Programmation 38

Création d'objets avec Python

I – Rappels

Les types simples font référence à un contenu unique.

Les types construits sont des conteneurs stockant plusieurs contenus.

Les objets sont des conteneurs stockant plusieurs variables (nommées attributs) et plusieurs fonctions (nommées méthodes). La syntaxe typique est **`objet.attribut`** ou **`objet.methode()`**.

[DOC 1 : en Python, tout est objet...]

```
>>> t = [10, 5, 8]
>>> t.append(40)
>>> t
[10, 5, 8, 40]
>>> t.sort()
>>> t
[5, 8, 10, 40]
>>> t.pop()
40
>>> t
[5, 8, 10]
```

II – L'instanciation (création d'un objet)

2.1 - Classe et instance d'une classe

Les instructions d'une **Classe** permettent de réaliser de vrais **objets**. La classe est donc une sorte de recette ou de moule.

2.2 - Création d'une Classe vide

A - Déclaration d'une Classe (le moule)

Créer une Classe en Python nécessite l'utilisation

- du mot-clé **`class`** suivi
- du nom de la Classe **qui commence par une Majuscule par convention**
- de « : » en fin de déclaration

On placera une documentation d'une ligne ensuite.
Les instructions sont à décaler de 4 espaces.

[DOC 2 : déclaration d'une classe]

```
1  class Personnage:
2      '''Ceci est une classe de mon super RPG'''
3      pass
```

B - Informations sur la Classe (le moule)

Une Classe possède un type **`<class 'type'>`**, une adresse et l'interpréteur affiche son lieu de création et son nom par défaut lorsqu'on lui demande ce qu'elle contient.

[DOC 3 : informations sur une classe]

```
>>> type(Personnage)
<class 'type'>
>>> id(Personnage)
21741560
>>> Personnage
<class '__main__.Personnage'>
```

2.3 - Création d'objets

A - Instanciation et instances de la classe

Instanciation : création d'un nouvel objet à partir du Constructeur d'une Classe.

Instance : synonyme d'objet. La différence est qu'on indique la classe lorsqu'on parle d'instance.

Constructeur : pour créer un objet à partir d'une Classe, on utilise une fonction un peu particulière qu'on nomme le Constructeur. La syntaxe est simple : on utilise le nom de la Classe suivi des parenthèses !

Pas d'entrée particulière ⇒ **Constructeur** ⇒ Adresse d'un nouvel objet

B - Tester la classe d'un objet

Soit avec **`type()`**, soit avec **`isinstance()`**.

C - Comportement publique visible

Par défaut, Python affiche l'adresse de l'objet et la classe dont il est issu.

[DOC 4 : Différence objet et classe]

Classe : **`Personnage`**
Constructeur : **`Personnage()`**
Objet ou Instance de Personnage : **`x = Personnage()`**

```
>>> alice = Personnage()
>>> bob = Personnage()
>>> id(alice)      →    139663546393488
>>> id(bob)        →    139663546395728
>>> id(Personnage) →    21741560

>>> type(alice)
<class '__main__.Personnage'>
>>> type(alice) == Personnage
True
>>> isinstance(alice, Personnage)
True
>>> isinstance(alice, list)
False

>>> alice
<__main__.Personnage object at 0x7f25459ff9b0>
```

III – Mauvaise pratique : rajouter des attributs à la volée

3.1 - Création d'attributs à la volée

A - Attribut

Un attribut est une sorte de variable placée à l'intérieur d'un objet.

B - Rajouter des attributs

On crée un attribut avec **`objet.attribut = valeur`**

[DOC 5 : Rajout d'attributs]

```
1    # Déclaration des classes
2    class Personnage:
3        '''Ceci est une classe de mon super RPG'''
4
5    bob = Personnage()
6    bob.nom = "Skywalker"
7    bob.prenom = "Bob"
8    bob.age = 25
```

C – Accéder aux valeurs des attributs

On accéde à une valeur avec **`objet.attribut`**.

D - Modifier les valeurs des attributs

Les objets sont fortement **muables** : on peut modifier le contenu sans modifier l'adresse de l'objet. On utilise la syntaxe **`objet.attribut = nouvelle_valeur`**

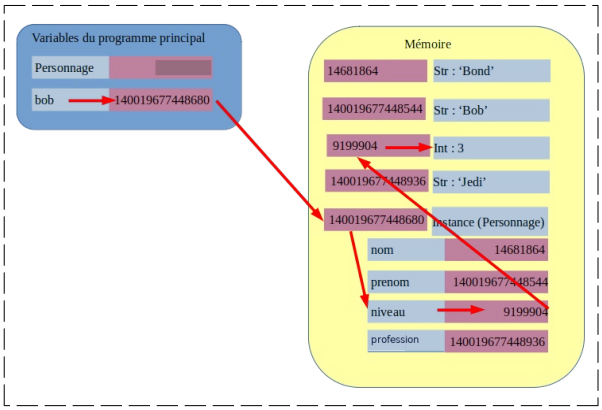
[DOC 6 : Lecture et modification d'attributs]

```
>>> bob.nom
'Skywalker'
>>> bob.prenom
'Bob'
>>> bob.nom = "Bond"
>>> bob.nom
'Bond'
```

3.2 - Principe de la structure d'un objet

Un objet possède une adresse. A cette adresse se trouve une sorte d'espace des noms interne à l'objet lui-même.

[DOC 7 : Une vision d'un objet]



Jamais plus d'attributs créés à la volée

Même si Python permet de rajouter des attributs à la volée n'importe où dans le code, on veillera à ne pas le faire.

IV – La méthode spéciale __init__()

4.1 - Les 3 étapes lors de l'instanciation

1. Réservation d'une adresse
2. Initialisation des attributs avec **`__init__()`**
3. Renvoi de l'adresse

4.2 - La méthode __init__()

A - Rôle de __init__()

Centraliser la création et l'initialisation des attributs.

B - Comment déclarer une méthode ?

Comme une fonction mais en décalant de 4 espaces pour l'inclure dans la Classe.

[DOC 8 : Déclaration de la méthode __init__()]

```
1  class Personnage:
2      '''Ceci est une classe de mon super RPG'''
3      def __init__(self):
4          self.nom = 'aucun'
5          self.prenom = 'aucun'
6          self.niveau = 0
7          self.profession = 'aucune'
```

C - Quelles conséquences lors de la création ?

Les attributs sont préremplis dès la création.

[DOC 9 : Conséquence sur les attributs à la création]

```
>>> p = Personnage()
>>> p.nom
'aucun'
>>> p.prenom
'aucun'
>>> p.niveau
0
```

D - A quoi ça sert de faire cela ?

A **centraliser l'information** sur les attributs.
La méthode spéciale **`__init__()`** est donc nommée la **méthode-initialisateur** ou l'**initialisateur**.
Puisque c'est le constructeur qui l'appelle automatiquement, par simplification, on la nomme parfois **méthode-constructeur** ou même **constructeur.** C'est un abus de langage courant. Le vrai constructeur est **`Personnage()`**.

4.3 - Le paramètre self

Ce paramètre est particulier puisque **l'utilisateur ne l'envoie pas.**

Le paramètre **`self`** doit toujours être placé **en premier dans le prototype** d'une méthode.

self contient l'adresse de l'objet en cours d'utilisation.

4.3 - Transmission d'arguments lors de la constuction

4.3.1 Principe

On peut transmettre des arguments au constructeur qui va alors les transmettre à la méthode **__init()**.

[DOC 10 : Liaison paramètres - attributs]

```
class Personnage :
    '''Ceci est une classe de mon super RPG'''
    def __init__(self, a, b, c, d):
        self.nom = a
        self.prenom = b
        self.niveau = c
        self.profession = d

>>> bob = Personnage("Luke", "S", 5, "Jedi")
>>> bob.nom
'Luke'
```

4.3.2 Pratique usuelle, qui peut paraître magique

On a souvent tendance à nommer avec le même nom le paramètre et à l'attribut de l'objet.

[DOC 11 : Pratique usuelle]

```
class Personnage :
    '''Ceci est une classe de mon super RPG'''
    def __init__(self, nom, prenom):
        self.nom = nom
        self.prenom = prenom
```

4.3.3 Paramètre par défaut

Parfois la majorité des arguments envoyés au constructeur seront toujours les mêmes pour toutes les instances. Par exemple, le niveau de départ des personnages à 1. Dans ce cas, on utilise des valeurs par défaut pour certains paramètres.

Attention, **les paramètres avec valeur par défaut doivent tous être à droite.**

[DOC 12 : Valeurs par défaut]

```
class Personnage:
    '''Ceci est une classe de mon super RPG'''
    def __init__(self, nom='Aucun', niveau=1):
        self.nom = nom
        self.niveau = niveau

>>> alice = Personnage()
>>> alice.niveau
1
```

4.3.4 Paramètres nommés

On peut transmettre les paramètres dans n'importe quel ordre pourvu qu'on nomme tous les paramètres qui n'ont pas de valeur par défaut.

[DOC 13 : Paramètres nommés]

```
>>> bob = Personnage(
        prenom="Luke",
        nom="Skywalker",
        niveau=5,
        profession="Jedi")
```