Variables et méthodes de classe

Jusqu'à présent, nous avons surtout utilisé des variables et méthodes d'instance.

- Les variables d'instance sont propres à chaque instance de la classe et qui maintenent chacune de ces variables dans des emplacements mémtoires différents.
- Les méthodes d'instance sont des méthodes qui accèdent à des variables d'instances et doivent impérativement débuter par le paramètre `self`.

Variables de classe (variables statiques)

Il arrive parfois que toutes les instances doivent partager certaines variables qui ne concernent aucune instance particulière mais la classe dans son ensemble. Nous avions déjà utilisé ceci pour compter le nombre d'aliens qui ont atterri dans le code :ref:`code-space_invader_corrige` dans la classe `Alien` :

Syntaxe

Au contraire des variables d'instance qui sont définies dans le constructeur avec la syntaxe `self.variable_instance`, les variables de classe sont définies en dehors de toute méthode, au début de la définition sans utiliser le `self` :

```
class MaClasse(object):

    # définition des variables de classe
    variable_de_classe0 = 0
    variable_de_classe1 = "salut"

    def __init__(self):

        # définition des variables d'instance
        self.variable_instance1 = "truc"
        self.variable_instance2 = "machin"
```

Pour accéder aux variables de classe, il n'est pas nécessaire de créer une instance de la classe : suffit d'utiliser la notation `NomClasse.variable` :

```
>>> MaClasse.variable_de_classe0
0
>>> MaClasse.variable_de_classe1 = "ohé!"
>>> MaClasse.variable_de_classe1
'ohé!'
```

Exemple

Il est par exemple possible d'utiliser des variables de classe pour regrouper des variables dans une classe au lieu de les laisser polluer l'espace de noms global. Il n'est pas nécessaire de créer une instance de la classe pour accéder aux variables de classes.

```
1 import math
2
3 # ---------------- class Physics ----------------
4 class Physics():
5     # Avagadro constant [mol-1]
6     N_AVAGADRO = 6.0221419947e23
```

```
 7     # Boltzmann constant [J K-1]
 8     K_BOLTZMANN = 1.380650324e-23
 9     # Planck constant [J s]
10     H_PLANCK = 6.6260687652e-34;
11     # Speed of light in vacuo [m s-1]
12     C_LIGHT = 2.99792458e8
13     # Molar gas constant [K-1 mol-1]
14     R_GAS = 8.31447215
15     # Faraday constant [C mol-1]
16     F_FARADAY = 9.6485341539e4;
17     # Absolute zero [Celsius]
18     T_ABS = -273.15
19     # Charge on the electron [C]
20     Q_ELECTRON = -1.60217646263e-19
21     # Electrical permittivity of free space [F m-1]
22     EPSILON_0 = 8.854187817e-12
23     # Magnetic permeability of free space [ 4p10-7 H m-1 (N A-2)]
24     MU_0 = math.pi*4.0e-7
25
26
27 c = 1 / math.sqrt(Physics.EPSILON_0 * Physics.MU_0)
28 print("Speed of light (calulated): %s m/s" %c)
29 print("Speed of light (table): %s  m/s" %Physics.C_LIGHT)
```

Méthodes de classe

```
class Compteur(object):

    objets_crees = 0  # le compteur vaut 0 au départ

    def __init__(self):
        Compteur.objets_crees += 1

    @classmethod
    def count(cls):
        print("Jusqu'à présent,", cls.objets_crees, "objets ont été créés.")
```

Utilisation

Voici l'évolution de la variable de classe `Compteur.objets_crees` affiché à l'aide de la méthode de classe `Compteur.count()`

```
>>> Compteur.count()
Jusqu'à présent, 0 objets ont été créés.
>>> a = Compteur()
>>> Compteur.count()
Jusqu'à présent, 1 objets ont été créés.
>>> b = Compteur()
>>> Compteur.count()
Jusqu'à présent, 2 objets ont été créés.
>>> a = Compteur()
>>> Compteur.count()
Jusqu'à présent, 3 objets ont été créés.
```

Il est paradoxalement également possible d'appeler une méthode de classe à l'aide d'une instance particulière :

```
>>> a.count()
Jusqu'à présent, 3 objets ont été créés.
```

Méthodes statiques

Les méthodes statiques n'utilisent ni les attributs d'instance ni les attributs de classe sont des **méthodes statiques** et elles ne nécessitent pas le paramètre `self` en première position. De plus, elles sont décorées à l'aide du décorateur `@staticmethod` qui doit figurer sur la ligne précédent la définition de la méthode :

```
# ---------------- class OhmsLaw ----------------
class OhmsLaw():
    @staticmethod
    def U(R, I):
        return R * I

    @staticmethod
    def I(U, R):
        return U / R

    @staticmethod
    def R(U, I):
        return U / I

r = 10
i = 1.5

u = OhmsLaw.U(r, i)
print("Voltage = %s V" %u)
```

À retenir

- Les variables de classe sont partagées par toutes les instances d'une même classe et n'occupent qu'un espace mémoire commun pour toutes les instances. On y accède avec la syntaxe

  ```
  NomClasse.variable
  ```

- Une application type des variables de classe consiste à maintenir le nombre d'instance de la classe en question (par exemple le nombre d'acteurs de cette classe dans un jeu).
- Les méthodes de classe ne peuvent accéder qu'aux variables de classe et ne peuvent donc pas accéder aux variables d'instance. Leur définition suit la syntaxe :

  ```
  class MaClasse(object):

      @classmethod
      def methode_de_classe(cls, arg1, arg2):
          # code de la fonction
          pass
  ```

et utilisent par convention le parampètre `cls` au lieu de `self` en première position. Elles sont appelées avec la syntaxe

```
MaClasse.methode_de_classe(arg1=val1, arg2=val2)
```

- Les méthodes statiques ne accéder ni aux variables d'instance ni aux variables de classe. Leur définition suit la syntaxe :

```
class MaClasse(object):

    @staticmethod
    def methode_de_classe(arg1, arg2):
        # code de la fonction
        pass
```

et n'utilisent pas le paramètre `self` en première position. Elles sont appelées avec la syntaxe

```
MaClasse.methode_de_classe(arg1=val1, arg2=val2)
```