Créer des classes en Python

Programmation orienté objet

Sommaire

- Vers l’objet
- Classe, constructeur et méthodes
- Une classe un peut plus élaboré
- Composer des objets, déléguer des méthodes

Vers l'objet

Depuis les types :

- Prenons pour exemple deux liste x et y.
  - Elles possèdent un certains nombre de caractéristiques communes.
  - Elles ont par exemple plusieurs méthodes qui fonctionnent de façon identique pour l'une comme pour l'autre (sort(), append(), …).
- A l'inverse, une chaîne de caractère z
  - Même si elle contient les mêmes valeurs que x ou y, ne possède pas les méthodes sort() ou appende().
  - En revanche elle dispose d'une méthode upper().

Vers l'objet

Depuis les types :

```
x = ['2', 'a', '7']
y = ['8', 'k', 't']

x.append('5')
print(x)
y.append('5')
print(y)
```

```
['2', 'a', '7', '5']
['8', 'k', 't', '5']
```

```
z = "8kt"
z.append('5')
print(z)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-2-073b47ad48d1> in <module>
      1 z = "8kt"
----> 2 z.append('5')
      3 print(z)

AttributeError: 'str' object has no attribute 'append'
```

Vers l’objet

Depuis les types :

- x, y, et z sont trois objets distincts.
- x et y sont deux instances d’une même classe (les listes) définie par leur contrat commun :
  - La façon commune de pouvoir les créer.
  - Le type d’éléments qu’elles peuvent stocker.
  - Les méthodes qui fonctionnent de façons identique.

En se basant sur ça on peut émettre l’idée que **créer des classes revient à inventer des types**.

Tandis qu’**instancier des objets revient à créer des exemplaires concrets de ces types** en leur associant des valeurs.

Vers l'objet

<u>Depuis les fonctions :</u>

Une autre façon de se représenter ce qu'est une classe est par exemple de l'**extrapoler à partir du concept de fonction**.

Toute opération qu'effectue une fonction lorsqu'on l'appelle dépend :

- Des arguments qu'on lui passe explicitement.
- Des arguments que lui passe implicitement l'environnement.

```
def f(x):
    return x + a
```

```
a = 1
print(f(12))
```

```
13
```

```
a = -6
print(f(12))
```

```
6
```

Un objet peut être vus comme la données **d'un ensemble de fonctions** (une librairie) ET **de variables utilisées** par les fonctions mais préservées d'un appelle à l'autre.

Classes, constructeurs et méthodes

Avec les objets comme avec les fonctions on peut travailler en deux temps :

- D'abord on définit la classe.
- Ensuite on peut y faire appelle autant de fois que désiré.

Exemple :

On souhaite créer un objet qui :

- Stocke des caractères comme une chaîne.
- Soit modifiable comme une liste.
- Dispose de la méthode upper() propre aux chaines.

Classes, constructeurs et méthodes

- 1ère bloc :
  Définition de la classe introduite par l'instruction « class » à l'image de « def » pour les fonctions.
- 2ème bloc :
  Début du programme proprement dis où l'on construit l'objet s1 et on appelle sur lui diverses méthodes.

```
class superstring :
    def __init__(self, chaine):
        self.ch = chaine
    def ajoute(self, char):
        self.ch += char
    def insert(self, char, i):
        self.ch = self.ch[:i] + char + self.ch[i:]
    def upper(self):
        self.ch = self.ch.upper()
    def __str__(self):
        return self.ch
```

```
s1 = superstring("Ecoutez bien c'est important")
s1.ajoute(" ce que je dis !")
s1.insert(" très", 18)
s1.upper()
print(s1)
```

```
ECOUTEZ BIEN C'EST TRÈS IMPORTANT CE QUE JE DIS !
```

Classes, constructeurs et méthodes

Dans la définition de la classe nous utilisons **« def » pour définir les trois méthodes** ajoute(), insert() et upper().

Pour définir une méthode on procède exactement comme pour définir une fonctions à la différence que :

- Cela se fait dans une classe
- Nous ajoutons à chaque fois l'argument « **self »**.
  Celui-ci sert à faire référence à l'objet lui-même de façon à pour le modifier dans la méthode.

Classes, constructeurs et méthodes

Dans notre exemple la première méthode définit est une méthode très particulière appelé **le constructeur** :

- Celle-ci s'appelle toujours **__init__**() quel que soit l'objet.
- C'est elle qui est toujours **implicitement appelé** (la manière dont la classe va traiter l'objet par défaut sans appeler aucune autres méthodes).

Ici sont effet, outre de créer l'objet lui-même, est de créer et affecter une variable, « ch », qui va servir à stocker l'argument « chaine ».

Par conséquent cette variable sera disponible pour toutes les autres méthodes.

Classes, constructeurs et méthodes

Une autre méthode spécifique : **__str__()**

Celle-ci est destinée à expliquer ce qui doit être renvoyé si on demande d'**exprimer l'objet sous forme de chaine de caractère** par exemple avec un « print ».

Ci-contre un exemple de la classe sans la méthode __str__().

```
class superstring :
    def __init__(self, chaine):
        self.ch = chaine
    def ajoute(self, char):
        self.ch += char
    def insert(self, char, i):
        self.ch = self.ch[:i] + char + self.ch[i:]
    def upper(self):
        self.ch = self.ch.upper()
```

```
s1 = superstring("Ecoutez bien c'est important")
s1.ajoute(" ce que je dis !")
s1.insert(" très", 18)
s1.upper()
print(s1)
```

```
<__main__.superstring object at 0x0000017C282ECE88>
```

Classes, constructeurs et méthodes

Exercice :

- Ajouter une méthode capsdown() qui passe en minuscule les éléments de la classe « superstring ».
- Ajouter une méthode tri() qui trie les mots de la classe « superstring » par ordre alphabétique. Utiliser les méthodes split(), sorted() et join().
- Modifier la méthode __str__ de façon à avoir un affichage de la forme « type : superstring, content : ECOUTEZ BIEN C'EST TRÈS IMPORTANT CE QUE JE DIS ! ».

Une classe un peut plus élaboré

Dans cette classe nommé formulaire, l'or de la définition du constructeur nous instancions trois **attributs** :

- Nom
- Prenom
- Naissance

Ensuite nous disposons de trois méthodes :

- age()
- majeur() qui détermine si l'individus est majeur.
- memeFamille() qui compare les attributs nom de deux formulaire.

```
class formulaire:
    def __init__(self, nom, prenom, anneeNaissance):
        self.nom = nom.upper()
        self.prenom = prenom.upper()
        self.anneeNaissance = anneeNaissance
    def age(self):
        return 2020 - self.anneeNaissance
    def majeur(self):
        return self.age() >= 18
    def memeFamille(self, formulaire):
        return self.nom == formulaire.nom
```

```
jd = formulaire('Doe', 'John', 2005)
ad = formulaire('doe', 'Alice', 2000)

print(jd.majeur())
print(ad.majeur())
print(jd.memeFamille(ad))
```

```
False
True
True
```

Une classe un peut plus élaboré

Exercice :

- Ajouter une méthode qui vérifie si deux formulaires renvoient à la même personne.
- Faire en sorte qu'on puisse afficher les formulaires sous la forme [nom, prenom, anneeNaissance]. Créer une liste de formulaire et triez la par ordre de naissance.

Composer des objets, déléguer des méthodes

Il est possible de définir une classe dont les **attributs appartiennent eux-mêmes à une autre classe** que nous aurons définie au préalable. C'est en fait une pratique très générale.

- Un **programme orienté objet** est le plus souvent une **collection de classes**, disposant chacune d'attributs et de méthodes.
- Ces attributs peuvent être des objets de la librairie standard comme « str » ou « list ». Mais le plus souvent figurerons aussi des instances des **autres classes du programme**.

Composer des objets, déléguer des méthodes

Pour bien comprendre la manière de programmer orienté objet je vous propose de partir d'un exemple pas du tout orienté objet et de le transformer pas à pas :

Il s'agit d'un jeu de bataille un peu simplifié (en cas d'égalité entre les valeurs on utilise l'ordre sur les couleurs).

- Au début de la partie, on initialise une main de 7 cartes pour chacun des deux joueurs
- Puis à chaque manche on pioche une carte par joueur
- Enfin, on compare chacune des cartes piochées entre elles.
- Le gagnant est celui qui remporte le plus de manches.

Composer des objets, déléguer des méthodes

Première partie du code :

On simule un paquets de carte et deux mains vides.

```
from random import randrange

# On initialise toutes les valeurs et couleurs
# que peuvent prendre les cartes
valeurs = [i for i in range(1, 11)]
couleurs = [i for i in range(1, 5)]

# On choisi le nombre de tour que va durée la partie
# et on initialise le score à 0.
nbTours = 7
score = 0

# Enfin on crée une liste de tuple
# pour représenter le paquet de cartes
paquet = [(v, c) for v in valeurs for c in couleurs]
main1, main2 = [], []
```

Composer des objets, déléguer des méthodes

Deuxième partie du code :

Chacun des deux joueurs tire une carte aléatoirement, elle est supprimé du paquet et on répète l'opération pour chaque tour.

```
for i in range(nbTours):
    # Le joueur 1 tire une carte aléatoirement dansle paquet
    x = paquet[randrange(len(paquet))]
    # La carte est ajouté à la main du joueur 1
    main1.append(x)
    # La carte tiré est supprimé du paquet
    paquet.remove(x)
    # Idem pour le joueur 2
    y = paquet[randrange(len(paquet))]
    main2.append(y)
    paquet.remove(y)
```

Composer des objets, déléguer des méthodes

<u>Troisième partie :</u>

Pour chaque carte tirée on détermine qui du joueur 1 ou 2 à pris l'avantage en faisant + 1 au score si c'est le joueur 1 si non -1.

```
for i in range(nbTours):
    if main1[i][0] > main2[i][0] or (
    main1[i][0] == main2[i][0] and main1[i][1] > main2[i][1]):
        score += 1
    else :
        score -= 1
```

```
print("Vainqueur : " + ("joueur 1" if score > 0 else "joueur 2"))
```

```
Vainqueur : joueur 2
```

Composer des objets, déléguer des méthodes

Travaille à faire :

- Créer une fonction plusGrandQue() qui doit faire la comparaison entre deux cartes (représenté par des tuples) et donc remplacer la troisième partie du code.
- Créer une fonction piocher() qui doit sélectionner l'ensemble des carte tiré par un joueur et donc remplacer la deuxième partie du code .
- Créer une classe « carte » ayant les attribut couleur et valeur. Elle devra être appelé dans la fonction plusGrandQue().

Composer des objets, déléguer des méthodes

Travaille à faire :

Créer une classe « partie » qui nous permettra d'effectuer différentes partie avec des jeux différents. C'est-à-dire que l'on peut choisir :

- Un nombre de couleurs autre que 4.
- Un nombre de valeurs autre que 10.
- Un nombre de tour autre que 7.

A vous de trouver :

- Quelle sont les attribut à définir dans la méthode __init__
- Quelle autre méthode définir et quelle seront ces attributs.