Leçon 2 - Paradigmes de programmation

Niveau : Terminale

Prérequis : POO, programmation dynamique

Motivation : Mettre en évidence les différents paradigmes de programmation, leur utilisation, avantages et inconvénients respectifs, et ainsi introduire le paradigme fonctionnel.

Définition 1 : Un paradigme de programmation est un "style" de programmation, c'est-à-dire la manière dont les solutions aux problèmes sont décrites dans un langage de programmation. Il s'agit en quelque sorte de la "vision" de la programmation qu'a le langage.

I] Le paradigme impératif.

Définition 2 : Le paradigme impératif consiste à décrire un programme sous la forme d'une séquence d'instructions ordonnée, pouvant affecter l'état du programme (variables, fonctions).

Remarque 3 : Python, C sont des exemples de langages qui utilisent le paradigme impératif.

Remarque 4 : En programmation impérative, l'ordre des instructions lors de l'exécution du programme a son importance.

Exemple 5 :

```
x = 2
y = 3
x = y
z = x
print(z)
>> 3
```

```
x = 2
y = 3
z = x
x = y
print(z)
>> 2
```

Remarque 6 : La programmation dynamique se base sur le paradigme impératif pour la résolution de problèmes.

(Développement 1 : La distance de Levenshtein)

II] Le paradigme objet.

Définition 7 : Le paradigme objet consiste à définir des structures de données (objets) et leur comportements (méthodes). Ces différentes structures sont regroupées par classes.

Exemple 8 :

```
class Voiture :
    def __init__(self, marque: str, couleur: str, kilometrage: int):
        self.marque = marque
        self.couleur = couleur
        self.kilometrage = kilometrage
    def roule(self, nb_kilometres: int):
        print(f"La voiture roule {nb_kilometres} kilomètres !")
        self.kilometrage += kilometrage
```

Constructeur : `__init__`

attribut : `self.kilometrage`

méthode : `roule`

Remarque 9 : C#, Java, Python sont des langages qui utilisent le paradigme objet.

Remarque 10 : Le paradigme objet peut être vu comme une abstraction au-dessus du paradigme impératif. C'est d'une certaine manière du sucre syntaxique : on pourrait représenter les différentes classes par des structures de données (fonctionnalités déjà existantes en programmation impérative. Cette abstraction permet d'écrire des programmes plus lisibles, en offrant une structure plus intuitive.

Exemple 11 : réimplémentation de la classe Voiture en programmation impérative.

```
def creer_voiture(marque, couleur, kilometrage):
    return {"marque": marque, "couleur": couleur, "kilometrage": kilometrage}

def roule(voiture, nb_kilometres):
    print(f"La voiture roule {nb_kilometres} kilomètres !")
    voiture["kilometrage"] += nb_kilometres

voiture = creer_voiture("Peugeot", "noir", 3500)
roule(voiture, 20)  # >> "La voiture roule de 20 kilomètres !"
```

III | Le paradigme fonctionnel

Définition 12: Le paradigme fonctionnel prend la fonction comme objet de manipulation, au même titre que les variables. Une fonction peut prendre en paramètre des fonctions, et renvoyer comme valeur une autre fonction.

Remarque 13: OCaml, Haskell sont des langages qui utilisent le paradigme fonctionnel.

Remarque 14: Les fonctions en programmation fonctionnelle sont comparables à des fonctions au sens mathématique du terme. Elles respectent 2 propriétés : la transparence référentielle et de ne pas créer d'effet de bord. On dit que ces fonctions sont pures.

Définition 15: Une fonction respecte la transparence référentielle si, pour les mêmes paramètres donnés, elle renvoie les mêmes valeurs.

Définition 16: Une fonction a des effets de bord si elle modifie un ou des éléments extérieurs à son calcul.

Exemple 17:

```
x = 1
def f(y):
    x += 1
    return x+y
print(f(1)) # >> 2
print(f(1)) # >> 3
```

Cette fonction ne respecte pas la transparence référentielle : 2 appels consécutifs de f(1) donnent 2 résultats différents.

Elle crée aussi un effet de bord : à chaque appel, la valeur de x est modifiée.

Remarque 18: Un langage peut être multi-paradigme. C'est le cas de Python par exemple : il utilise des approches impératives, objets mais aussi fonctionnelle.

Remarque 19:

- L'usage de fonctions pures simplifie grandement le débugage et la correction d'erreur. En effet, la transparente référentielle assure un comportement prévisible du programme, et l'absence d'effet de bord certifie aussi l'absence d'effets indésirables extérieurs.

- En contrepartie, ces restrictions excluent certains usages du paradigme impératif, notamment la gestion des entrées/sorties. Par exemple, la fonction input() de Python ne respecte pas la transparence référentielle, du fait que sa valeur dépend de l'entrée de l'utilisateur. Dans un langage purement fonctionnel, on ne peut pas considérer de telles fonctions.