Programmation 37 Les méthodes dans les objets

I - Déclaration d'une méthode

Déclarer une méthode revient à **déclarer une fonction à l'intérieur de la déclaration de la Classe**.

Deux différences avec une fonction "classique" :

1. il faut **tabuler** pour que l'interpréteur Python comprenne l'appartenance de la méthode à la Classe.
2. la **méthode doit nécessairement avoir un premier paramètre** qui recevra automatiquement l'adresse de l'objet sur lequel on est en train d'agir. En Python, on le nomme souvent **`self`**.

(DOC 1 – Création de méthodes)

```
class Personnage:
    '''Ceci est une classe permettant de créer un personnage dans mon super RPG'''

    def __init__(self, nom='Aucun', prenom='Aucun', niveau=0, profession="Humain de base"):
        self.nom = nom
        self.prenom = prenom
        self.niveau = niveau
        self.profession = profession
        self.pv_max = self.niveau * 10
        self.pv = self.pv_max

    def soigner(self, x):
        '''Le personne regagne x points de vie'''
        self.pv = self.pv + x
```

II Comprendre l'appel à une méthode

(DOC 2 – Appel de méthodes)

Appel :

```
>>> heros = Personnage()
>>> heros.soigner(5)
. . . transformé à l'interne en . . .
>>> Personnage.soigner(heros, 5)
```

Déclaration :

```
def soigner(self, x):
    self.pv = self.pv + x
```

Transfert des paramètres : **`self`** contient l'objet devant l point.

`>>> heros.soigner(5)`
Le paramètre **`self`** va contenir **`heros`** lors de cet appel.

On peut comprendre ce que fait la méthode **`soigner()`** : la ligne **`self.pv = self.pv + x`** montre qu'on incrémente l'attribut pv de x pour l'objet sur lequel on a lancé la méthode. On ne sait pas qui il est mais on connaît son adresse car elle a été stockée dans **`self`**.

III Modification via méthode vs modification directe

Pourquoi créer une méthode **`soigner()`** si on peut faire pareil avec une instruction ?

`heros.soigner(5)` VS **`heros.pv = heros.pv + 5`**

Le principe de l'interface (via l'utilisation des méthodes) **permet à l'utilisateur de provoquer l'effet voulu sans risque d'endommager les données internes de l'objet :** pas de manipulation directe mal effectuée.

IV Encapsulation

IV.1 Principe de l'encapsulation

Seul un objet d'une classe A peut modifier les attributs d'un objet d'une Classe A en manipulant son modèle, c'est à dire sans passer par une méthode. Cette notion est l'**encapsulation** et c'est l'un des principes fondamentaux de la programmation orientée objet :

- on fournit à l'utilisateur une Classe qui contient un ensemble de méthodes d'interface
- niveau d'encapsulation 1 : l'utilisateur n'est pas autorisé à modifier directement les attributs d'un objet.
- niveau d'encapsulation 2 : l'utilisateur n'est pas autorisé à lire les attributs sans passer par l'interface.
- niveau d'encapsulation 3 : l'utilisateur n'est pas autorisé à connaître les noms des attributs.

IV.2 Ne pas respecter l'encapsulation

Avantage : programme court.
Désavantage : aucune sécurité !

IV.3 Respecter l'encapsulation

Avantage : sécurité si les méthodes sont bien conçues.
Désavantage : programme plus long à écrire et à créer.