NSI Terminale - Programmation

Paradigmes de programmation

qkzk

2020/05/04

Dans ce court chapitre nous allons définir ce qu'est un paradigme de programmation.

Cette notion fait référence à la *diversité et l'unité* des langages de programmation, abordée en première.

Paradigme de programmation

Définition :

> Un paradigme de programmation est une façon d'approcher la programmation informatique et de traiter les solutions aux problèmes et leur formulation dans un langage de programmation approprié.

Un paradigme de programmation fournit la vue qu'a le développeur de l'exécution de son programme. Par exemple, en **programmation orientée objet**, les développeurs peuvent considérer le programme comme une *collection d'objets en interaction*, tandis qu'en **programmation fonctionnelle** un programme peut être vu comme une *suite d'évaluations de fonctions sans états*. Lors de la programmation d'ordinateurs ou de systèmes multi-processeurs, la **programmation orientée processus** permet aux développeurs de voir *les applications comme des ensembles de processus agissant sur des structures de données localement partagées.*

Exemples de paradigmes de programmation

Programmation impérative

La programmation impérative décrit les opérations en séquences d'instructions exécutées par l'ordinateur pour modifier l'état du programme. Ce type de programmation est le plus répandu parmi l'ensemble des langages de programmation existants

Exemple

```
x = 1
y = x + 2
z = x * y
```

On déclare différentes variables auxquelles on attribue des valeurs.

Les opérations sont effectuées sur les variables elles-mêmes.

On peut aussi déclarer une fonction et l'exécuter sur des variables

```
def former_presentation(prenom, nom, ville):
  return prenom + " " + nom + " habite à " + ville

pres = former_presentation("Robert", "Duchmol", "Miami")
```

La variable `pres` pointe vers la chaîne de caractères :

```
"Robert Duchmol habite à Miami"
```

L'inconvénient immédiat de la programmation impérative et la multiplication des variables.

On doit souvent se questionner sur l'état d'une variable durant le déroulé des instructions.

Les fonctions ont-elles connaissance de cet état ? Est-ce une variable globale ?

Pourrait-on regrouper certaines de ces variables en un... objet ?

Fortran (IBM 1954) est un des premiers langages de programmation impérative. À ce jour il est toujours utilisé par les scientifiques pour la qualité de ses librairies.

Plus tard, C est devenu la référence de la programmation impérative. La majorité des applications de bureau que vous utilisez sont écrites en C. Tout comme le noyau des systèmes d'exploitation les plus courants.

Python, Java, Javascript... implémentent tous la programmation impérative.

Programmation orientée objet

La programmation orientée objet est le paradigme de programmation dominant en 2020. Elle a vu le jour durant les années 70 et sa première implémentation reconnue est SmallTalk.

Python, Java, C++ sont des langages qui implémentent la programmation objet.

La programmation objet consiste en la définition et l'interaction de briques logicielles appelées objets ; un objet représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre.

Il possède une structure interne et un comportement, et il sait interagir avec ses pairs. Il s'agit donc de représenter ces objets et leurs relations ; l'interaction entre les objets via leurs relations permet de concevoir et réaliser les fonctionnalités attendues, de mieux résoudre le ou les problèmes. Dès lors, l'étape de modélisation revêt une importance majeure et nécessaire pour la POO. C'est elle qui permet de transcrire les éléments du réel sous forme virtuelle.

```
class Personne:
  def __init__(self, prenom, nom, ville):
    self.prenom = prenom
    self.nom = nom
    self.ville = ville

  def presenter(self):
    return self.prenom + " " + self.nom + " habite à " + self.ville

robert = Personne("Robert", "Duchmol", "Miami")
pres = robert.presenter()
```

Ici, `Personne` désigne un objet personne qui a une méthode : `presenter`

Nom, prénom et ville pourraient eux mêmes être des objets.

Dans certains langages, comme Python, tout est objet (les entiers, les fonctions, les classes elles mêmes).

Cet exemple n'est pas très utile : une classe qui n'a que deux méthodes dont `__init__`... n'est qu'une fonction !

En pratique les classes ont parfois des dizaines de méthodes... et certaines classes héritent les unes des autres.

Concevoir un programme dans ce paradigme consiste donc d'abord à définir les objets dont on aura besoin au travers de leur intéraction.

Que doit faire la personne ? Comment intéragit-elle avec l'objet ville ?

Ces intéractions permettent de définir les *méthodes*.

Programmation fonctionnelle

La programmation fonctionnelle est un paradigme de programmation de type déclaratif qui considère le calcul en tant qu'évaluation de fonctions mathématiques.

Comme le changement d'état et la mutation des données ne peuvent pas être représentés par des évaluations de fonctions la programmation fonctionnelle ne les admet pas, au contraire elle met en avant l'application des fonctions, contrairement au modèle de programmation impérative qui met en avant les changements d'état.

Alors que l'origine de la programmation fonctionnelle peut être trouvée dans le lambda-calcul, le langage fonctionnel le plus ancien est Lisp, créé en 1958 par McCarthy. Lisp a donné naissance à de nombreuses variantes variantes telles que Scheme (1975).

Des langages fonctionnels plus récents tels Haskell (1987), OCaml, Scala (2003) ont vu le jour et sont employés dans l'industrie.

La programmation fonctionnelle présente de nombreux atouts devant la programmation objet.

La programmation fonctionnelle s'affranchit de façon radicale des effets secondaires (ou effets de bord) en interdisant toute opération d'affectation.

On s'assure ici d'éviter un problème majeur : *les effets de bord*

En *programmation impérative*, une **même fonction appelée à différents moments avec les mêmes paramètres peut renvoyer différents résultats.**

Exemple en Python

```
tableau = [1]

def cumuler(entier):
  tableau[0] = tableau[0] + entier
  return tableau[0]
```

Lorsqu'on appelle une première fois `cumuler(2)`, on obtient 3.

Lorsqu'on la rappelle une seconde fois, on obtient 5.

Pourquoi ?

Parce que la fonction a un *effet de bord* consistant à modifier le contenu de `tableau`.

Aussi, *tester* cette fonction demande de définir d'abord l'état de la machine (le contenu de `tableau`) et d'en vérifier l'état après l'exécution de la fonction.

Ce n'est pas très génant que la "machine" ne contient qu'une variable.

C'est plus difficile quand elle est le résultat de milliers de lignes d'instructions.

C'est typiquement ce que souhaite éviter la programmation fonctionnelle.

Nous approfondirons le sujet plus longuement.

Retenez simplement que Python n'est pas un langage fonctionnel... mais qu'il en implémente une partie.

En particulier, les listes par compréhension, les fonctions lambda mais aussi la fonction `map` ou encore `enumerate` s'inspirent de la programmation fonctionnelle.

Nous étudierons plus longuement la programmation fonctionnelle au travers d'un exemple implémenté en Python.

Programmation événementielle

la *programmation événementielle* est un paradigme de programmation fondé sur les événements. Elle s'oppose à la programmation séquentielle (impératif, objet). Le programme sera principalement défini par ses réactions aux différents événements qui peuvent se produire, c'est-à-dire des changements d'état de variable, par exemple l'incrémentation d'une liste, un mouvement de souris ou de clavier.

Elle permet de réaliser des opération asyncrones comme, par exemple, réagir à une action venant de l'extérieur :

"si la souris quitte l'écran, afficher un popup '*ne partez pas, abonnez vous !*' "

Javascript est le langage le plus utilisé qui implémente la programmation événementielle.

En Javascript, par exemple, il est totalement impossible de “figer” l’exécution durant un temps précis.

En Python il suffit de faire :

```
from time import sleep

print("Dans 3 secondes ça explose !")
sleep(3)
print("BOOM !")
```

En javascript, le moteur continuera toujours à effectuer des calculs en attendant la réalisation d’un événement.

Par exemple si la page a fini de charger les images et qu’elle attendait ça pour les afficher, vos images peuvent apparaître soudainement durant le déroulement d’une instruction.

La programmation événementielle peut être modélisée dans les jeux vidéos, par exemple, avec une boucle infinie.

```
Tant que Vrai
  lire les actions du joueur
  calculer le nouvel état du jeu
  dessiner le monde
Fin tant que
```