1 Principe et vocabulaire

En python, on a vu plusieurs types de fonctions pré-existantes s'appelant de manières différentes :

```
#fonctions habituelles
print("coucou")
len(liste)

#mais on a aussi croise
liste.append(2)
```

On peut se demander pourquoi, par exemple, la fonction `append` ne s'appelle pas en écrivant `append(liste, element)` comme pour les premières fonctions.

Un premier élément de réponse se trouve dans le fait que, contrairement au premières, la fonction `append` modifie la liste et on peut donc voir `liste.append` comme une fonction à un argument ayant pour effet de modifier la liste `liste`. En ce sens, la fonction `append` n'existe qu'attachée à une liste donnée.

C'est en fait le princi de la programmation orientée objet : il s'agît d'attacher à des objets (dans l'exemple précédent les listes) un certain nombre de variables (les attributs) et de fonctions (les méthodes). Ainsi, `append` est une méthode des objets listes.

Par exemple, on pourrait imaginer vouloir travailler sur des objets `MaisonConnectee`. Ces objets auraient un certain nombre d'attributs (`adresse`, `proprietaire`, `temperature` etc.) et un certain nombre de méthodes (`ouvrirVolets()`, `augmenterChauffage(dT)` etc.). On voit que ces variables et fonctions n'ont pas de sens en soit mais uniquement du point de vue d'une maison donnée.

En programmation orienté objet, le modèle spécifiant les attributs et les méthodes des objets que l'on veut modéliser s'appelle une classe.

Exemple : `list` est une classe, `MaisonConnectee` serait une classe etc.

Les variables d'une classe sont appelées ses attributs. Les fonctions d'une classe sont appelées ses méthodes.

Exemple : `append` est une méthode de la classe `list`, `proprietaire` serait un attribut de la classe `MaisonConnectee`.

Une fois une classe créée, on peut créer des objets de cette classe, on parle alors d'instances.

Exemple : `liste1 = [1,2,3]` est une instance de la classe `list`, `maison1` (on n'a pas précisé comment instantier les classes pour l'instant) pourrait être une instance de la classe `MaisonConnectee`.

2 En python

Reprenons l'exemple de l'introduction. On veut créer une classe `MaisonConnectee`, dont les attributs sont :

- `adresse` (chaine de caractères) ;
- `proprietaire` (chaine de caractères) ;
- `etatVolets (booléen)` ;
- `temperature (flottant)` ;

et dont les méthodes sont :

- `afficherEtat()` (résume les attributs) ;
- `ouvrirVolets()` (ouvre ou ferme les volets) ;
- `augmenterTemperature(dT)` (modifie la température de dT).

```
class MaisonConnectee :
    def __init__(self, prop, adr):
    #fonction permettant d'instancier la classe
        self.proprietaire = prop
        self.adresse = adr
        #quand on cree une maison, il
        #faut specifier proprietaire et adresse
        self.etatVolets = False
        self.temperature = 20.0
        #les volets sont par defaut fermes,
        #la temperature est par defaut de 20.0

    def afficherEtat(self):
        print("Ad : ", self.adresse)
        print("Prop : ", self.proprietaire)
        if self.etatVolets : print("Volets ouverts")
        else : print("Volets fermes")
        print("T = ", self.temperature)
```

```
19
20      def ouvrirVolets(self):
21          self.etatVolets = not(self.etatVolets)
22
23      def augmenterTemperature(self, dT):
24          self.temperature += dT
```

Pour déclarer une classe, la syntaxe est `class NomDeLaClasse :`. Par convention, les classes commencent par une majuscule, les méthodes et attributs par une minuscule.

La fonction `__init__` sert à instancier la classe. On la place en premier.

L'argument `self` présent dans les méthodes sert à désigner l'instance sur laquelle on appelle la méthode définie.

On accède aux attributs de l'instance par `self.<attribut>`.
Il est assez courant d'écrire certaines fonctions, notamment :

— une méthode d'affichage de l'objet ;
— des méthode de modification des attributs de l'objet (mutateurs) ;
— des méthodes renvoyant les attributs de l'objet (assesseurs).

Remarque 1 : Il est possible de faire en sorte que `print(<instance>)` appelle la méthode d'affichage de la classe.

Remarque 2 : Les assesseurs et mutateurs ne sont pas particulièrement nécessaires dans le cadre de petits projets car une fois une classe instanciée, on peut toujours avoir accès et modifier les attributs comme n'importe quelle variable avec `<instance>.<attribut>` mais il est possible, lorsque l'on permet à quelqu'un d'autre de travailler avec une classe que l'on a codée, de limiter ces accès aux seules utilisations des assesseurs et mutateurs.

Remarque 3 : Le fait que la fonction `__init__` soit entourée des symboles `__` ne change rien à sa définition. En revanche, cela implique que l'interpréteur Python va l'utiliser via d'autres syntaxes que `__init__(<arguments>)`. En fait, il existent d'autres fonctions (à définir dans une classe si cela est pertinent) que l'on peut définir de la sorte et qui permettent l'utilisation de syntaxes particulières pour des objets `a` et `b` du même type :

Méthode	Appel	Effet	Re
`__str__(self)`	`srt(a)`	Revoie une chaîne de caractère décrivant `a`	
`__lt__(self,b)`	`a<b`	Teste si `a` est plus petit que `b`	Le concepteur de la classe choisit ce que signif
`__eq__(self,b)`	`a==b`	Teste si `a` et `b` sont égaux	Le concepteur de la classe choisit ce que
`__contains__(self,x)`	`x in a`	Teste si l'élément `x` est contenu dans `a`	À n'utiliser que dans des classes
`__add__(self,b)`	a+b	Renvoie la somme de `a` et `b`	Le concepteur de la classe choisit ce que

Maintenant que la classe maison est définie, on peut créer et utiliser des maisons :

```
maison1 = MaisonConnectee("3 rue de la mer", "Toto")
maison2 = MaisonConnectee("5 rue de la mer", "Titi")
maison1.ouvrirVolets()
maison2.augmenterTemperature(-3)
maison1.afficherEtat()
maison2.afficherEtat()
```

Ici, on créé deux maisons, on en modifie les attributs à l'aide de nos mutateurs et on les affiche.

Remarque 1 : pour instancier un objet la syntaxe est `nomObjet = NomClasse(<arguments>)`. Cette syntaxe appelle la fonction `__init__`.

Remarque 2 : On voit que l'argument `self` n'apparaît pas dans les méthodes appelées. C'est normal, le self désignant l'instance sur laquelle la méthode est appelée, il est inutile de le préciser lors de l'appel.

Ajouter à la classe `MaisonConnectee` quelques attributs et quelques méthodes cohérentes à la situation modélisée (modifier le `__init__` pour les nouveaux attributs).

Tester vos modifications en instanciant plusieurs maisons.

3 Exercice : jeu de rôle

Le but de ce mini-projet est de créer la base d'un petit jeu de rôle en s'aidant de la programmation orientée objet.
Dans ce projet, on veut créer une classe `Personnage` disposant de trois attributs : `nom`, `force` et `pointsDeVie`.

La classe disposera en plus de plusieurs méthodes : `afficherStats()` qui affiches les attributs du personnage, `estVivant()` qui renvoie True ou False en fonction de l'état des points de vie du personnage et `attaquer(perso)` qui effectue une attaque sur un autre personnage (une attaque ne peut être effectuée que par et sur un personnage vivant et hôte un nombre de points de vie égal à la valeur de force de l'attaquant).

Enfin, on implémentera une fonction `combat(perso1, perso2)` lançant un combat entre deux personnages qui attaquent à tour de rôle jusqu'à ce que les points de vie de l'un des deux atteigne zéro.