RÉSUMÉ 24 - LES MÉTHODES DANS LES OBJETS

Lien vers l'activité : Les méthodes dans les objets
www.infoforall.fr - Dernière modif. : 26 10 2020
Ce résumé regroupe et complète les connaissances abordées sur les activités objets 23 et 24.

23.1 - DESCRIPTION D'UNE CLASSE

EXEMPLE DE CLASSE

```
 1  import random as rd
 2
 3  class Personnage :
 4      "Ceci est une classe permettant de créer un personnage de RPG"
 5
 6      niveau_max = 20
 7      classes = ['Humain', 'Jedi', 'Stormtrooper', 'Rebelle']
 8
 9      def __init__(self, nom, prenom, niveau=1, classe='Jedi') :
10          # Liste des attributs
11          self.nom = nom #   affectation directe
12          self.prenom = prenom   # affectation directe
13          self.niveau = 0   # juste pour montrer qu'il existe
14          self.classe = ''   # juste pour montrer qu'il existe
15          # Initialisation des derniers attributs
16          self.set_niveau(niveau)   # affectation par un "setter"
17          self.set_classe(classe)   # affectation par un "setter"
18
19      def set_niveau(self, niveau) :
20          if (type(niveau) != int) :
21              self.niveau = 1
22          elif niveau > Personnage.niveau_max :
23              self.niveau = Personnage.niveau_max
24          elif niveau < 0 :
25              self.niveau = 0
26          else :
27              self.niveau = niveau
28
29      def set_classe(self, classe) :
30          if classe in Personnage.classes :
31              self.classe = classe
32          else :
33              self.classe = Personnage.classes[0]
34
35      def get_niveau(self) :
36          return f"{self.classe} de niveau {self.niveau}"
37
38      def attaquer(self, cible) :
39          if (rd.randint(-5,5) + self.niveau) > cible.niveau :
40              cible.perdre_niveau()
41
42      def perdre_niveau(self, quantite = 1) :
43          self.set_niveau(self.niveau - quantite)
```

VOCABULAIRE À CONNAÎTRE : DIFFÉRENCE ENTRE CLASSE ET OBJET

- La **classe** est le moule qui va permettre de construire les **objets**. Elle contient les informations nécessaires à cette création.
- L'**objet** est le nom donné à la structure de données créée à l'aide des informations de la **classe**.
- on déclare une classe à l'aide du mot clé **class** suivi du nom de la classe qui doit **commencer par une Majuscule**.
- une Classe est un conteneur. Les choses contenues se nomment les **membres**.
  - une Classe peut contenir des **attributs** (les variables).
  - une Classe peut contenir des **méthodes** (les fonctions).

VOCABULAIRE À CONNAÎTRE : ATTRIBUT

Attributs : les attributs sont les variables contenues dans une Classe. Deux catégories :

- les **variables d'instance** : ce sont des **attributs** rattachés à un objet déjà concu. Voir les lignes 11-14. Si la Classe est bien écrite, la liste des variables d'instance devrait se trouver intégralement dans la méthode **__init__**.
  - Exemple : **nom** est un **attribut** et une **variable d'instance**.
  - Pour la lire de quelque part dans le programme, `nom_de_objet.nom`.
  - Pour la lire de l'intérieur d'une méthode, **`self.nom`**.

- les **variables de Classe** : ce sont les **attributs** rattachés directement à la Classe et pas à l'une de ses instances. Elles sont déclarées **directement** dans le corps de la Classe. Ne tentez pas de la modifier depuis un objet : cela va créer un attribut

d'instance à la volée et pas modifier la variable de classe !
    - Exemple : **niveau_max** et **listes** sur les lignes 6-7 sont des **attributs** et des **variables de Classe**.
    - Pour la lire de quelque part dans le programme, il suffit de taper **`Personnage.niveau_max`**.

VOCABULAIRE À CONNAÎTRE : MÉTHODE

- **Méthodes** : les méthodes sont les fonctions contenues dans une Classe.

- une méthode doit **obligatoirement** avoir au moins un paramètre : l'identifiant-mémoire de l'objet. Ce ramètre est **automatiquement** fourni lors de l'appel.
    - Python : nommé **self** par convention.

- Exemple : **perdre_niveau**
    - Appel depuis quelque part dans le programme, **`nom_de_objet.perdre_niveau(2)`**.
    - Appel depuis l'intérieur d'une méthode, **`self.perdre_niveau(2)`**.

23.2 - INSTANTIATION AVEC LE CONSTRUCTEUR

CONSTRUCTEUR

Vocabulaire : une fois la classe en mémoire, on peut créer des **objets** (ou **instances** de cette Classe) à l'aide d'une fonction **Constructeur**.

En Python, on active le **Constructeur** simplement en utilisant le nom de la classe suivi des parenthèses.

```
01      heros = Personnage(nom="Skywalker", pre="Luke", niv=8, cla="Jedi")
```

Que fait le constructeur ?
- il crée la structure de données permettant de stocker l'objet et réserve une adresse-mémoire
- il fait appel à la méthode spéciale **__init__** qui va initialiser certaines valeurs. On lui fournit souvent des **paramètres par défaut** : cela permet d'alléger son appel lorsque certains paramètres valent souvent la même chose à chaque instantiation.
- Une fois l'objet créé et initialisé, le **Constructeur** renvoie l'identifiant-mémoire de l'objet (via un **return**). Il peut ainsi être stocké. Ici, dans **heros**.

MÉTHODE SPÉCIALE __INIT__

VOCABULAIRE : on nomme souvent **__init__ méthode-constructeur** par abus de langage. Il s'agit à proprement parler d'une **méthode-initialisateur**.

BUTS : INITIALISER ET REGROUPER : L'un des buts de la méthode spéciale **__init__** est d'initialiser certaines variables de l'objet. Mais en réalité, elle sert aussi à REGROUPER les noms des attributs que nous allons utiliser sur cet objet : cela évitera à quelqu'un de vouloir créer un attribut qui est déjà utilisé.

23.3 - UTILISATION DE L'OBJET

CRÉATION D'ATTRIBUT À LA VOLÉE

- On **peut le faire** mais on **évitera** de le faire : si vous voulez utiliser un attribut, vous devez le déclarer dans la méthode spéciale **__init__** avec une valeur de base.

MODIFICATION D'ATTRIBUTS

- Depuis une méthode : **`self.nom = 'bob'`** par exemple.
- Depuis l'extérieur : si **heros** fait référence à une instance de **Personnage** : **`heros.nom = 'bob'`** par exemple.

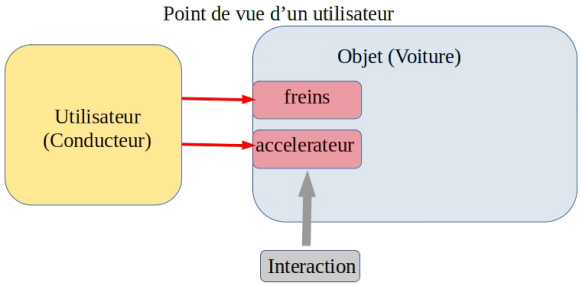
UTILISATION DE MÉTHODES

- Pour utiliser une méthode, il faut placer l'identifiant-mémoire d'une instance suivi d'un point et du nom de la méthode.
- Exemple 1 : si vous tapez **`heros.set_niveau(5)`** cela veut dire d'aller activer la méthode **set_niveau** présente dans l'objet d'identifiant **heros**. Le paramètre **self** va donc recevoir l'identifiant-mémoire de **heros**.
- Exemple 2 : si vous tapez **`self.set_niveau(5)`** cela veut dire d'aller activer la méthode **set_niveau** présente dans l'objet d'identifiant **self** : vous allez donc agir sur l'objet en cours d'utilisation.

23.4 - INTERFACE ET ENCAPSULATION

INTERFACE : en Programmation orientée Objet (POO), on ne permet pas à l'utilisateur de manipuler directement les attributs de vos objets. On **limite les capacités de l'utilisateur** à travers **une interface** limitée : on lui permet donc simplement d'utiliser certaines **méthodes d'interface**.

ENCAPSULATION : le fait de limiter l'accès au code interne et d'imposer l'utilisation de l'interface se nomme l'encapsulation. Cela permet :
- de **garantir la solidité** des objets car on ne permet pas à l'utilisateur de manipuler directement les données internes
- de **cacher la complexité** interne du code : l'utilisateur doit simplement apprendre à utiliser les méthodes d'interface, sans se soucier de comme elles fonctionnent.



PROTOTYPE : par contre, lorsqu'on veut juste créer un prototype permettant de vérifier la faisabilité d'une idée, on peut bien entendu se passer de cette notion d'encapsulation. L'encapsulation n'intervient que lorsqu'on délivre un produit fini à un utilisateur. En début de projet, en DS ou au BAC, on peut donc aller lire et modifier tranquillement les attributs, sauf mention contraire.

23.5 - UTIILSATION DE LA CLASSE DE L'INTRODUCTION

Imaginons qu'on veuille créer 2 personnages et les faire se combattre tour à tour jusqu'à que l'un d'entre eux n'ai plus de niveau. J'ai ici regroupé points de vie et niveau pour raccourcir le code.

```
1  a = Personnage("A", "Alice", 15)
2  b = Personnage("B", "Bob", 13, "Stormtrooper")
3  while not (a.niveau <=0 or b.niveau <= 0) :
4      a.attaquer(b)
5      b.attaquer(a)
6  if a.niveau <= 0 :
7      print("Le premier personnage est sonné !")
8  else :
9      print("Le deuxième personnage est sonné !")
```

On voit que la ligne 6 ne respecte pas l'encapsulation : on va lire le contenu d'un attribut directement depuis l'extérieur de l'objet.