Programmation orientée objet (POO)

1 Introduction

Le paradigme de **programmation orientée objet** s'appuie sur les notions suivantes :

- Un **objet** est un modèle, un moule qui va permettre de créer des représentations de cet objet. Chaque représentation est une **instance**;
- Chaque instance d'objet a des valeurs définies à la construction de l'objet. Ces valeurs sont les **attributs** de l'objet;
- Un objet possède différentes fonctionnalités. Toutes les instances bénéficieront de ces fonctions. On les appelle des **méthodes**.

1.1 Objet en Python

En python, un objet est défini par le mot clef **class** suivi du nom de l'objet et des 2 points.

```
class objet:
    # attributs et méthodes de l'objet
```

Une **classe** est une structure de donnée qui permet de créer des objets.
Une **classe** permet de définir les attributs d'un objet et les méthodes qui lui sont propres. Tout ce qui est défini dans la classe doit être **indenté**.

La création d'un objet se fera par une affectation avec la classe définissant l'objet. On dit qu'on **instancie** un objet.

Exemple

Imaginons que l'on souhaite créer un objet en python représentant une voiture. On peut définir les attributs et les méthodes de notre objet comme suit:

- Les attributs sont des caractéristiques ou des propriétés de la voiture: le nombre de roues, la marque, le modèle, le nombre de portes, la couleur, etc.
- les méthodes représentent des fonctionnalités ou des actions: avancer, accélérer, démarrer, ralentir, etc.

Pour créer nos objets voitures, on définit la classe automobile:

```
class automobile:
    pass # instruction qui ne fait rien mais évite une erreur dans l'interpréteur
```

Dans l'interpréteur ou le notebook, on peut créer différentes voitures:

```
clio=automobile()
polo=automobile()
```

Remarque

Nous avons deux instances d'objets construites avec la classe automobile : **polo** et **clio**.
On fera souvent le raccourci que **clio** et **polo** sont deux objets de la classe automobile.

2 Attributs et méthodes d'un objet

Nous avons créé deux objets mais ils n'ont ni attributs ni méthodes (puisque la classe est vide). On peut ajouter des attributs et des méthodes à un objet dans l'interpréteur (mais ce n'est pas la bonne méthode).

Comme pour tout objet, l'accès aux attributs et aux méthodes se fait avec la syntaxe suivante:

```
objet.attribut # appel d'un attribut de l'objet
objet.méthode() # appel d'une méthode de l'objet, les parenthèses rappellent que c'est une fonctio
```

2.1 Les attributs d'un objet

Les attributs d'un objet permettent de stocker des valeurs pour notre objet. Ces attributs sont accessibles et peuvent être modifiés. Dans certains langages, l'accès aux attributs est protégé et nécessite des fonctions pour accéder et modifier l'attribut. En python, l'accès est libre par défaut mais on peut le protéger avec une fonction.

Reprenons notre exemple de voiture. On peut définir comme attribut le nombre de roues et le nombre de portes.

```
# la clio a 4 roues et trois portes
clio.roues = 4
clio.portes = 3

# la polo a 4 roues et 5 portes
polo.roues = 4
polo.portes = 5
polo.carburant = diesel
```

On a défini deux attributs communs pour chaque objet. Il est possible de définir un attribut pour un objet et pas pour l'autre. Bien que ce soit possible, il vaut mieux éviter de le faire et plutôt créer des objets uniformes avec les mêmes attributs pour éviter des erreurs.

2.2 Les méthodes d'un objet

Les méthodes sont des fonctions propres aux objets, ce qui implique que la fonction ne peut être appliquée qu'à l'objet. Comme l'attribut, la méthode est placée après le nom de l'objet séparée par un point : `objet.méthode()`.

Une **méthode** renvoie:

- une valeur dont le type est un nombre (int, float), un bouléen (bool), une chaine de caractère (str) ou un type construit comme la liste (list) ou le dictionnaire (dict);
- un autre objet (class);
- rien ou None tout en agissant sur un attribut : modifier, créer;
- un affichage.

En python, une méthode d'objet est définie dans la classe. Pour faire référence à l'objet, nous devons utiliser le mot clef **self** qui désignera l'objet. Ce mot clef **self** sera passé en paramètre et précisé à chaque fois que l'objet sera référencé.

Reprenons l'exemple de l'objet automobile et créons une méthode pour savoir si la voiture avance.

Exemple

```
class automobile:
    # attributs
```

```
    roues=4
    portes=3
    vitesse=0

    # méthode
    def avancer(self):
        if self.vitesse > 0:
            return True
        else:
            return False

# premier objet : polo
polo=automobile()
print("la polo a %s portes" % polo.portes)

if polo.avancer():
    print("La polo avance.")
else:
    print("La polo est à l'arrêt")

# second objet : clio
clio=automobile()
# on modifie la vitesse de la clio
clio.vitesse=50

if clio.avancer():
    print("La clio avance.")
else:
    print("La clio est à l'arrêt")
```

Si on exécute ce programme, on obtient les affichages:

```
la polo a 3 portes.
la polo est à l'arrêt.
la clio avance.
```

Remarque

Ainsi définie, la classe impose un nombre de portes égal à 3 quel que soit l'objet créé. Donc la polo et la clio ont trois portes. Dans ce cas on parle **d'attribut de classe**.

Il est possible de modifier la valeur par une affectation : `clio.portes = 5` après la création de l'objet mais cela n'est pas très optimisé !

Exercice

Créer une cellule de code Python et copier coller le code de l'exemple précédent sur la classe automobile.

1. Modifier ce programme pour créer un nouvel objet représentant une voiture du modèle *mini* avec 3 portes.
2. La polo a 5 portes et 5 roues (roue de secours). Modifier le programme en conséquence.
3. La mini roule à 70 km/h. Modifier le code.
4. Modifier l'affichage pour chaque véhicule en donnant le nombre de roues, le nombre de portes et son statut (arrêt ou mobile) en indiquant sa vitesse.

En Plus: *est-il possible d'optimiser le code pour éviter la répétition des instructions ?*

Exercice

On reprend le code précédent que l'on continue de modifier.

1. Ajouter à la classe un nouvel attribut indiquant le nombre de places disponibles. La valeur par défaut sera égale à 4.
2. Ajouter la méthode accelerer à la classe. Cette méthode augmente la vitesse de l'objet. À chaque appel de la méthode, la vitesse augmente de 10.
3. La polo, initialement à l'arrêt, roule maintenant à 90 km/h. Écrire un code qui modifie la vitesse de la polo en utilisant la méthode accelerer.
4. Écrire une méthode ralentir qui diminue la vitesse par palier de 10.
5. Écrire une méthode arreter qui modifie la vitesse du véhicule à 0.

En Plus: *est-il possible d'utiliser un paramètre à la méthode accelerer pour indiquer la valeur à ajouter à la vitesse ? De même avec la méthode ralentir.*

3 Le constructeur : méthode pour définir les attributs

En python, il existe une méthode **___init___** qui permet de définir les attributs à la création de l'objet. Cette méthode est un **constructeur** d'objet qui initialise les attributs.

Comme toute méthode, elle aura un paramètre **self** et chaque attribut sera préfixé par le mot **self**.

Exemple

On initialise les attributs de la classe automobile avec ce constructeur **___init___** :

```
class automobile:

    # constructeur de l'objet
    def __init__(self):
        self.roues=4
        self.portes=3
        self.vitesse=0

    # les autres méthode de l'objet
    def avancer(self):
        if self.vitesse > 0:
            return True
        else:
            return False
```

```
# premier objet : polo
polo=automobile()
print("la polo a %s roues et %s portes." % (polo.roues,polo.portes))
```

Le reste du code ne change pas ! Quel est alors le véritable intérêt ?

Les fonctions acceptent des **paramètres**. Cela permet donc de passer, au moment de l'appel, des valeurs en argument pour modifier les attributs de chaque objet.

Exemple

En définissant des paramètres pour les différents attributs, on peut créer des objets avec des valeurs particulières passées comme arguments:

```
class automobile:

    # constructeur de l'objet
    def __init__(self,r,p,v=0):
        self.roues = r # r est le nombre de roues
        self.portes = p # p est le nombre de portes
        self.vitesse = v # v est la vitesse initiale par défaut égale à 0

    # méthode de l'objet
    def avancer(self):
        if self.vitesse > 0:
            return True
        else:
            return False

# premier objet : polo
polo=automobile(4,5)

print("La polo a %s roues et %s portes" % (polo.roues,polo.portes))
if polo.avancer():
    print("La polo avance à %s km/h." % polo.vitesse)
else:
    print("La polo est à l'arrêt.")


# second objet : clio
clio=automobile(4,3,40)

print("La clio a %s roues et %s portes" % (clio.roues,clio.portes))
print("la clio a %s portes" % clio.portes)
if clio.avancer():
    print("La clio avance à %s km/h." % clio.vitesse)
else:
    print("La clio est à l'arrêt.")
```

Exercice

On reprend le code de l'exercice précédent.

1. Ajouter le constructeur à votre classe automobile (attention au nombre d'attributs).
2. Créer avec ce constructeur les trois automobiles *polo*, *clio* et *mini*.
3. Afficher les attributs de chaque voiture.
4. Ajouter les attributs nom et couleur à la classe automobile.
5. Recréez vos trois voitures de couleurs différentes en utilisant comme nom de variable *a1*, *a2* et *a3*

En Plus: *est-il possible d'optimiser le code pour l'affichage ?*