Paradigmes de programmation : Impératif, fonctionnel, objet. Exemples et applications

Niveau : Terminale / MPII
Pré-requis : Bases de python / C / OCaml

Programmer c'est mettre en relation un cahier des charges et des instructions compréhensibles par la machine.

Définition 1 : Un paradigme de programmation définit la façon d'approcher la programmation informatique. Suivant le contexte, il en existe différents.

I - La programmation impérative (Terminale)

C'est la plus classique.

Définition 2 : La programmation impérative consiste à donner une suite d'instructions, chacune ayant pour seul effet de modifier l'état du programme (l'état de la mémoire, la valeur des variables, l'instruction où l'on est, etc...).

Remarque 3 : Ainsi, dans la programmation impérative, il n'existe pas de valeurs de retour. Si on en veut une, il faut écrire la valeur que l'on veut dans la mémoire.

Remarque 4 : Informellement, programmer impérativement, c'est utiliser des variables, des affectations, des tableaux, des boucles for et while, etc...

Exemple 5 : La majorité du code en python est impératif.

```
x = 1
y = x+3
while(x ≠ y):
    print(y)
    y -= 1
```

Ce programme écrit 1 dans la case mémoire de x, puis y accède pour écrire 4 dans celle de y, puis écrit la valeur de y dans l'espace mémoire dédié à l'affichage, etc...

Remarque 6 : Impératif est pris ici dans son sens courant (en informatique). Une autre définition d'impératif est que l'on dit exactement ce que la machine doit faire (exe : l'assembleur), ce qui s'oppose alors au déclaratif (comme SQL). Mais cette notion est à degré (dans tous les langages il y a une marge plus ou moins grande pour la machine) et n'est pas nécessairement la notion à laquelle on pense quand on mentionne la programmation impérative (même si les notions sont liées).

II - Programmation fonctionelle (MPII)

Définition 7 : La programmation fonctionelle consiste à composer le programme de fonctions (au sens mathématiques) et de ne récupérer que la valeur de retour.
Les changements d'état ne peuvent pas être représentés par des évolutions de fonctions, donc la programmation fonctionelle ne les admet pas. On dit que les structures données fonctionelles sont immuables.

Exemple 7 : `let max (x,y) = if x>y then x else y`
(fonction de type `int * int -> int`).

Exemple 8 : Beaucoup des données étant immuables en OCaml, « = » ne signifie pas l'affectation mais le test d'égalité.

Informellement, programmer en fonctionnel, c'est considérer les fonctions comme des objets comme les autres, ne regarder les programmes que comme des fonctions et n'avoir que des structures de données immuables.

Remarque 9 : Un argument d'une fonction ou sa valeur de retour peut être une fonction. C'est ce que l'on appelle des fonctions d'ordre supérieure.

Définition 10 : La curryfication est la transformation d'une fonction à plusieurs arguments en une fonction à un argument qui retourne une fonction sur le reste des arguments.

Exemple 11: On peut transformer la fonction max de l'exemple 7 en la fonction let max x y = if x>y then x else y de type int -> int -> int.

Exemple 12: Ici, sur l'exemple 11, on veut que max puisse comparer des éléments sur lesquels on ne connait pas l'ordre, on peut en faire une fonction d'ordre supérieur en lui fournissant une fonction de comparaison :

let max compar x y = if compar x y then x else y

de type ('a -> 'a -> bool) -> 'a -> 'a -> 'a.

On a alors max (fun x y -> x>y) qui calcule le max

max (fun x y -> x>y) 3 qui est une fonction de type (int -> int) renvoyant le maximum de son argument et 3

max (fun x y -> x<y) qui calcule le min.

Remarque : La puissance du fonctionnel vient de la récursivité

Développement 1: Équivalence fonctionnel / Impératif

III - Programmation orientée objet (Terminale)

III-1) Obtenir de la modularité

Définition 13: Une classe est un ensemble de types de données appelés attributs et de fonction appelés méthode.

Un objet est un représentant d'une classe. C'est un espace mémoire contenant les valeurs des différents attributs, les méthodes étant communes à tous les objets d'une classe.

Exemple 14:

```
class Noeud:
    def __init__(self, x):
        self.valeur = x
    def afficher(self):
        print(self.valeur)
```

Ici, Noeud est une classe, valeur un attribut, afficher et est_egal sont des méthodes de la classe Noeud et a un objet (representant) de la classe Noeud.

```
    def est_egal(self, autre):
        return self.valeur == self.autre
a = Noeud(5)
a.afficher()
```

Définition 15: La programmation orienté objet consiste à utiliser des classes et des objets de ces classes quand on programme.

Remarque 16: Une utilisation massive des classes est de permettre la modularité : on peut avoir une interface pour un type abstrait, rendant l'utilisation du type et la structure de données l'implémentant indépendants.

Exemple 17: En python, le package numpy propose les objets de type numpy.array que l'on crée via la commande a = numpy.array([...]). Un tel objet possède des attributs (exe: a.size) et des méthodes (exe: a.sort()). Cette classe implémente des tableaux de taille fixe et de nombreuses méthodes dessus. On peut les utiliser en ne comprenant rien à leur fonctionnement, n'ayant connaissance que de leur résultat, mais on peut aussi les réimplémenter sans changer l'utilisation de ces tableaux par des millions de personnes.

III-2) Pour résoudre un problème

Une autre utilité de la programmation orienté objet et de représenter un problème avec ses différents objets que l'on fera intéragir entre eux.

Exemple 18: Sur l'exemple 17, on peut rajouter la classe Arbre contenant des noeuds.

```
class Arbre:
    def __init__(self, n, liste_arbre):
        self.noeud = n
        self.fils = liste_arbre
    def afficher(self):
        n.afficher()
        for a in self.fils:
            a.afficher()
```

TP: Reforester l'amazonie

Développement 2 : Representation d'une personne sur une carte en python par des objets, illustrant les différents paradigmes de programmation

IV- Dans la vraie vie ?

IV-1) Le multiparadigme

Dans la vraie vie, la pluyart des langages implémentent plusieurs paradigmes. En effet, python, comme C et OCaml, permettent de faire des boucles while, de faire des tableaux, de faire des structures et des fonctions récursives, et même les fonctions d'ordre supérieure dans une certaine mesure.
On appelle cela le multiparadigme. Néanmoins, certains langages sont plus adaptés à certains paradigmes, eux-mêmes plus adaptés à certaines contraintes.
Des langages comme C, C++, Fortran, python, Java sont des langages impératifs, quand Haskell, ML, OCaml sont fonctionnelles.
De plus python, C++ sont orientés objets.

IV-2) Comparaison des paradigmes

* Pour des structures récursives comme des arbres (ou des graphes peu denses) le paradigme fonctionnel est approprié
* Le paradigme fonctionnel offre également élégance et lisibilité au code avec moins d'instruction "superflu"
* Le caractère intrinsèquement modulaire et sans effet de bord le rend aussi plus facile à tester et sécuriser
C'est en OCaml (ou Coq) qu'est implémenté CompCert, un compilateur C vérifié.
* La programmation impérative est beaucoup plus proche de la machine et rend donc la compilation plus simple, et le développement intelligent potentiellement plus efficace
* Il est aussi très performant pour des structures de données séquentiels et des accès "aléatoires" à des données. Par exemple représenter une matrice, en faire des multiplications etc... paraît beaucoup plus simple en C qu'en OCaml

TP 19 : Implémenter un tas min en C et en OCaml

* L'orienté objet est quant à lui de plus haut niveau et repose souvent sur d'autres paradigmes plus bas niveau.
* Il est souvent utilisé pour représenter des situations complexes grâce à sa modularité.

TP 20 : Implémenter les classes representant un personnage de jeu vidéo, ses objets, ses compétences, etc...

IV-3) Et SQL ?

Il existe néanmoins bien d'autres paradigmes, comme par exemple le paradigme logique, où seul le résultat est presenté par le code, et non la manière de l'obtenir. C'est par exemple le cas du SQL pour les bases de données, où l'exécution n'est pas dicté par la requête, seule son sens l'est, laissant le SGBD se charger du déroulement.