.prenom = "Bela"  
print(chimiste.prenom)
```

PROBLÈMES	SORTIE	CONSOLE DE DÉBOGAGE	<u>TERMINAL</u>
			Tapi Bela

```
chimiste.salaire = "50k"  
print(chimiste.salaire)
```

PROBLÈMES	SORTIE	CONSOLE DE DÉBOGAGE	TERMINAL
			Traceback (most recent call last): File "c:\Users\pierre-paul.gallant\Cégep Édouard (Pilote réseau)\R21\exemple.py", line 22, in <m chimiste.salaire = "50k"



Propriétés (setters)

- > Les setters nous permettent de changer les valeurs des propriétés.
- > Il faut encore utiliser un décorateur et le comportement des setters sera défini par une méthode.

```
...@salaire.setter
...def salaire(self,nvx_salaire):
...    ...if nvx_salaire > self._salaire:
...    ...    self._salaire = nvx_salaire
```

- > **N.B.**, La syntaxe de ce décorateur est légèrement différente.
`@nom_de_la_propriété.setter`



Propriétés (setters)

- > Ce setter contrôle la façon dont on change la valeur du salaire.
- > Si la nouvelle valeur est inférieure à l'ancienne, la valeur du salaire n'est pas changée.

```
....@salaire.setter  
....def salaire(self,nvx_salaire):  
....|....if nvx_salaire > self._salaire:  
....|....|....self._salaire = nvx_salaire
```

```
chimiste = Employe("Belatekallim","Tapputi",45000)
```

```
chimiste.salaire = 30000  
print(chimiste.salaire)  
chimiste.salaire = 60000  
print(chimiste.salaire)
```

PROBLÈMES

SORTIE

CONSOLE DE DÉBOGAGE

TERMINAL

.NET I

45000

60000



Propriétés (setters)

- > Ce setter contrôle la façon dont on change la valeur du salaire.
- > Si la nouvelle valeur est inférieure à l'ancienne, la valeur du salaire n'est pas changée.

```
....@salaire.setter  
....def salaire(self,nvx_salaire):  
....|....if nvx_salaire > self._salaire:  
....|....|....self._salaire = nvx_salaire
```

```
chimiste = Employe("Belatekallim","Tapputi",45000)
```

```
chimiste.salaire = 30000  
print(chimiste.salaire)  
chimiste.salaire = 60000  
print(chimiste.salaire)
```

PROBLÈMES

SORTIE

CONSOLE DE DÉBOGAGE

TERMINAL

.NET I

45000

60000



Propriétés (delete)

- > Les propriétés ne peuvent pas être supprimées normalement.
- > Le décorateur `@nom_de_la_propriété.delete` est nécessaire pour supprimer une propriété

```
....@salaire.delete
....def salaire(self):
....|....del self._salaire
```



Propriétés (delete)

> Sans delete :

```
30
31 del chimiste.salaire
32 print(chimiste.salaire)
```

```
PROBLÈMES  SORTIE  CONSOLE DE DÉBOGAGE  TERMINAL

in <module>
  del chimiste.salaire
AttributeError: can't delete attribute 'salaire'
```

> Ne peut pas exécuter la ligne 31, ne peut pas supprimer l'attribut "salaire"

> avec delete :

```
23 .....@salaire.delete
24 .....def salaire(self):
25 .....|.....del self._salaire
26 .....
27 chimiste = Employe("Belatekallim", "Tapputi", 45000)
28
29 del chimiste.salaire
30 print(chimiste.salaire)
31
```

```
Traceback (most recent call last):
  File "c:\Users\pierre-paul.gallant\Cégep Édouard-Montpetit\CMT-420_Intro
in <module>
  print(chimiste.salaire)
  File "c:\Users\pierre-paul.gallant\Cégep Édouard-Montpetit\CMT-420_Intro
in salaire
  return self._salaire
AttributeError: 'Employe' object has no attribute '_salaire'. Did you me
```

> Ne peut pas faire l'impression, l'attribut a été supprimé.



Propriétés (delete)

- > Puisque le comportement des deleters est déterminé par une fonction, on décide ce qui se passe lorsqu'on supprime un attribut.

```
....@salaire.delete  
....def salaire(self):  
....    self._salaire = 0
```

```
chimiste = Employe("Belatekallim", "Tapputi", 45000)
```

```
del chimiste.salaire  
print(chimiste.salaire)
```

- > Maintenant, supprimer la valeur "salaire" ne retire pas l'attribut, mais plutôt il réinitialise le salaire à 0.

PROBLÈMES

SORTIE

CONSOLE DE DÉBOGAGE

TERMINAL

.NET INTERACTIVE

0

Exemple



```
demo.py > ...
1 class Employe:
2     def __init__(self,nom,prenom,salaire) -> None:
3         self.nom = nom
4         self.prenom = prenom
5         self._salaire = salaire
6
7     @property
8     def salaire(self):
9         return f"{self._salaire} $"
10
11    @salaire.setter
12    def salaire(self,nvx_salaire):
13        if nvx_salaire > self._salaire:
14            self._salaire = nvx_salaire
15
16    @salaire.deleter
17    def salaire(self):
18        self._salaire = 0
19
```

```
20
21 emp1 = Employe("Tremblay","Jonathan",60000)
22 print(emp1.salaire)
23 print(emp1._salaire)
24
25 emp1.salaire = 70000
26 print(emp1.salaire)
27
28 del emp1.salaire
29 print(emp1.salaire)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\pierre-paul.gallant\OneDrive - Cégep Édouard-
lant/OneDrive - Cégep Édouard-Montpetit/Session 24H/2N6R-
60000 $
60000
70000 $
0 $
```

Méthodes privées

- > Méthodes qui ne sont pas utilisées hors de la classe.
- > Seule la classe peut appeler ces méthodes.
- > On indique qu'une méthode est privée en mettant un "__" (double underscore)

```
.....def __methode_prive():  
.....|.....print("Cette méthode est privée.")
```



Méthodes privées

- Seule la classe peut appeler ces méthodes.
- Appeler une méthode privée hors de la classe génère une erreur.

```
28     ...def __methode_prive():
29     ...     print("Cette méthode est privée.")
30
31     Employe.__methode_prive()
32
```

PROBLÈMES SORTIE CONSOLE DE DÉBOGAGE TERMINAL .NET INTERACTIVE JUPYTER

```
Employe.__methode_prive()
AttributeError: type object 'Employe' has no attribute 'methode prive'
```




Méthodes privées

- Toutes les types de méthodes peuvent être privés.

```
... def __methode_instance_prive(self):  
...     pass  
... @classmethod  
... def __methode_classe_prive(cls):  
...     pass  
... @staticmethod  
... def __methode_static_prive():  
...     pass
```



- > On utilise les méthodes privées pour les mêmes raisons qu'on utilise des fonctions dans un script :
 - > Faire des blocs de code réutilisable.
 - > Séparer les tâches en sous-tâches plus simples.
 - > Rendre le code lisible en donnant des noms significatifs aux différentes tâches.
 - > Rendre le code facile à maintenir.
- > DE PLUS, les méthodes privées permettent d'encapsuler des opérations qu'on ne veut pas qu'elles soient accessibles par d'autres classes ou objets.

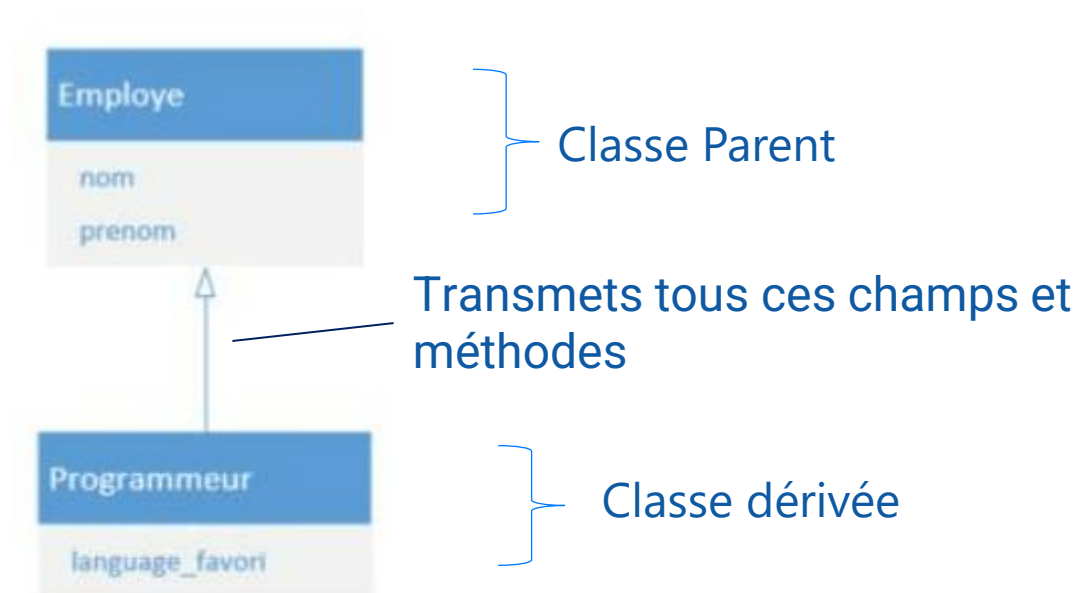
```
class Employe:
    ....liste_employe = []
    ....next_ID = 1000
    ....
    ....def __init__(self,nom,prenom) -> None:
    ....    ....self.nom = nom
    ....    ....self.prenom = prenom
    ....    ....self.ID = Employe.next_ID
    ....
    ....    Employe.next_ID += 1
    ....    ....Employe.liste_employe.append(self)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

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.


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



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



```
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.

Méthodes de classes



retour



```
class Employe:
    nb_employes = 0
    base_augmentation = 1.04

    def __init__(self, prenom, nom, salaire):
        self.prenom = prenom
        self.nom = nom
        self.salaire = salaire
        self.courriel = prenom + '.' + nom + '@gmail.com'
        Employe.nb_employes += 1

    def nom_complet(self):
        return '{} {}'.format(self.prenom, self.nom)

    def donner_augmentation(self):
        self.salaire = int(self.salaire * self.base_augmentation)
        # nous utilisons self car l'augmentation de base pourrait varier selon l'employé instancié

    @classmethod
    def from_string(cls, emp_str):
        """Constructeur pour créer un employé à partir d'une chaîne séparée avec un '-'"""
        prenom, nom, salaire = emp_str.split('-')
        return cls(prenom, nom, salaire)
```

Decorator

Méthode
de classe

Fais référence à la classe

Ex : Monopoly - Pseudocode



- > Vérifier si le « propriétaire » a bel et bien le terrain qu'on veut acheter
 - > Si oui
 - > Vérifier si on a assez d'argent
 - > Si oui
 - > Retirer le terrain de la liste des terrains du propriétaire
 - > Retirer le cash du joueur
 - > Ajouter le cash dans le montant_cash du propriétaire
 - > Ajouter le terrain de la liste des terrains du joueur
 - > Si non
 - > Écrire un message « Vous n'avez pas assez d'argent pour acheter le terrain X »
 - > Si non
 - > Écrire un message « Désolé, je ne suis pas propriétaire de ce terrain »



Ex : Monopoly – Diagramme de flux

