Les paradigmes de programmations

Table des matières

1) Les principaux paradigmes de programmation

1.1) Les paradigmes

a) Introduction

Le mot paradigme vient du grec ancien $\pi\alpha\rho\alpha\delta\epsilon\iota\gamma\mu\alpha$ qui signifie modèle. En informatique, ce mot a été adopté pour distinguer les différentes façon d'envisager l'écriture d'un programme.

Cependant, quelque soit le programme et son mode d'écriture, il est traduit sous la forme de code machine impératif : donner à la machine des instructions élémentaires pour lui dire comment faire ses allers-retours entre le processeur et la mémoire.

Ce style (paradigme) de programmation impératif étant au coeur de la machine, la majorité des langages l'a adopté.

b) Pourquoi d'autres paradigmes

En 1954, John Von Neumann, lorsqu'on lui a présenté le FORTRAN, le premier véritable langage de programmation aurait déclaré : "Pourquoi voudrait-on autre chose que le langage machine ?"

A cette question, on peut répondre qu'il est préférable de rapprocher le langage de la pensée du programmeur plutôt que de lui demander de penser en fonction de la machine. Or l'être humain a différentes manières de penser : différents types de langages ont donc été proposés.

La tendance actuelle est de ne pas attacher un langage à un paradigme mais de permettre l'adoption de plusieurs paradigmes au sein d'un même langage afin de pouvoir s'adapter à différentes situations.

1.2) Exemple de paradigmes de programmation

a) La variétés des paradigmes

Des dizaines de paradigmes de programmation ont été décrits, outre le paradigme impératif. Son direct opposé est le paradigme déclaratif où l'on décrit ce que l'on veut faire sans demander au programmeur de donner des instructions précises à la machine. On trouve notamment dans cette catégorie la programmation logique, fonctionnelle, le SQL.

A la fin des années 1950, les concepts de la programmation orientée objets apparaissent. Ils auront un impact très important sur l'informatique d'aujourd'hui et sont présentés plus en détails dans la suite de ce cours.

La programmation orientée acteurs et plus généralement la programmation permettant de gérer la concurrence a pris énormément d'importance depuis une dizaine d'années pour tenir compte du changements drastiques dans l'architecture des machines qui disposent de multiples coeurs ou processeurs. Des langages tels que Erlang, Elixir, Go et Scala en sont l'illustration.

La liste ne saurait être exhaustive mais il faut évoquer la programmation qui révolutionne le monde depuis 2014 : l'apprentissage automatique. On programme des machines pour qu'elles pensent et programment elles-même.

b) La programmation modulaire

Dans le cadre de la programmation classique de type impératif en langage python, C, Fortran, une notion fondamentale s'est dégagée : celle de la modularité à travers l'usage de fonctions. Un programme est alors constitué de de deux types d'entités distinctes : les données et les fonctions.

Les fonctions peuvent être utilisées hors de leur contexte initial mais il faut alors redéfinir les données à manipuler (moins d'utiliser des variables globales, ce qui est connu comme étant une source intarissable de déboires).

Cette conception donne la primauté au traitements. Les briques logicielles sont essentiellement des morceaux de code réutilisables n'intégrant pas les données. Cette dissociation des données et des traitements restreint l'autonomie de ces briques logicielles et par la même leur réutilisabilité, ce à quoi la notion d'objet va essayer de remédier.

c) Le paradigme objet

Le paradigme objet a été élaboré par les norvégiens Ole-Johan Dahl et Kristen Nygaard au débit des années 1960 à travers le langage Simula et repris et approfondi par l'américain Alan Kay dans les années 1970 dans le développement du langage SmallTalk qui est devenu une référence dans le domaine. La programmation orientée objet se retrouve dans les langages aujourd'hui comme C++, C#, Java, Python, Eiffel, Ruby ou OCaml qui sont des langages à classes.

Dans la programmation orientée objet, pour remédier aux difficultés posée par la programmation modulaire, on considère un nouveau type d'entité : l'objet qui intègre les données et les traitements. Les données trouvent ainsi leurs place au sein de ces briques logicielles. L'objet prend une certaine autonomie et est en mesure de dialoguer avec les autres objets de son univers.

2) Programmation fonctionnelle

Le paradigme fonctionnel offre un excellent modèle de décomposition d'un problème. Il est adapté aux besoins modernes des machines multiprocesseurs et a pour but de de se rapprocher du mode de pensée humain.

La programmation fonctionnelle est née en même temps que la programmation impérative (Lisp dans les années 1950) mais est longtemps restée marginale, plus académique qu'industrielle (langage ML et dérivés, Haskell). Cependant, la

plupart des langages apparus au XXI^{ème} siècle sont fortement influencés par un style fonctionnel. Python, par exemple, propose le filtrage par motif structurel qui est un des points forts de la programmation fonctionnelle.

Des entreprises comme Google (avec Go), Mozilla (avec Rust),Android (avec Kotlin), Apple (avec Swift), Whatsapp (avec Erlang), Tweeter (avec Scala) entre autres ont lancé ou utilisent de nouveaux langages qui ont intégré de nombreux aspect de la programmation fonctionnelle notamment pour faciliter le traitement de gros flux de données et la programmation concurrente, car ce n'est plus tant la puissance des processeurs qui évolue, mais leur nombre, et la possibilité de travailler avec des logiciels en ligne (SaaS : software as a service).

Le paradigme fonctionnel cherche à éviter au maximum les effets de bord, dit autrement, en programmation fonctionnelle on va éviter de modifier les valeurs associées à des variables.

Pour ce faire, on va chercher au maximum à utiliser les fonctions (d'où le nom de programmation fonctionnelle), mais ces fonctions ne devront pas modifier les variables : en programmation fonctionnelle, on s'efforce de coder des fonctions qui ne modifient pas l'état courant des variables. Les fonctions utilisées en programmation fonctionnelle sont parfois appelées "fonction pure" : le résultat renvoyé par une fonction pure doit uniquement dépendre des paramètres passés à la fonction et pas des valeurs externes à la fonction (elle ne doit pas non plus engendrer d'effet de bord) :

Exercice 1 : A votre avis, laquelle de ces fonctions ci-dessous est-elle une fonction pure ?

```
i = 5
def fct():
  if i > 5:
    return True
  else :
    return False

fct()
```

ou bien :

```
def fct(i):
  if i > 5:
    return True
  else :
    return False
fct(5)
```

Même si certains langages de programmation ont été conçus pour "imposer" au programmeur le paradigme fonctionnel (Lisp, Scheme, Haskell...), il est tout à fait possible d'utiliser le paradigme fonctionnel avec des langages de programmation plus "généralistes" (Python par exemple).

Exemple :

Selon vous, le programme ci-dessous respecte-t-il le paradigme fonctionnel ?

```
l = [4,7,3]
def ajout(i):
  l.append(i)
```

```
def ajout(i,l):
        tab = l + [i]
  return tab
```

La fonction ajout ne modifie aucune variable, elle crée un nouveau tableau (tab) à partir du tableau l et du paramètre i (le signe + permet de créer un nouveau tableau, ce nouveau tableau est constitué des éléments contenus dans le tableau l auxquels on ajoute la valeur i), la fonction renvoie le tableau ainsi créé.

D'une façon plus générale, la méthode append de Python ne respecte pas le paradigme fonctionnel puisque append modifie une donnée existante. Le paradigme fonctionnel va amener le programmeur non pas à modifier une valeur existante, mais plutôt à créer une nouvelle grandeur à partir de la grandeur existante : une grandeur existante n'est jamais modifiée, donc aucun risque d'effet de bord.

3) Programmation objet

3.1) Architecture objet

La conception objet possède un aspect intuitif fort dans la mesure où l'on s'efforce de calquer la représentation informatique sur les entités physiques ou conceptuelle apparaissant dans le processus à modéliser. Ce procédé s'appelle la réification et consiste à partir des objets du monde réel.

Les termes d'objets et d'architectures objet évoquent un volonté de se rapprocher du monde physique de la construction et de l'assemblage. Ce rapprochement est même plus marqué à travers l'usage des design pattern (ou patron de conception) qu'utilisent les programmeurs déjà expérimentés. L'héritage y joue un rôle important.

3.2) Principes et terminologie objet

a) Objets et encapsulation

Un objet comprend un partie figée qui représente son état et les liens qui l'unissent à d'autres objets et une partie dynamique qui décrit son comportement, c'est-à-dire toutes les opérations qu'on peut lui appliquer ainsi que sa manière de réagir aux événements de l'environnement.

Sa partie fixe est constitué d'un ensemble de champs (ou d'attributs) et sa partie dynamique d'opérations appelées méthodes. Certaines de ces méthodes constituent la partie visible de l'objet. C'est par elle que l'on s'adresse à lui. Elles constituent autant de services qu'on peut demander à l'objet de fournir.

Certains champs peuvent être partiellement ou totalement inaccessibles à d'autres objets. C'est alors qu'on parle d'encapsulation.

b) Classe et instances

Quand les objets possèdent une structure et des comportements en commun, on peut les regrouper sous forme de classe. Une classe est une sorte de moule à partir duquel sont produits les objets que l'on appelle les instances. Les instances d'une même classe ne diffèrent les une des autres que par les valeurs de leurs attributs.

La programmation objet consiste donc à définir les bonnes classes dotées de leurs champs et méthodes, puis à les instancier pour créer des objets et les faires interagir.

4) Utiliser le bon paradigme de programmation

Il est important de bien comprendre qu'un programmeur doit maîtriser plusieurs paradigmes de programmation (impératif, objet ou encore fonctionnelle). En effet, il sera plus facile d'utiliser le paradigme objet dans certains cas alors

que dans d'autres situations, l'utilisation du paradigme fonctionnelle sera préférable. Être capable de Choisir le "bon" paradigme en fonction des situations fait partie du bagage de tout bon programmeur.