Initiation à Python - leçon 5.1

---
s2

Dans cette séquence, nous allons découvrir la programmation objet avec Python.
Une précision : cette leçon n'a pas la dimension d'un vrai cours de programmation Objet. Elle a pour simple objectif de montrer :

- comment on crée une classe avec Python,
- comment on l'instancie,
- d'introduire les opérations principales entre instances,
- ainsi que le mécanisme d'héritage entre classes.

Quelques liens bibliographiques vous permettront de poursuivre cette initiation.

---
s3

Nous l'avons dit plusieurs fois : Python est un langage de programmation orienté objet. Revenons une nouvelle fois sur cette notion d'objet.

En informatique, un objet représente un concept. Reprenons l'exemple des avions. Un objet avion possède des caractéristiques qu'on appelle attributs, tels que par exemple sa vitesse et le nombre de sièges passagers ; un objet avion a d'autres caractéristiques que l'on appelle méthodes et que l'on peut comprendre comme des fonctions permettant de modifier ses attributs, comme ici la fonction accélère que l'on doit bien entendu définir et qui on le devine va changer l'attribut vitesse.

Pour l'ordinateur, cet objet représente un programme qui s'exécute. Ce programme contient toutes les informations liées à cet objet, attributs et méthodes.

Jusqu'à présent nous avons utilisé plusieurs objets simples qui font partie de Python, telles les chaînes de caractères. Lorsque l'interpréteur exécute la chaîne "bonjour", Python reconnaît l'instruction comme une chaîne de caractères et crée dans la mémoire de l'ordinateur un objet de type chaîne de caractères. Cet objet contient d'emblée plusieurs informations : d'abord un attribut qui contient l'information "bonjour", pour simplifier appelons cet attribut valeur. La valeur de ce nouvel objet est "bonjour". Mais également, puisque cet objet est de type chaîne de caractères, Python lui attribue toutes les méthodes associées aux objets de type chaîne de caractères - c'est-à-dire un jeu de fonctions qui va pouvoir s'appliquer à cet objet.
Parmi ces méthodes, on trouve par exemple la méthode upper() qui si elle est appliquée à la chaîne "bonjour" va réécrire sa valeur en majuscules. Comment utilise-t-on ces méthodes ? Tout simplement en les accolant à un objet et en les faisant précéder d'un point, comme dans l'exemple présenté. Le point est une manière générique d'appeler une méthode sur un objet.

Nous avons également utilisé bien d'autres objets Python : les entiers, les flottants, les listes, les tuples, les fonctions, etc. En réalité, dans Python "tout" est Objet.

---
s4

Se pose naturellement la question de créer et de manipuler avec Python des objets qui n'existeraient pas au départ dans Python.

Reprenons l'exemple des avions. Tous les avions possèdent des caractéristiques qu'on appelle attributs, tels que par exemple sa vitesse et le nombre de sièges passagers ; les avions ont d'autres caractéristiques que l'on appelle méthodes et que l'on peut comprendre comme des fonctions permettant de modifier ses attributs, comme ici la fonction accélère que l'on doit bien entendu définir et qui on le devine va changer l'attribut vitesse.

Ces attributs et méthodes communes à tous les objets de type avion définissent ce que l'on appelle une classe. Une classe est une façon abstraite de représenter de façon tout un ensemble d'éléments, ici, c'est la classe Avion. Une classe est donc une façon de définir et de manipuler au sein d'un programme informatique un ensemble d'objets possédant des caractéristiques communes, appartenant à une même famille.

Plusieurs classes reliées entre elles permettent de définir, de modéliser une entité plus ou moins complexe : un contexte métier, le fonctionnement d'un système, d'une organisation, un processus, etc.

Dans Python, et dans les autres langages objet, il est possible de créer de nouvelles classes. Voyons comment.

---
s5

Dans Python, la classe Avion peut être définie de la façon suivante. D'emblée, voici l'instruction qui permet de créer une nouvelle instance de cette classe, c'est-à-dire un nouvel avion.

Examinons ce code.

D'abord le mot-clé class qui précise que l'on est en train de définir une classe Python.

Cette classe Avion contient deux fonctions :

- la fonction init()
- la fonction accelere()

La première fonction s'appellera toujours __init__ (syntaxe un peu étrange, mais qu'il faut impérativement respecter et sur laquelle nous reviendrons). Cette fonction init() sert - si elle existe - à définir la valeur des attributs de l'objet avion qui vient d'être créé et sera appelée à chaque fois que la classe Avion sera appelée. On l'appelle aussi le constructeur, car elle construit un nouvel objet. Que fait cette fonction. Deux attributs sont définis : l'attribut vitesse et l'attribut nombreDeSieges ; le constructeur leur affecte les valeurs qui lui sont passées en argument.

Une petite remarque. Les trois arguments acceptés par la méthode __init__ sont self, v et nb, tandis que seuls deux arguments sont passés à la classe Avion au moment de la création de l'instance a1 (flèche à ce moment). Que représente cet argument self ?

Pour accéder aux attributs de l'objet avion à partir de la méthode __init__ nous avons besoin d'une référence vers cet objet qui vient d'être créé. Dans Python, par convention, cet objet est appelé self ; la ligne self.vitesse = v signifie donc en quelque sorte :
"affecte à l'attribut vitesse de moi-même la valeur v transmise en argument".
Autre convention : quand une méthode est appelée, une référence vers l'objet qui vient d'être créé est toujours passée en tant que premier argument.

Voici comment nous pouvons accéder aux attributs d'un objet.

Revenons à la deuxième fonction. La classe avion possède deux attributs et une méthode, la méthode accelere(). C'est la deuxième fonction définie dans la classe Avion. Au moment d'être utilisée, cette méthode ne possède pas d'argument ; pourtant, toujours en vertu de la convention évoquée plus haut et afin de pouvoir accéder aux attributs de l'objet concerné, une référence vers cet objet est passée en tant qu'argument.

Que fait cette méthode ? Elle incrémente l'attribut vitesse de 10. Comme l'exécution nous permet de le vérifier.

---
s6

Après cette première illustration de la création et l'utilisation d'un objet, un petit mot sur l'intérêt de cette approche.
Les objets associent des caractéristiques (les attributs) *et* les méthodes manipulant ces attributs. La manipulation d'un objet (et donc de ses caractéristiques) se fait à travers une syntaxe simple (ici, par exemple : a1.vitesse ou a1.accelere() ; toute personne utilisant l'objet a juste besoin de connaître les attributs et les méthodes qui sont accessibles depuis l'extérieur de l'objet grâce à la syntaxe (on les appelle les méthodes publiques) ; la manière dont ces opérations sont exécutées à *l'intérieur* de l'objet devient juste un détail de l'implémentation. C'est ce que l'on appelle l'encapsulation. Ceci facilite la réutilisation de cette classe dans une autre partie du programme et sa mise à jour, à condition de conserver le nom des attributs et

des méthodes publiques. L'ensemble constitué par les attributs et les méthodes d'une classe est appelé interface.

---
s7

Un autre intérêt de la programmation objet est que l'on peut passer tout un objet comme argument d'une fonction au lieu de simplement passer des données.

Comme dans l'exemple présenté, où c'est bien tout l'objet a1 (attributs et méthodes) qui est passé à la fonction accelereAvionEtAffiche().

---
s8

Quelques remarques.

- Une fois une classe déclarée, contrairement aux langages à typage statique, il est toujours possible en Python de déclarer de nouveaux attributs ou méthodes, soit à une instance particulière, soit à la classe comme dans l'exemple présenté.
- La fonction dir() permet de lister toutes les méthodes de la classe, y compris celles qui ont été créées automatiquement par Python (et dont l'étude dépasse le cadre de cette leçon).

---
s9

Quelques éléments concernant les attributs privés. Les attributs privés sont des attributs dont on ne souhaite pas qu'ils puissent être modifiés directement par le programmeur.

Prenons un exemple. Considérons que nous souhaitions réaliser une classe qui gère un compte bancaire comme dans l'exemple présenté.

Cette définition d'un compte pose un problème. Les attributs de la classe compte ne sont pas protégés. N'importe quel utilisateur peut modifier l'état du compte.

---
s10

Par convention, pour avertir le programmeur dles attributs que l'on souhaite rendre privés, on les préfixe par le signe _ et on délègue leur modification à des méthodes spécifiques. Attention,

c’est une simple convention : un nom préfixé par un tiret bas doit être considéré comme une partie non publique.

---
s11

Enfin, nous allons dire un mot sur l’héritage.

L'héritage est un mécanisme permettant de définir une nouvelle classe à partir d'une classe existante en lui conférant les propriétés et les méthodes de la première, comme ici la Classe VehiculePrioritaire hérite d’une classe plus générale appelée Voiture.

De cette façon, les classes héritées forment une hiérarchie descendante, au sommet de laquelle se situe la classe de base.

Si la classe Classe2 hérite de la classe Classe1 :

- Classe2 est UNE sous-classe de Classe1 ;
- Classe1 est UNE super-classe de Classe2, car, en effet, en Python, à la différence de certains autres langages comme Java, une classe peut hériter de plusieurs autres classes (héritage multiple).

---
s11

Prenons un exemple pour voir comment mettre cela en oeuvre.

Nous avons ici une classe Vehicule avec un attribut et une méthode.

Nous choisissons une deuxième classe, la classe VehiculePrioritaire qui hérite de Vehicule.

Vehicule est ici la super classe de VehiculePrioritaire ; VehiculePrioritaire est une sous-classe de Vehicule.

Tout objet de type VehiculePrioritaire est aussi de type Vehicule. le contraire n’est évidemment pas vrai.

Comment mettre en oeuvre cet héritage ?

D’abord dans la définition de la classe VehiculePrioritaire, on précise que c’est un objet de type Vehicule.
Ensuite, le constructeur de VehiculePrioritaire appelle le constructeur la classe parent en lui passant en argument l’attribut requis, puis il définit les attributs supplémentaires - attributs que possède la classe VehiculePrioritaire (et qui en quelque sorte la particularisent) et que ne

possède pas la classe parente : ici il s'agit de l'attribut feuAllume. VehiculePrioritaire possède par construction les méthodes de Vehicule ; ici VehiculePrioritaire possède une méthode supplémentaire, la méthode allumeFeu().

Voici quelques lignes de code que vous pouvez reproduire et qui permettent d'utiliser ces classes.

---
s13

Enfin, quelques références utiles pour poursuivre la conception et la programmation OO avec Python.