Classes et objets

Développeur Python

Sommaire

1. Présentation des concepts.
2. Classes et objets en Python.

1. Présentation des concepts.

1. Présentation des concepts.

Langages procéduraux : rappels

- Dans ce type de langages, les données sont séparées des sous-programmes (procédures et fonctions) qui les traitent.

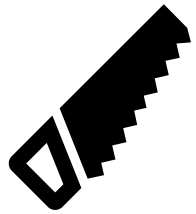
- On doit donc commencer par identifier les fonctions principales et les structures de données manipulées par ces fonctions.

1. Présentation des concepts.

Langages procéduraux : deux inconvénients majeurs

1. Les fonctions écrites pour un projet ne seront que rarement réutilisables dans un autre projet.

2. Le découplage entre données et fonctions fait qu'une modification des structures de données entraine de multiples points de correction du logiciel.

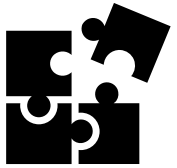


1. Présentation des concepts.

Programmation Orientée Objet : idée fondamentale

- Regrouper au sein d'une même entité certaines données et les moyens de traitement de ces données.

- Rompre le découplage entre données et fonctions produira des codes plus facilement maintenables.

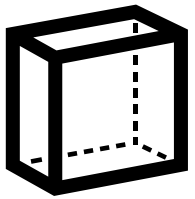


1. Présentation des concepts.

Notion d’objet : définition

- Il s’agit d’une entité possédant :

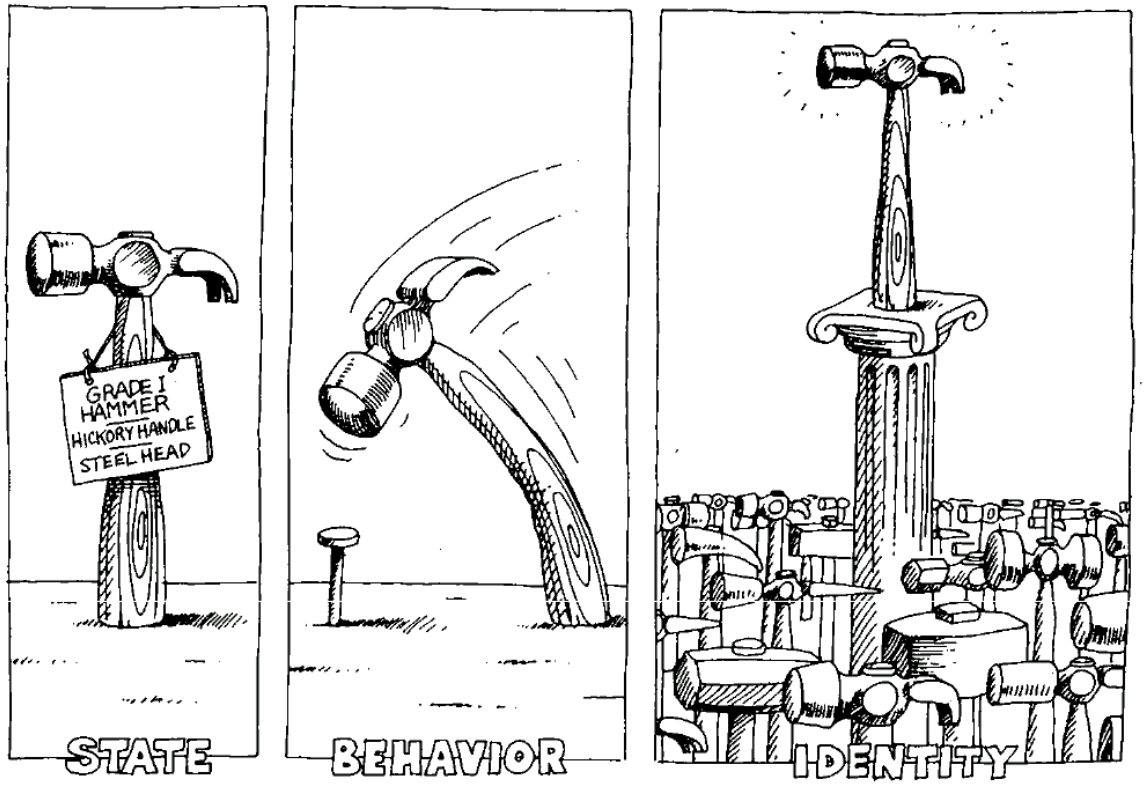
    1. Une identité.
    2. Des variables définissant son état appelées attributs.
    3. Des sous-programmes gérant son comportement appelées méthodes.



1. Présentation des concepts.

Notion d'objet : représentation imagée

- Les trois caractéristiques d'un objet : une identité, un état et un comportement.



1. Présentation des concepts.

Notion de classe : définition

- Il s’agit d’une abstraction regroupant des objets ayant les mêmes attributs et les mêmes méthodes.

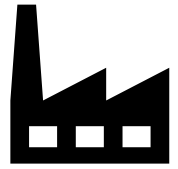
- Un objet est alors une instance de la classe correspondante, et se distingue des autres instances par son identité et la valeur de ses attributs.

1. Présentation des concepts.

Notion de classe : remarque

- On retrouve ainsi notre manière de pensée habituelle, où nous “classifions“ chaque élément de notre entourage : animaux, véhicules, ordinateurs, étudiants, livres…

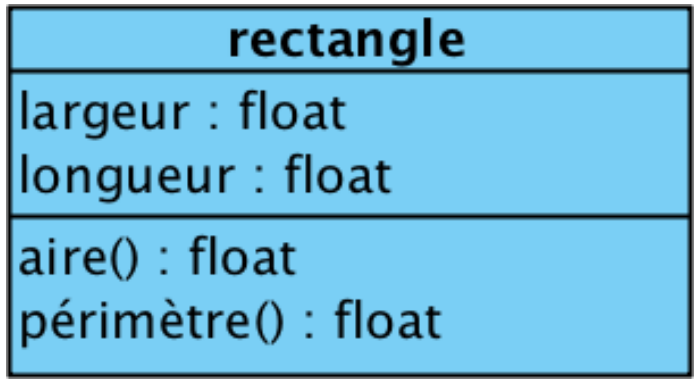
- De façon très imagée, une classe peut être vue comme un moule ou une usine à objets.

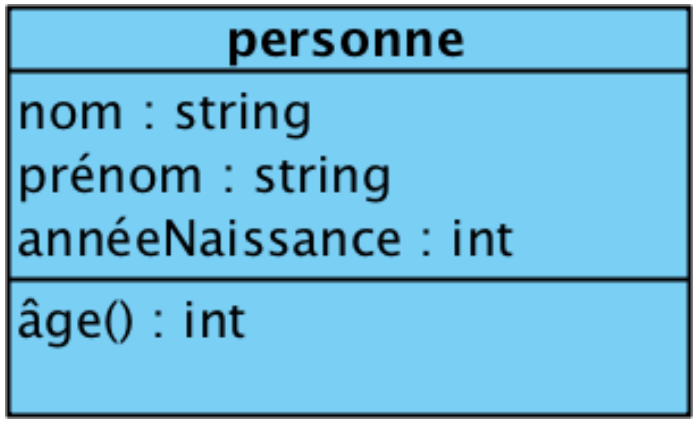


1. Présentation des concepts.

Notion de classe : premiers exemples

- Une classe “rectangle“ et une classe “personne“ :

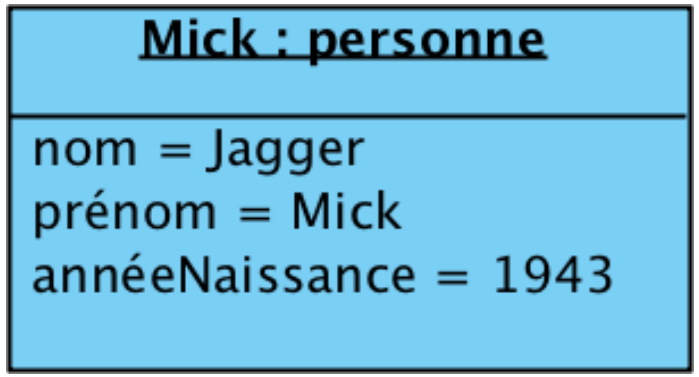


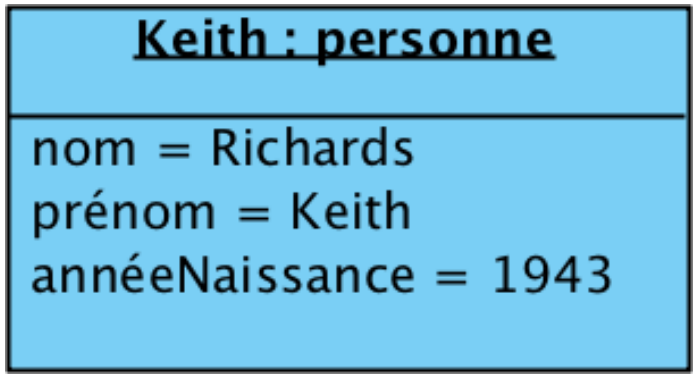


1. Présentation des concepts.

Instanciation d'objets : exemple

- Deux instances de la classe "personne" précédente :





1. Présentation des concepts.

Programmation procédurale vs orientée objet

- En POO une des questions fondamentales du développeur est donc : sur quoi porte le programme ?

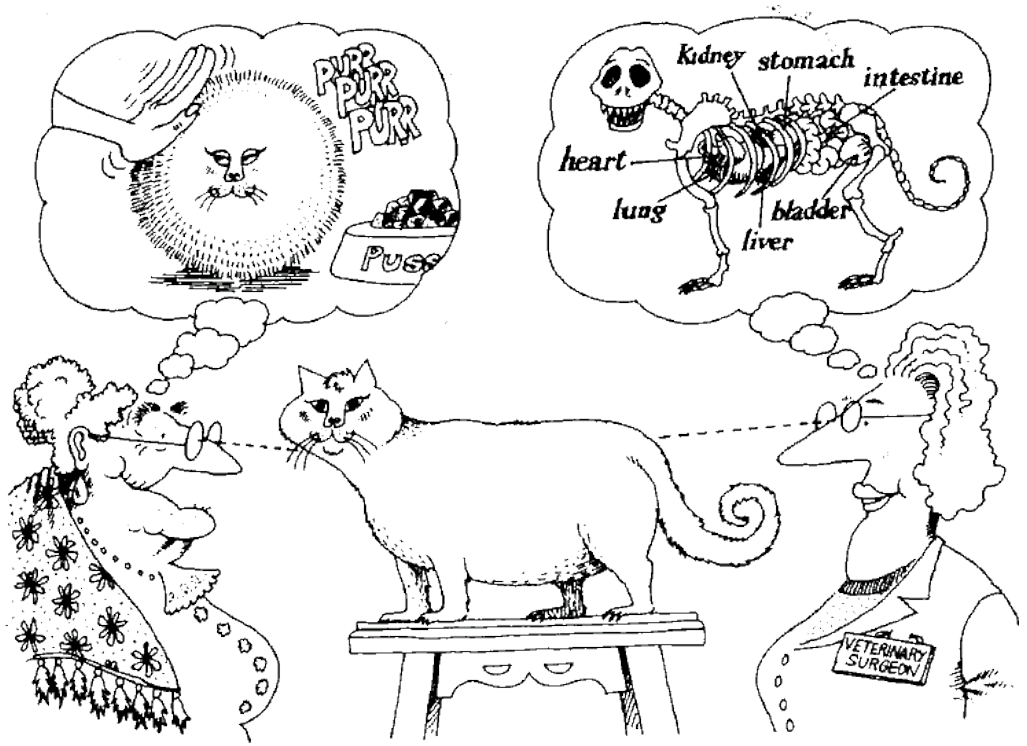
- A opposer à la question fondamentale à se poser en programmation procédurale : à quoi sert le programme ?

- Cette différence se retrouvera aussi lors de la phase de conception, où nous réserverons l'algorithmique classique à l'implémentation des méthodes, alors que pour concevoir les classes nous utiliserons un langage graphique plus adapté : l'UML.

1. Présentation des concepts.

Principe d'abstraction

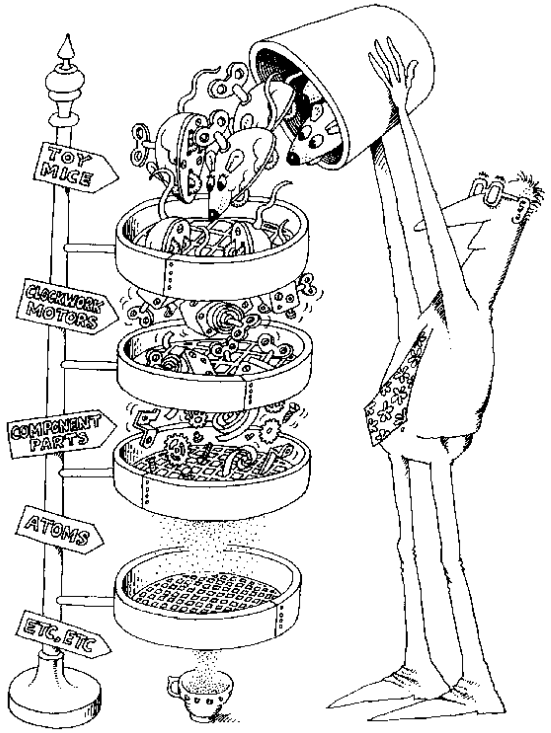
- Démarche consistant à ne retenir que les propriétés (attributs et méthodes) pertinentes d'un objet pour un problème précis.



1. Présentation des concepts.

Principe d’abstraction

- D’un contexte à un autre on ne retiendra donc pas nécessairement les mêmes attributs et méthodes pour modéliser une même classe.



1. Présentation des concepts.

Principe d’abstraction : citation de Graady Booch

“*An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of object and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer.*”

1. Présentation des concepts.

Principe d’abstraction : exemple

- Une classe “Personne“ aura dans un contexte SUPINFO, des attributs nom, prénom, ID, Marks,...

- Alors que dans un contexte Sécurité Sociale, on retiendra nom, prénom, numéro d’assuré, mutuelle,...

1. Présentation des concepts.

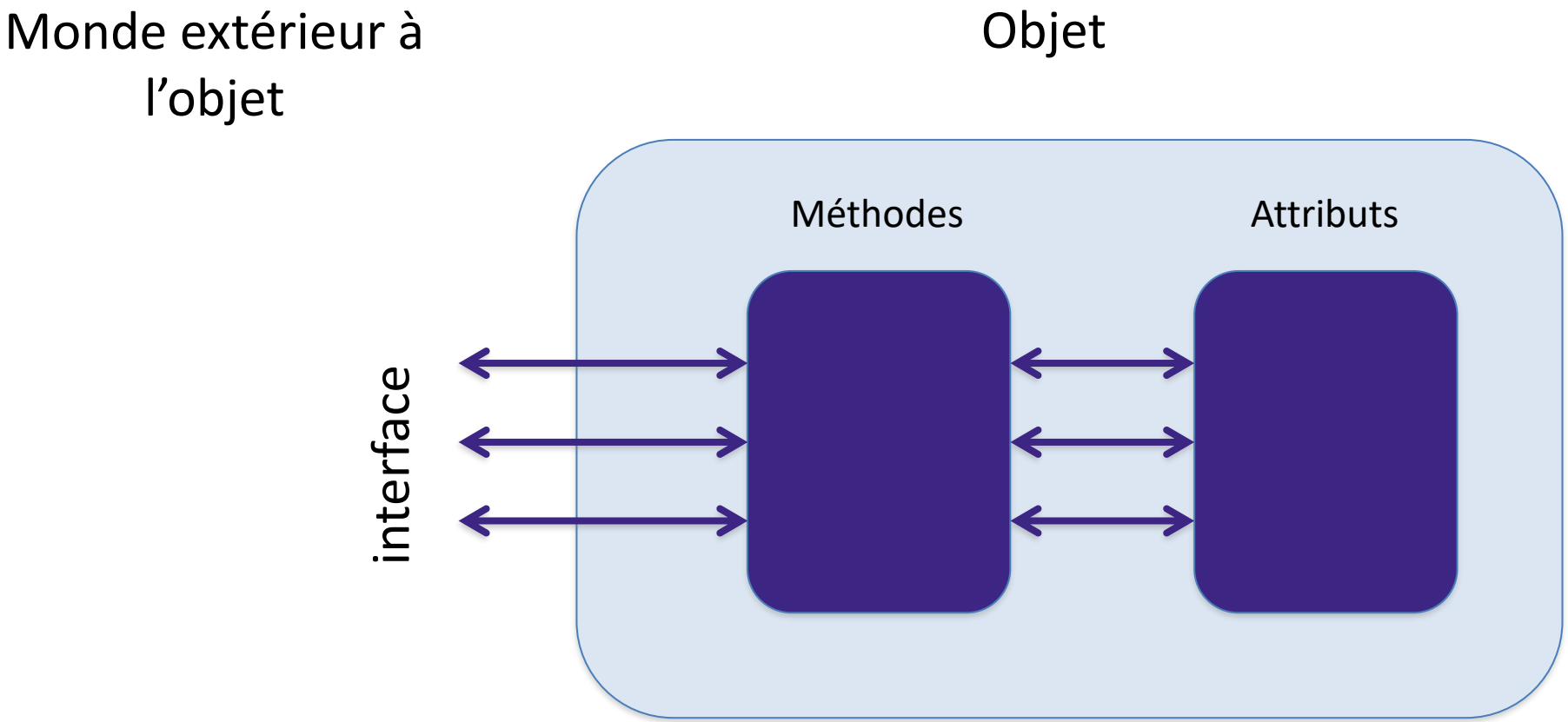
Principe d’encapsulation

- Principe interdisant l’accès direct aux attributs.

- On ne dialoguera avec l’objet qu’à travers une interface définissant les services accessibles aux l’utilisateurs de l’objet. Ce sera le rôle des méthodes.

1. Présentation des concepts.

Principe d'encapsulation : représentation imagée

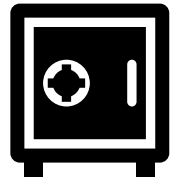


1. Présentation des concepts.

Principe d'encapsulation : deux intérêts majeurs

1. Facilitation de l'évolution d'une application : on peut modifier les attributs d'un objet sans modifier la façon dont il est utilisé.

2. Garantie de l'intégrité des données : leur accès direct est interdit ou en tout cas limité et contrôlé.



1. Présentation des concepts.

Principe d’encapsulation : citation de Graady Booch

“*Encapsulation is the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior; encapsulation serves to separate the contractual interface of an abstraction and its implementation.*”

”
“

1. Présentation des concepts.

Principe d’encapsulation : visibilité des attributs et méthodes

- Un attribut ou une méthode sont dits privés si leur utilisation est interdite en dehors de la classe.
- Un attribut ou une méthode sont dits publics si leur utilisation est autorisée en dehors de la classe.
- Le principe d’encapsulation est donc de déclarer les attributs de façon privée et les méthodes de façon publique.

1. Présentation des concepts.

Principe d’encapsulation : accès aux attributs

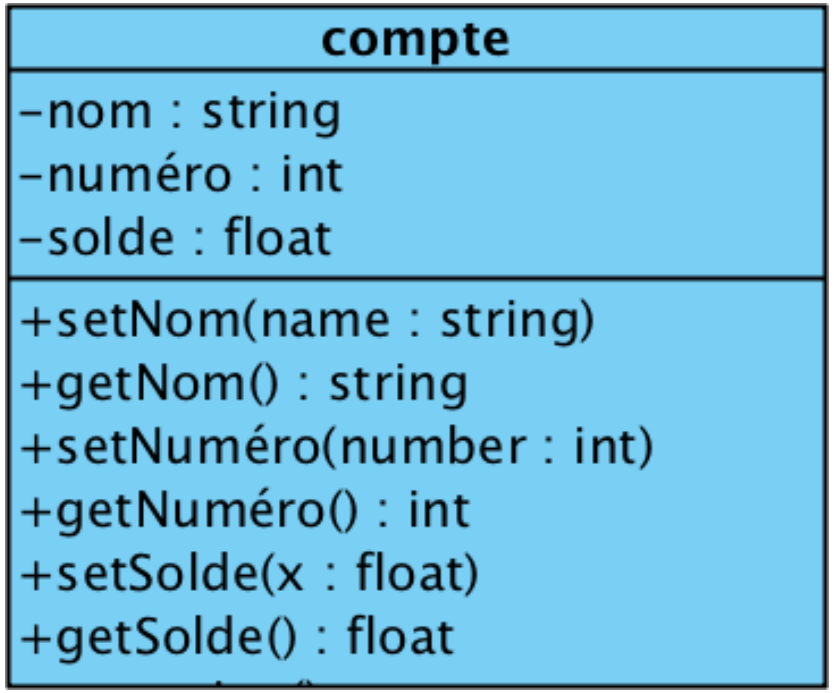
- Pour accéder aux attributs depuis l’extérieur de l’objet on utilisera des méthodes spécifiques :
    - Un “getter“ est une méthode dont le rôle est de retourner la valeur d’un attribut.
    - Un “setter“ est une méthode dont le rôle est de mettre à jour la valeur d’un attribut.

1. Présentation des concepts.

Principe d'encapsulation : exemple

- Une classe "compte" avec des attributs privés.

- Pour changer la valeur du solde on doit nécessairement utiliser la méthode "setSolde" qui contrôlera la nouvelle valeur afin par exemple de ne pas être à découvert .



1. Présentation des concepts.

2. Classes et objets en Python.

2. Classes et objets en Python.

Démarche générale

- On commence par déclarer une classe qui sera le moule commun à nos futurs objets :

```
class myClass:
    ...
```

- Selon nos besoins, on instanciera ensuite des objets de cette classe, chacun ayant une identité et des valeurs d'attributs qui lui sont propres :

```
myObject = myClass(...)
```

2. Classes et objets en Python.

Déclaration de méthodes au sein d'une classe : syntaxe

```
class myClass:
    def myMethod1(self, para1, para2, ...):
        ...
    def myMethod2(self, para1, para2, ...):
        ...
```

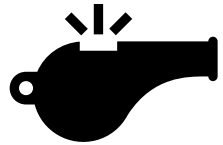
- Le premier paramètre d'une méthode est toujours une référence vers l'objet courant. Il est usuellement appelé "self".

- Les autres paramètres sont les données transmises à la méthode pour son bon fonctionnement.

2. Classes et objets en Python.

Déclaration de méthodes au sein d'une classe : remarques

- Ces déclarations de méthodes se font dans la classe.
- La syntaxe d'une méthode est la même que celle d'un sous-programme. C'en est d'ailleurs un.
- En particulier les paramètres des méthodes peuvent comporter des valeurs par défaut.



2. Classes et objets en Python.

Une méthode bien particulière : le constructeur

- Méthode spécifique permettant de déclarer et initialiser les attributs.
- Son nom est imposé : __init__

```
class myClass:
    def __init__(self, para1, para2, ...):
        ...
    def myMethod1(self, para1, para2, ...):
        ...
```

2. Classes et objets en Python.

Le constructeur : remarque importante

- Le langage Python est assez permissif et permet l'ajout dynamique d'attributs "à la volée", après l'instanciation d'un objet, sans passer par le constructeur.

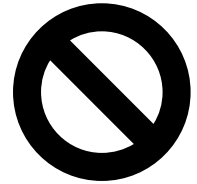
- On bannira cette mauvaise pratique, non partagée par la plupart des langages orientés objets.

2. Classes et objets en Python.

Implémentation du principe d'encapsulation

- Pour déclarer un attribut privé on fait précéder son nom d'un double underscore.

```
def __init__(self, para1, para2, ...):
    self.__myAttribute1 = para1
    self.__myAttribute2 = para2
    ...
```



2. Classes et objets en Python.

Accès aux attributs et méthodes

- Depuis l'intérieur de la classe :

```
self.__myAttribute

self.myMethod(para1, para2, ...)
```

- Depuis l'extérieur de la classe :

```
myObject.myMethod(para1, para2, ...)
```

2. Classes et objets en Python.

Getter : syntaxe classique

- Son rôle étant de retourner la valeur d'un attribut il ne contient en général que la commande "return" :

```
def getMonAttribut(self):
    return self.__monAttribut
```

- En pratique on aura un "getter" pour chaque attribut.

2. Classes et objets en Python.

Setter : syntaxe classique

- Son rôle étant de mettre à jour la valeur d'un attribut il contient principalement une affectation (conditionnée parfois à la vérification d'une contrainte) :

```
def setMonAttribut(self, nouvelleValeur):
    self.__monAttribut = nouvelleValeur
```

- En pratique on aura un "setter" pour chaque attribut.

2. Classes et objets en Python.

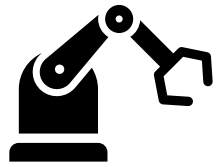
Attributs de classe : définition

- Attribut ayant la même valeur pour toutes les instances de la classe.
- Il doit être déclaré et initialisé au sein de la classe, en dehors de toute méthode (y compris __init__).
- Exemples d'utilisation : décompte du nombre d'objets instanciés de la classe, constantes relatives à la classe (valeur d'un taux, etc.).

2. Classes et objets en Python.

Méthodes de classe : définition

- Méthode ayant un effet indépendant des objets.
- Leur rôle sera donc concrètement de manipuler les attributs de classe.
- Leur nom sera précédé du mot clé “@classmethod“ et son premier paramètre sera une référence vers la classe, nommée généralement “cls“.



2. Classes et objets en Python.

Attributs et méthodes de classe : syntaxe

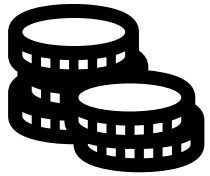
```
class myClass:
    __myClassAtribute = ...

    def __init__(self, para1, para2, ...):
        ...
    def myMethod1(self, para1, para2, ...):
        ...

    @classmethod
    def myClassMethod(cls, para1, para2, ...):
        ...
```

2. Classes et objets en Python.

Attributs de classes : remarques

- Dans le cas d'un attribut de classe dénombrant le nombre d'objets instanciés, il est pertinent de le mettre à jour, *i.e.* l'incrémenter, dans le constructeur.

- Inversement il conviendra de le décrémenter lorsqu'un objet sera déréférencé.



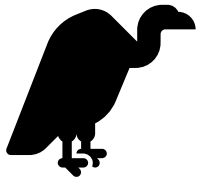
2. Classes et objets en Python.

Une autre méthode singulière : le destructeur

- Méthode spécifique permettant de libérer la mémoire d'un objet qui n'est plus référencé.

- Il n'est pas nécessaire d'en implémenter une version dans chaque classe car Python gère lui-même la mémoire.

- Il sera utile de le faire quand une autre tâche est requise que celle-ci, *e.g.* la mise à jour d'un attribut de classe.



2. Classes et objets en Python.

Une autre méthode singulière : le destructeur (suite)

- Son nom est imposé : __del__

```
class myClass:
    def __init__(self, para1, para2, ...):
        ...
    def __del__(self):
        ...
    def myMethod1(self, para1, para2, ...):
        ...
```

2. Classes et objets en Python.