1 Notion de paradigme de programmation

> Un paradigme de programmation est une façon d'envisager la conception et la manière d'écrire un programme informatique.

Nous manipulerons cette année les paradigmes **impératif**, **fonctionnel** ou **objet**. Mais d'autres existent encore.

Le paradigme principal que vous avez dû rencontrer jusqu'à maintenant est le paradigme impératif :

> Le paradigme **impératif** consiste à écrire une suite d'instructions que le programme doit réaliser.

Les programmes impératifs contiennent des **variables qui peuvent êtres modifiées au cours de l'exécution** du programme.

Certains langages imposent la programmation selon un certain paradigme, mais beaucoup de langages, comme Python par exemple, permettent de programmer selon différents paradigmes au sein d'un même programme.

2 Paradigme fonctionnel

Comme son nom l'indique, le **paradigme fonctionnel** repose essentiellement sur l'utilisation de **fonctions**.
La notion de fonctions (ou de procédures) existe aussi lorsqu'on programme de façon impérative (pour structurer le programme et faciliter la modularité), mais dans la programmation fonctionnelle la notion de fonction correspond un peu plus au concept mathématique de fonction :

> Une fonction reçoit une entrée et renvoie une **sortie qui ne dépend que de cette entrée**.

La sortie est toujours la même pour une certaine entrée quel que soit « l'historique » de l'exécution du programme.

> Par ailleurs, l'exécution d'une fonction ne modifie jamais, le contenu d'une variable annexe à la fonction.

Enfin, la programmation fonctionnelle repose beaucoup sur la composition de fonctions : on peut appliquer une fonction à une autre fonction :

> En programmation fonctionnelle, **une fonction peut être passée en entrée** d'une autre fonction.

Exemple :

```
def transforme_tableau(t, f):
    """ Applique la fonction f à chaque élément d'un tableau t
    Entrées :
    - f : fonction de transformation
    - t : tableau d'éléments à transformer
    Sortie : Nouveau tableau contenant les éléments transformés
    """
    return [f(element) for element in t]
```

```
def carre(x):
    """ Renvoie le carré du nombre x """
    return x**2

>>> n = 5
>>> t = [i for i in range(n+1)] # tableau de 5 premiers entiers
>>> transforme_tableau(t, carre)
[0, 1, 4, 9, 16, 25]
>>> t
[0, 1, 2, 3, 4, 5] # t n'a pas été modifié !
# si on veut modifier t, il faut affecter le résultat de la fonction à t :
>>> t = transforme_tableau(t, carre)
>>> t
[0, 1, 4, 9, 16, 25]
```

Contre-exemple :

```
def incremente_tableau(t):
    """ Incrémente chaque élément d'un tableau t (modification en place).
    Entrée : t : tableau d'éléments à incrémenter
    """
    for i in range(len(t)):
        t[i] += increment # paramètre externe
    return t # facultatif

>>> n = 5
>>> t = [i for i in range(n+1)]
>>> increment = 5
>>> incremente_tableau(t)
[5, 6, 7, 8, 9, 10]
# t a été modifié : ne respecte pas le paradigme fonctionnel !
>>> t
[5, 6, 7, 8, 9, 10]

# autre interdit : l'appel de la même fonction avec la même entrée
# produit un autre effet !
>>> t = [i for i in range(n+1)] # même tableau initial
>>> increment = 1
>>> incremente_tableau(t)
[1, 2, 3, 4, 5, 6]
```

Le respect du paradigme fonctionnel peut **ajouter de la robustesse** au programme et éviter certains bugs difficiles à déceler, notamment lorsque les fonctions modifient des variables en place ou utilisent des paramètres externes (voir contre-exemple ci-dessus).

3 Paradigme objet

Le paradigme objet conduit à la **programmation orientée objet** POO.
Ce paradigme repose sur la création de données, appelées objets, modélisant des concepts quelconques.
Exemples : un personnage de jeu (nom, force, vie...), une fenêtre graphique (dimension, titre, ...), une carte à jouer (couleur, hauteur), une liste ...

Un objet est caractérisé par ses attributs et ses méthodes.

Les **attributs** sont des variables qui caractérisent un objet, et les **méthodes** sont des fonctions qui peuvent s'appliquer spécifiquement à l'objet, par exemple pour agir sur ses attributs ou interagir avec d'autres objets du programme.

Voir plus spécifiquement le cours P1.II-Vocabulaire de la programmation objet.

On reconnaît un programme orienté objet avec la « notation pointée » qui désigne l'accès à un attribut ou l'application d'une méthode à un objet.

Remarque : En Python, tout est objet.

Exemples :

```
>>> l = [1, 2, 3]
>>> l.append(4) # application de la méthode append à l'objet list [1, 2, 3]
                # modifie l'objet en lui ajoutant un élément
>>> l.pop() # application de la méthode pop à l'objet list [1, 2, 3, 4]
            # modifie l'objet en lui retirant le dernier élément
4
```

On pourrait imaginer des objets représentants des concepts de géométrie (point, segment, cercle...) et quelques manipulation sur les attributs ou méthodes de ces objets :

```
>>> p1 = Point(2, 3)       # instanciation d'un objet Point
>>> p2 = Point(-1, 5)
>>> AB = Segment(p1, p2)   # instanciation d'un objet Segment
>>> AB.extremite1          # accès à l'attribut extremite1 de l'objet AB
(2, 3)
>>> AB.translater((2 ,4)) # application de la méthode translater à l'objet AB
>>> AB.extremite2
(1, 9)
>>> c1 = Cercle(p1, 3)     # instanciation d'un objet Cercle
>>> c1.surface()           # application de la méthode surface à l'objet c1
28.27
```

Remarque : le paradigme de programmation orientée objet repose aussi sur d'autres concepts comme l'**encapsulation**, l'**héritage** ou le **polymorphisme**, mais ces notions (pourtant très importantes) restent hors programme en Terminale.