



Présentation ADRAR Pôle Numérique

Suivez-nous... LinkedIn  et Twitter  @Adrar_Numerique



ADRAR\o/ PÔLE NUMERIQUE

PÔLE NUMERIQUE DU CENTRE DE FORMATION ADRAR

- > SUPPORT, ADMINISTRATION SYSTEMES & RESEAUX
- > DEVELOPPEMENT D'APPLICATIONS WEB & MOBILES
- > WEBDESIGN & DEVELOPPEMENT INTERFACES
- > TRANSFORMATION NUMERIQUE DES ENTREPRISES

<http://www.adrar-numerique.com>

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com



Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

LA PROGRAMMATION ORIENTEE

OBJET : POO

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

La programmation orientée objet (POO) est une méthode de structuration de code permettant de rendre votre programme mieux organisé et plus lisible. Comme vous le savez en python TOUT est objet, c'est donc un langage propice à l'apprentissage de la POO. Quelques définitions :

- Classe : Ensemble de code regroupant des variables et des méthodes permettant de créer des objets.
- Objet : Instance d'une classe disposant d'attributs et de méthodes.
- Constructeur : Méthode qui va être utilisée pour instancier un objet.
- Assesseur (getter) : Méthode qui va être utilisée pour afficher un attribut.
- Mutateur (setter) : Méthode qui va être utilisée pour modifier un attribut.
- Héritage : Principe voulant qu'une classe hérite des propriétés de son parent. De base toutes les classes héritent de la classe Object

Suivez-nous sur LinkedIn 

Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

Assesseeur et Mutateurs

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

Nous avons vu comment créer une classe avec ses attributs et ses méthodes. Nous avons également vu comment initialiser un objet de notre classe. Et enfin nous avons vu comment redéfinir les méthodes de base.

Nous allons maintenant aborder un principe pilier de la programmation orientée objet : l'encapsulation. L'encapsulation est un principe visant à ne pas rendre tout accessible à tout le monde.

Les attributs et méthodes que nous avons déclarés jusqu'à maintenant étaient dites publiques, car accessibles simplement par tout le monde (vous n'avez qu'à faire *MonObjet.MonAttribut* ou *MonObjet.MaMethode()* pour accéder à des éléments de votre objet).

Pour déclarer un attribut ou une méthode privée il faut faire précéder son nom par « `_` ». Un élément privé ne sera pas accessible directement par votre code. Il pourra cependant être utilisé par une autre méthode de votre classe. Il faudra donc créer des méthodes, des accesseurs et des mutateurs (ou `getter` et `setter` en anglais) pour pouvoir accéder à des attributs privés.

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

Prenons l'exemple d'un distributeur de café. Lorsque vous sélectionnez votre boisson la machine vous sors le gobelet puis vous verse la boisson, vous ajoute le sucre et vous fournit une touillette.

Vous ne pouvez pas cependant demander à la machine qu'elle vous donne un gobelet, ni uniquement de vous verser le café le sucre ou la touillette. Vous pouvez par contre demander une quantité de sucre spécifique. Vous ne pouvez pas non plus visualiser la quantité de sucre ou de café restante.

Nous aurions donc, en POO, une classe *Distributeur* avec une méthode publique *demandeCafe* et des méthodes privées *sortirGobelet*, *verserCafe*, *verserSucre*, *donnerTouillette* (oui c'est un super distributeur gratuit qui n'a qu'une boisson possible).

Notre classe aurait donc quatre attributs privés *stock_gobelet*, *stock_cafe*, *stock_sucre* et *stock_touillette*.

Suivez-nous sur LinkedIn Twitter @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

Voici à quoi ressemblerait notre classe :

```
class Distributeur:

    _stock_cafe = 0
    _stock_sucre = 0
    _stock_touillette = 0
    _stock_gobelet = 0

    def __init__(self):
        print("Bonjour, je suis une machine a café gratuite mais limité")
        self._stock_sucre = 1000
        self._stock_touillette = 100
        self._stock_gobelet = 100
        self._stock_cafe = 100
```

```
def demanderCafe(self, quantite_sucre):
    self._sortirGobelet()
    if quantite_sucre > 0:
        self._verserSucre(quantite_sucre)
        self._donnerTouillette()
    self._verserCafe()
    print("Votre café est pret !")
```

```
def _sortirGobelet(self):
    print("voici votre gobelet")
    self._stock_gobelet -= 1
```

```
def _verserSucre(self, sucre):
    print("Voici votre sucre")
    self._stock_sucre -= sucre
```

```
def _donnerTouillette(self):
    print("Et voila la touillette")
    self._stock_touillette -= 1
```

```
def _verserCafe(self):
    print("Voila le café")
    self._stock_cafe -= 1
```


Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

Et le programme :

```
monDistributeur = Distributeur()  
monDistributeur.demanderCafe(8)
```

```
Bonjour, je suis une machine gratuite mais limité  
Voici votre gobelet  
Voici votre sucre  
Et voila la touillette  
Voila le café  
Votre café est pret !
```

Suivez-nous sur LinkedIn Twitter @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

Python étant un langage très permissif il est quand même possible d'accéder à nos attributs et méthodes privés par leurs nom (car comme l'a dit Guido van Rossum le fondateur de Python « *We are all consenting adults here* », voulant dire « *nous sommes tous des adultes consentant ici* »).

Dans notre déclaration d'objet précédant si j'essaye d'accéder directement à l'attribut `_stock_sucre` j'aurais bien sa valeur :

```
print(monDistributeur._stock_sucre)
```

```
992
```

Cependant ce n'est pas une pratique recommandé. Voici ce que m'affiche l'auto complétion de PyCharm quand j'écris le nom de mon objet :

```
m demanderCafe(self, quantite_sucre) Distributeur
m __init__(self) Distributeur
print print(expr)
ifn if expr is None
not not expr
```

PyCharm ne me propose pas les méthodes et attributs privés. Il ne me propose que les éléments publics.

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

Réfléchissons maintenant au technicien qui doit lui pouvoir visualiser le stock de sucre et également remplir la machine. Il est possible de définir des méthodes d'appel simple pour manipuler nos attributs (getter et setter).

Pour cela nous allons utiliser une fonction built-in de python : *property()*.

La fonction *property()* prend 4 valeur en paramètre : une fonction getter, une fonction setter, une fonction de suppression et une documentation (qui s'affichera quand on appellera la fonction *help()* sur notre objet). Nous allons donc créer ces éléments.

```
def _voirSucre(self):
    return self._stock_sucre

def _definirSucre(self, quantite):
    self._stock_sucre = quantite

def _delSucre(self):
    del self._stock_sucre

stock_sucre = property(_voirSucre, _definirSucre, _delSucre, "Voici le getter setter de l'attribut _stock_sucre")
```

Suivez-nous sur LinkedIn  Twitter  @Adrar_Numerique

... et sur le web : www.adrar-numerique.com

Grâce à cette fonction *property()* nous pouvons accéder simplement à notre attribut *_stock_sucre* :

```
monDistributeur = Distributeur()
monDistributeur.demanderCafe(8)
print("le stock de sucre est de :", monDistributeur.stock_sucre)
monDistributeur.stock_sucre += 1
print("le stock de sucre est de :", monDistributeur.stock_sucre)
del monDistributeur.stock_sucre
print("le stock de sucre est de :", monDistributeur.stock_sucre)
```

```
Bonjour, je suis une machine gratuite mais limité
Voici votre gobelet
Voici votre sucre
Et voila la touillette
Voila le café
Votre café est pret !
le stock de sucre est de : 992
le stock de sucre est de : 993
le stock de sucre est de : 0
```

EXERCICE

Classe : Elève

Attributs :

Nom : str

Prénom : str

Age : int

_Section : str

Méthodes:

Anniversaire()

_ChangerSection(section)

Vous travaillez dans une école primaire. Il y a deux types d'utilisateurs : les secrétaires et le corps enseignant. Les secrétaires peuvent ajouter des élèves au programme et déclarer l'anniversaire d'un élève. La section d'un élève est définie par son âge. Les professeurs peuvent changer de section un élève quand ils le veulent.

Rédigez dans un fichier la classe élève avec ses attributs et méthodes publiques et privées ainsi que les getter setter des attributs privés.

Rédigez un programme qui demande si l'utilisateur est secrétaire ou professeur. Si c'est un secrétaire il lui demande si il veut ajouter un élève ou déclarer un anniversaire. Les élèves sont enregistrés dans un tableau. Si c'est un professeur il lui demande quel élève il veut changer de section et il lui demande ensuite la section dans laquelle il veut le placer. Le programme tourne en continue (while True).