concept de classe (1)

Une classe est un modèle à partir duquel il est possible de créer des objets. Ce modèle définit les deux éléments constitutifs des objets, à savoir leurs attributs et leurs fonctionnalités. Un objet se construit à partir d'une classe : il est une instance de la classe.

Un objet possède des attributs et offre des fonctionnalités :
-- Un attribut est une donnée stockée dans l'objet et qui permet de le caractériser. L'ensemble des valeurs de tous les attributs d'un objet définit son état.
-- Une fonctionnalité permet d'effectuer une opération grâce à l'objet. On peut, par exemple, l'interroger pour obtenir une information, ou alors effectuer une action sur cet objet.

Les objets instanciés à partir d'une classe doivent posséder des valeurs pour chacun des attributs associés à la classe.

constructeur __init__

Pour qu'un objet puisse être construit à partir d'une classe, il est nécessaire que la classe contienne un constructeur, lequel se définit comme une fonction :
-- la fonction __init__
-- qui prend au moins un paramètre, dont le nom doit être self, et qui doit être le premier paramètre.
La référence self représente l'objet cible, c'est une variable qui contient une référence vers l'objet qui est en cours de création. Grâce à ce dernier, on peut accéder aux attributs et fonctionnalités de l'objet cible.

Dans l'exemple ci-dessous, le constructeur de la classe cycle reçoit deux paramètres radius et color, en plus du paramètre spécial self. Lorsqu'on crée des instances c1 et c2 de la classe cycle, il faut spécifier les valeurs des deux paramètres pour les objets instanciés

```
class cycle:
    def __init__(self, radius, color): #Constructeur de la classe cycle
        self.radius = radius
        self.color = color

c1 = cycle(650, "blue")  #Instanciation de l'objet c1
c2 = cycle(700, "white") #Instanciation de l'objet c2
```

Il est important de faire la différence entre les deux types de variables qui se trouvent dans le code de ce constructeur :

-- la variable self.radius représente la variable d'instance, c'est-à-dire celle associée à l'objet, qui existe à partir de la création de l'objet jusque sa destruction ;

-- la variable radius représente le paramètre reçu par le constructeur et n'existe que dans le corps de ce dernier.

Le paramètre self permet donc d'accéder aux variables d'instance, c'est-à-dire aux attributs de l'objet cible, depuis le constructeur.

Puisqu'on a initialisé des variables d'instance dans le constructeur, on peut y accéder pour n'importe quel objet créé. Si on considère l'objet c1, on va pouvoir afficher la valeur des deux variables d'instance à l'aide de l'opérateur d'accès (.).

```
print(c1.radius, c1.color)
```

Affichage obtenu : 650 blue

méthode __str__

La méthode ou fonction __str__, qui est appelée par défaut par l'instruction print( ), sert à définir ce que l'on souhaite voir s'afficher lorsqu'on fait un print de l'objet instancié. Cette méthode est essentiellement destinée à favoriser la lisibilité côté utilisateur.

```
class cycle:
    def __init__(self, radius, color): #Constructeur de la classe cycle
        self.radius = radius
        self.color = color

    def __str__(self):
        return (f"Les valeurs des attributs de l'objet instancié sont
        {self.radius} et {self.color}.")

c1 = cycle(650, "blue")
print(c1)
```

Affichage dans la console Python : Les valeurs des attributs de l'objet instancié sont 650 et blue.

En l'absence d'une définition de l'affichage via la fonction __str__, comme dans le code ci-dessous, nous obtiendrions un affichage moins convivial et explicite.

```
class cycle:
    def __init__(self, radius, color): #Constructeur de la classe cycle
        self.radius = radius
        self.color = color

c1 = cycle(650, "blue")
print(c1)
```

Affichage dans la console Python : <__main__.cycle object at 0x10ac37f28>
Le print(c1) ne renvoie plus un message explicite puisque la fonction __str__ n'a pas été implémentée.

méthode __repr__

La méthode ou fonction __repr__ est définie par le concepteur de la classe pour que les utilisateurs de la classe aient un moyen de représenter sans ambiguïté, par une chaîne de caractères, toutes les valeurs des attributs. Contrairement à la fonction __str__, la fonction __repr__ est destinée à simplement permettre la représentation des valeurs sous

une forme lisible par une machine, laquelle pourrait être, on non, une forme adaptée à l'impression. Lorsqu'aucune méthode __str__ n'est définie, c'est la méthode __repr__ qui est appelée par print pour renvoyer un résultat.

Exemple 1

```
class cycle:
    def __init__(self, radius, color): #Constructeur de la classe cycle
        self.radius = radius
        self.color = color

    def __repr__(self):
        return (f"{self.radius}, {self.color}")

c1 = cycle(650, "blue")
print(c1)
```

Affichage dans la console Python : 650, blue
Dans l'exemple 1, comme aucune fonction __str__ n'est définie, le print appelle par défaut la fonction __repr__.

Exemple 2

```
class cycle:
    def __init__(self, radius, color): #Constructeur de la classe cycle
        self.radius = radius
        self.color = color

    def __repr__(self):
        return (f"{self.radius},{self.color}")

    def __str__(self):
        return (f"Les valeurs des attributs de l'instance créée de la
classe cycle sont {self.radius} et {self.color}.")

c1 = cycle(650, "blue")
print(c1)
```

Affichage dans la console Python : Les valeurs des attributs de l'instance créée de la classe cycle sont 650 et blue.
Dans l'exemple 2, le print appelle par défaut la fonction __str__. Dans la console Python en mode interactif, taper c1 renverrai le message : >>> 650, blue