Les paradigmes de la programmation

Qu'est-ce que c'est ?

Au sens large, un paradigme est un « modèle théorique de pensée qui oriente la recherche et la réflexion scientifiques » selon Larousse. En programmation, un paradigme représente les différentes façons d'organiser le code d'un programme.

La séparation des langages en différents paradigmes est parfois critiqué par des chercheurs en programmation. Leur principal argument est que cette séparation est inéxacte car aucun langage ne peut être strictement classé. Ils pensent que chaque language possède des caractéristiques de différents paradigmes

La Programmation Impérative

La Programmation Impérative est un style de programmation où vous donnez des instructions à l'ordinateur pour effectuer certaines tâches, étape par étape. C'est comme si vous écriviez une liste d'instructions que l'ordinateur doit suivre, une par une, dans un ordre précis.

Imaginons que vous deviez écrire un programme pour calculer la somme de deux nombres en utilisant la Programmation Impérative. Voici à quoi cela pourrait ressembler en Python :

```python
# Définis deux nombres
nombre1 = 5
nombre2 = 7

# Additionne les nombres
somme = nombre1 + nombre2

# Affiche le résultat
print("La somme de", nombre1, "et", nombre2, "est égale à", somme)
```

La Programmation Procédurale

La Programmation Procédurale est un style de programmation où nous organisons notre code en petites étapes ou "procédures" pour résoudre un problème plus complexe. Vous pouvez penser à ces procédures comme à des mini-instructions que l'ordinateur doit suivre pour accomplir une tâche.

Prenons un exemple simple en Python pour calculer la moyenne de trois nombres en utilisant la Programmation Procédurale :

```python
# Définis une fonction pour calculer la moyenne
def calculer_moyenne(nombre1, nombre2, nombre3):
    somme = nombre1 + nombre2 + nombre3
    return somme / 3  # Moyenne


# Appele la fonction avec trois nombres
resultat = calculer_moyenne(10, 15, 20)

# Affiche la moyenne
print("La moyenne des nombres est :", resultat)
```

La Programmation Orientée Objet

La Programmation Orientée Objet *(POO)* est un style de programmation qui se base sur des "objets" pour organiser et gérer des données et les actions associées à ces données. Chaque objet est comme une boîte qui peut contenir des informations (appelées attributs) et des actions (appelées méthodes) que l'objet peut effectuer.

Pour rendre cela plus concret, prenons un exemple avec des animaux. Supposons que nous voulions créer un programme pour représenter des animaux et interagir avec eux. En utilisant la POO, nous pourrions créer une classe "Animal" qui définit les caractéristiques communes à tous les animaux. Voici un exemple en Python :

```python
# Définis la classe Animal
class Animal:
    # Méthode d'initialisation pour définir les attributs de l'animal
    def __init__(self, nom, age):
        self.nom = nom
        self.age = age

    # Méthode pour faire parler l'animal
    def parler(self, son):
        print(f"{self.nom} dit {son}")


# Création d'une instance d'un objet
mon_animal = Animal("Rex", 5)

# Appel de la méthode 'parler'
mon_animal.parler("Ouaf ouaf !")
```

La Programmation Parallèle

La Programmation Parallèle est une approche en informatique qui consiste à exécuter plusieurs tâches ou parties d'un programme en même temps, plutôt que de les exécuter séquentiellement l'une après l'autre. Cela permet d'exploiter pleinement la puissance de traitement des ordinateurs modernes qui ont plusieurs cœurs de processeur.

Imaginez que vous ayez une équipe de personnes qui construisent un puzzle géant. En Programmation Parallèle, chaque personne peut travailler sur une partie différente du puzzle en même temps, accélérant ainsi la finition du puzzle.

Un exemple simple en Programmation Parallèle pourrait être de calculer la somme de plusieurs nombres. En utilisant la Programmation Parallèle, vous pourriez répartir la tâche de calcul entre plusieurs processeurs ou cœurs de votre ordinateur, de sorte que chaque processeur calcule une partie de la somme en même temps.

En réalité, la mise en œuvre de la Programmation Parallèle est plus complexe et nécessite des outils spécifiques, mais l'idée principale est d'effectuer plusieurs tâches en même temps pour gagner en efficacité et en vitesse. C'est particulièrement utile pour des tâches intensives en calcul, telles que la simulation scientifique, l'inférence en intelligence artificielle ou le rendu d'images dans les jeux vidéo.

La Programmation Déclarative

La Programmation Déclarative est un style de programmation où vous spécifiez ce que vous voulez accomplir, plutôt que de dire comment le faire de manière détaillée. Au lieu de donner une séquence d'instructions exactes à l'ordinateur, vous décrivez le résultat souhaité et laissez l'ordinateur trouver la meilleure façon de le faire. C'est un peu comme donner des ordres à un assistant en lui disant ce que vous voulez, sans lui dire comment le faire étape par étape.

Prenons un exemple simple pour illustrer la Programmation Déclarative. Imaginons que vous vouliez obtenir une liste des nombres pairs dans une liste de nombres donnée. En utilisant la Programmation Déclarative en Python, cela pourrait ressembler à ceci :

```python
# Liste de nombres
nombres = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Utilisation d'une expression en compréhension de liste pour obtenir les nombres pairs
nombres_pairs = [x for x in nombres if x % 2 == 0]

# Affiche la liste des nombres pairs
print("Nombres pairs :", nombres_pairs)
```

Dans cet exemple, au lieu de dire à l'ordinateur comment vérifier si un nombre est pair (par exemple, en utilisant une boucle for et une condition if), nous utilisons une expression en compréhension de liste pour déclarer notre intention. En langage courant, ça donnerait : "je veux une liste des nombres x pour lesquels x % 2 == 0".

La Programmation Déclarative est couramment utilisée avec des langages et des outils qui permettent de décrire ce que vous voulez faire de manière concise et expressive. Cela rend souvent le code plus lisible et plus facile à comprendre, car il se rapproche davantage de la manière dont nous exprimons nos pensées.

La Programmation Logique

La Programmation Logique est un style de programmation basé sur la logique mathématique. Au lieu de donner à l'ordinateur des instructions précises sur comment effectuer une tâche, vous lui fournissez des règles logiques et des faits, puis vous lui demandez de résoudre des problèmes en utilisant ces règles. C'est comme si vous écriviez des règles du jeu et que l'ordinateur jouait en respectant ces règles pour obtenir une solution.

Pour expliquer cela plus clairement, prenons un exemple simple en utilisant un langage de programmation logique appelé Prolog. Imaginons que nous voulions créer un programme pour déterminer si un animal est un oiseau en fonction de certaines caractéristiques. Voici comment cela pourrait être fait en Prolog :

```
% Définir les faits : Quels animaux sont des oiseaux
oiseau(canari).
oiseau(aigle).
oiseau(pingouin).

% Définir des règles logiques
peut_voler(X) :- oiseau(X).
a_des_ailes(X) :- oiseau(X).
a_des_plumes(X) :- oiseau(X).

% Posez des questions pour résoudre des problèmes
?- peut_voler(canari).
```

Dans cet exemple, nous définissons d'abord des faits en spécifiant quels animaux sont des oiseaux. Ensuite, nous définissons des règles logiques en disant que si un animal est un oiseau, alors il peut voler, a des ailes et a des plumes. Enfin, nous posons une question à l'ordinateur pour savoir si un canari peut voler. L'ordinateur utilise les règles et les faits que nous avons fournis pour déterminer la réponse, qui est "true" (vrai) dans ce cas, car un canari est un oiseau.

La Programmation Logique est particulièrement utile pour résoudre des problèmes où la logique joue un rôle clé, comme la résolution de problèmes mathématiques, la planification de tâches, ou l'analyse de données.

La Programmation Fonctionnelle

La Programmation Fonctionnelle est un style de programmation basé sur le concept de fonctions, tout comme les fonctions en math. Dans ce style de programmation, les fonctions sont des "citoyens de première classe", ce qui signifie qu'elles peuvent être traitées comme des valeurs.

Attention : la différence est assez suptile avec la programmation procédurale surtout que Python regroupe ces deux types de paradigmes.

Imaginons que nous voulions créer un programme pour calculer la somme des carrés de plusieurs nombres. En utilisant la Programmation Fonctionnelle en Python, cela pourrait ressembler à ceci :

```
# Définis une fonction pour calculer le carré d'un nombre
def carre(nombre):
    return nombre * nombre


# Définis une fonction pour calculer la somme des carrés
def somme_des_carres(liste_de_nombres):
    somme = 0
    for nombre in liste_de_nombres:
        somme += carre(nombre)
    return somme


# Utilise la fonction pour calculer la somme des carrés
nombres = [1, 2, 3, 4, 5]
resultat = somme_des_carres(nombres)
print("La somme des carrés est :", resultat)
```

La Programmation Réactive

La Programmation Réactive est un style de programmation qui se concentre sur la gestion des flux de données et des événements. Dans ce style de programmation, le programme réagit aux changements de données ou d'événements en temps réel, plutôt que d'exécuter des tâches de manière séquentielle.

Un exemple simple de Programmation Réactive serait la gestion d'une boîte de réception d'e-mails. Plutôt que de constamment actualiser la boîte de réception pour vérifier s'il y a de nouveaux e-mails, la Programmation Réactive permettrait au programme de réagir immédiatement lorsqu'un nouvel e-mail arrive.

Un exemple plus concret en Python serait l'utilisation de la bibliothèque tkinter pour créer une interface utilisateur réactive. Voici un exemple très basique d'une fenêtre qui réagit lorsque vous cliquez sur un bouton :

```
import tkinter as tk

# Fonction appelée lorsque le bouton est cliqué
def bouton_clique():
    label.config(text="Bouton cliqué!")

# Création de la fenêtre
fenetre = tk.Tk()
fenetre.title("Programmation Réactive")

# Création d'un bouton
bouton = tk.Button(fenetre, text="Cliquez-moi!", command=bouton_clique)
```

```
bouton.pack()

# Création d'une étiquette
label = tk.Label(fenetre, text="")
label.pack()

# Lancement de la boucle d'événements
fenetre.mainloop()
```

Dans cet exemple, nous avons créé une fenêtre avec un bouton. Lorsque le bouton est cliqué, il déclenche une fonction (`bouton_clique`) qui modifie le texte d'une étiquette pour afficher "Bouton cliqué!". Cela montre comment la fenêtre réagit en temps réel aux événements (cliquer sur le bouton) sans attendre que l'utilisateur fasse autre chose.

La Programmation Réactive est souvent utilisée dans le développement d'applications Web pour gérer les mises à jour en temps réel, dans les jeux vidéo pour gérer les mouvements des personnages en fonction des entrées des joueurs, et dans de nombreuses autres applications qui nécessitent une réponse rapide aux événements.

Les autres Paradigmes de Programmation

Il existe plusieurs autres paradigmes moins connus, en voici quelques-un :

1. Programmation Orientée Aspect (POA) : séparation des préoccupations transversales (en anglais, cross-cutting concerns) liées à l'aspect (ex : gestion de log) du reste du code en les encapsulant dans des aspects.
2. Programmation Générique : création de code générique capable de fonctionner avec différents types de données sans avoir à les spécifier explicitement.
3. Programmation Concurrente : gestion de plusieurs tâches ou processus en même temps pour améliorer les performances ou la réactivité d'une application.
4. Programmation Contrainte : résolution de problèmes tout en définissant des contraintes sur les variables.
5. Programmation par Contrat : chaque composant logiciel définit un contrat qui spécifie ses préconditions et ses postconditions.
6. Programmation Visuelle : interfaces graphiques permettant de créer des programmes en plaçant des composants visuels.