Paradigmes de Programmations : impératif, fonctionnel, objet. Exemples et applications

Introduction : Il existe différentes théories de calcul (λ calcul, machine de Turing, logique...) et différentes modélisations possibles qui conduisent à la création de différents paradigmes de programmation. Chaque langage de programmation repose sur un ou plusieurs paradigmes de programmation

Apparition des paradigmes : Si la programmation est trop compliquée d'un point de vue technique et non à cause du problème que l'on souhaite résoudre alors un nouveau concept de la programmation attend à être découvert.

I Programmation impérative.

Définition : Machine à états est un modèle constitué d'une mémoire centrale et d'une suite d'instructions qui modifie l'état de la mémoire grâce à des affectations successives.

Def : Programmation impérative : Paradigme de programmation où les programmes sont une suite d'instructions encadrées par des structures de contrôle de l'exécution.

Exemple : les langages machines reposent sur la programmation impérative.

Opération élémentaire en programmation impérative :
- séquence d'instruction.
- Affectation.
- condition.
- boucle.
- saut.

Def : Procédure : suite d'instruction qui s'applique sur une entrée (potentiellement vide) pour la modifier ou pour renvoyer un résultat.

On parle de programmation impérative ou procédural

Exemple de procédure : Procédure qui prend en entrée une liste d'entier et qui modifie cette liste pour trier les élément par ordre croissant.

Exemple :
```
def tri_selection(liste):
    n = len(liste)
    for i in range(n-1):
        min = liste[i]
        indice_min = i
        for j in range(i+1, n):
            if liste[j] < min:
                min, indice_min = liste[j], j
        temp = liste[i]
        liste[i] = liste[indice_min]
        liste[indice_min] = temp
```

Avantage des procédures : Permet une meilleur lisibilité
Permet de limiter la répétition de code, et permet la modularité
Facilite le raisonnement pour les preuves et les tests des procédures.

Difficulté de la programmation impérative : A chaque instruction il faut connaître l'état actuelle de la mémoire pour savoir comment elle va être modifier.

Attention : Une procédure ne se comporte pas comme une fonction mathématique.

Exemple :
```
def carre(x):
    x = x**2
    return x

x = 3
y = carre(x)   # y prend la valeur 9 sachant que x
               # est modifié et prend aussi la valeur 9
z = carre(x)   # z = 81 alors qu'on pouvait s'attendre à ce que z soit 9.
```

II Programmation fonctionnelle.

Définition : Le paradigme fonctionnel est celui des langages qui permettent de définir des fonctions au sens mathématique du terme. On demande alors à l'ordinateur de calculer une expression.

Rappel : Une fonction prend en entrée un n-uplet et renvoie une sortie unique.

Exemple. $f : \mathbb{N} \longrightarrow \mathbb{N}$, $n \longmapsto n+1$ fonction successeur.

En programmation fonctionnel on est dépourvue du concept d'état de la mémoire

Avantages :
- pas d'effet de bord provenant d'une modification de la mémoire.
- il est souvent plus simple de prouver des propriétés sur des fonctions mathématique
- le programme final est une suite d'appel de fonction. Il s'agit donc d'une application au sens mathématique du terme.
(- souvent plus simple à comprendre pour des personnes n'ayant jamais programmé)

déf : fonction récursive : une fonction faisant appel dans le corps de son programme à elle même est dite récursive.

Bénéfice de l'utilisation de la pile : l'appel récursif a une fonction permet de stocker un résultat intermédiaire sur la pile.

Exemple :

```
def factoriel (n) :
    assert n >= 0
    if n == 1 :
        return 1
    else :
        return n * factoriel (n-1)
```

// L'expression est exactement la même que la définition mathématique.

La programmation fonctionnel n'a pas de notion de temps : une fonction mathématique ne change pas au cours du temps et retourne toujours pour une même paramètre le même résultat. Les problèmes de concurrence sont donc évité.

Difficulté : Les machines actuelles utilisent des langage machine qui correspondent à des langage impératifs. Il est donc nécessaire de disposer d'un outil de compilation pour traduire les programme fonctionnels en suite d'instruction compréhensible par le processeur. *

Ouverture : La programmation fonctionnelle fait partie de la programmation dite déclarative dont une autre branche est la programmation logique (basé sur les prédicats).
La programmation déclarative essaye au maximum de se rapprocher de la description de l'objectif souhaiter plutôt que du déroulement des étapes de résolution.

* Dev 1 Fonction récursive ; récursive terminal.
Dérécursivité et passage par continuation.

III Programmation orienté objet

La programmation impérative permet de représenter l'évolution d'un état ce qui est utile dans de nombreuse modélisation. Cependant, lorsque l'état est complexe et les modifications nombreuses il peut être nécessaire d'introduire une méthodologie plus évolué.

Définition : Le paradigme objet est celui des langages où l'on modélise un problème par une collection d'objets qui communiquent entre eux par envoi de message.

Historique : Apparition avec le langage Simula en 1967 pour la simulation d'un ensemble de robots dans une usine.

Définition classe : description des données contenues pour chaque instance de la classe et les méthodes permettant de répondre aux messages reçus par une instance de la classe.

Déf : attribut : chaque données d'une classe est appelée un attribut.

Déf : objet : une instance de la classe est appelée un objet.

Exemple :

```
class Message:
    def __init__(self, texte):
        self.texte = texte          # attribut
    def afficher_message(self):     # méthode affichant le texte
        print(self.message)
    def modifier_message(self, texte):   # méthode permettant de modifier le texte
        self.texte = texte
```

Encapsulation : Les données d'un objet peuvent uniquement être modifier et accédé par l'objet lui-même. Limite les risques d'incohérence des données.

Avantage programmation orienté objet : forte modularité. Facilité à diviser le travail. Le code des méthodes dépend de l'objet et peut être modifier sans modifier l'interface de la classe.

Inconvenient : Complexifie le code qui peut paraître plus lourd.

Déf Heritage : une classe peut hériter d'une autre classe, on parle alors de classe fille ou de sous classe. Il est possible d'enrichir ces classe fille en ajoutant des attributs et des méthodes supplémentaires. Il est aussi possible de redéfinir une méthode de la classe mère dans la classe fille.
L'héritage permet d'éviter la duplication de code.

Dev 2 Application de la POO pour l'édition d'une calculatrice.