Paradigmes de programmation

Un paradigme de programmation est une façon d'approcher la programmation.
Nous allons détailler trois paradigmes principaux.

- La programmation impérative : la brique élémentaire est l'instruction.
- La programmation fonctionnelle : la brique élémentaire est l'expression.
- La programmation orientée objet : la brique élémentaire est l'objet (ou la classe).

La programmation évènementielle utilisée notamment dans les interfaces graphiques est aussi un paradigme de programmation mais les codes associés sont généralement orientés objet. Le programme est défini par des évènements, des changements d'état des variables qui peuvent être causés par le mouvement de la souris, le clavier…

Compléments : https://fr.wikipedia.org/wiki/Paradigme_(programmation)

Programmation impérative

C'est le paradigme de programmation le plus classique, le premier historiquement.
Il décrit les opérations en séquences d'instructions exécutées par l'ordinateur pour modifier l'état du programme, défini par le contenu de la mémoire à un instant donné, comme une recette de cuisine.

La grande majorité des langages de programmation est impérative car les micro-processeurs ont été conçus pour exécuter des séquences d'instructions.

Quelques langages impératifs

- Langage machine, langage des premiers ordinateurs avec un jeu d'instruction minimal
- A-0 : premier compilateur écrit en 1951 par Grace Murray Hopper (une femme !)
- Fortran : premier langage de programmation digne de ce nom (avec variables nommées, sous-programmes, expressions complexes…). Développé par John Backus chez IBM à partir de 1954, Fortran est toujours utilisé aujourd'hui, ce qui en fait le langage ayant eu la plus grande longévité.
- Basic : langage interprété conçu en 1963 comme une version simplifiée du Fortran destiné aux débutants. Très populaire pendant des années, il a été délaissé car la pratique du Basic menait à l'écriture de programmes non structurés.
  "It is practically impossible to teach good programming to students that have had a prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration." Dijkstra, 1975
  "The teaching of BASIC should be rated as a criminal offence: it mutilates the mind beyond recovery." Dijkstra, 1984
- C : inventé au début des années 70 dans les laboratoires Bell par Dennis Ritchie et Ken Thompson pour réécrire UNIX. C'est l'un des langages les plus utilisés, encore aujourd'hui. La syntaxe du C a été reprise par le C++, le Java, le Javascript et le PHP.

- Python : de Guido van Rossum en 1990
- PHP : de Rasmus Lerdorf en 1994
- Java : de Sun Microsystems en 1995
- Javascript : de Brendan Eich, Netscape Navigator en 1995

La plupart de ces langages acceptent d'autres paradigmes de programmation.

Programmation fonctionnelle

Le paradigme fonctionnel, c'est utiliser un langage basé sur des expressions. Une expression est la représentation arborescente d'un calcul, composée à l'aide de fonctions. Évaluer une expression permet d'obtenir le résultat du calcul. Les fonctions peuvent apparaître comme paramètres, retours ou données d'un programme.

En programmation fonctionnelle pure, il n'y a que des évaluations de fonctions, et l'évaluation d'une fonction ne dépend que de la valeur de ses paramètres, comme en mathématiques, et pas de facteurs externes, ce qui exclut les changements d'état qui pourraient provoquer des effets de bord. Les « variables » sont donc constantes, on ne modifie pas leurs valeurs, mais on peut créer de nouveaux espaces mémoires par appels récursifs.
Il n'y a donc pas de compteur, et pas de boucles. Elles sont remplacées par les appels récursifs de fonctions.

Une fonction "impure", produisant ou dépendant de facteurs externes, est dite réaliser des effets de bords. Exemple :

```
a = [0]
def f():
    a[0] += 1
    return a[0]
```

Le programme ci-dessus crée des changements d'état (modification de la variable a). C'est interdit en programmation fonctionnelle. Le résultat de f() dépend de facteurs externes. C'est le nombre de fois où la fonction f est appelée.
f() - f() peut renvoyer 1 ou –1 selon l'ordre dans lequel s'effectuent les calculs. Python calcule de gauche à droite, mais dans d'autres langages, le résultat de ce calcul n'est pas spécifié. En C, on obtient parfois 1, parfois -1 selon le compilateur.

La fonction random() est un autre exemple de fonction impure : le résultat dépend du moment de l'appel.

La lecture ou l'écriture dans un fichier, une base de données, la dépendance à un générateur aléatoire, à une mesure externe (heure, lieu …) créent des effets de bord.

Quelques langages fonctionnels

- Lisp crée en 1958 par McCarthy
- Scheme, 1975
- Haskell, 1990
- OCaml, 1996

Les caractéristiques de ces langages encouragent la programmation fonctionnelle, mais OCaml par exemple, permet aussi la programmation impérative et orientée objet.

Intérêt de la programmation fonctionnelle

Un code sans effet de bord est plus fiable, plus facile à maintenir, à tester, à réutiliser. Un programmeur qui comprend la programmation fonctionnelle a des cordes supplémentaires à son arc, il comprend mieux la programmation d'une manière générale et peut utiliser certaines techniques de programmation fonctionnelle sans forcément faire de la programmation fonctionnelle pure.

Utiliser la programmation fonctionnelle

Il s'agit de faire rentrer toutes les constructions qu'on connait en programmation dans un langage basé sur des expressions.

Fonctions : On accepte les fonctions qui peuvent être vues comme des expressions plus complexes. La fonction lambda en Python est vraiment une expression. La différence entre une fonction définie par lambda et une fonction définie par def est que la première est anonyme.

Instructions conditionnelles : if, then, else peuvent être inclus dans une expression, par exemple en Python :

Méthode impérative (avec instructions)

```
def abs(x) :
    if x > 0:
        return x
    else:
        return -x
```

Méthode fonctionnelle (avec expression)

```
def abs(x) :
    return x if x > 0 else -x
```

Suppression des boucles : Toute boucle est convertible en appel récursif et vice versa.

```
def fun_with_loop(n):
    res = 0
    for i in range(n):
        res += i
    return res
```

On analyse le programme :

- état initial : res = 0
- à la fin on renvoie la valeur de res
- à chaque exécution de la boucle, i augmente de 1 et res augmente de i

On le transforme en programme récursif qui va faire la même chose :

```
def fun_with_rec(n):
    def f_rec(res, i):
        if i<n:
            return f_rec(res+i, i+1)
        else:
            return res
    return f_rec(0, 0)
```

Programmation orientée objet

La programmation orientée objet (POO) consiste en la création et l'interaction de briques logicielles appelées objets. Un objet peut représenter à peu près n'importe quoi : un concept, une idée, une personne, un pays, une carte, un jeu… Il est doté d'attributs (variables, propriétés) et de méthodes pour agir sur lui-même ou interagir avec d'autres objets. On modélise ainsi des éléments réels par des objets virtuels

Un exemple d'objet : un point du plan nommé p1

<u>En Javascript</u>

```
var p1 = {} ;     // création d'un objet vide
p1.x = 2 ;        // ajout de données à l'objet, ici ses coordonnées
p1.y = 3 ;
p1.get_coords = function(){console.log(this.x, this.y)} ;      // ajout d'une méthode à l'objet
```

Si on crée un autre point, on doit redéfinir la fonction get_coords. Si on rajoute d'autres méthodes (représenter un point, calculer la distance entre deux points, …), on devra les ajouter à chaque point. Si on crée 10 points, c'est très lourd !
C'est parce que Javascript n'est pas un vrai langage orienté objet. On peut programmer des objets, mais il manque les classes qui permettent de définir les attributs et les méthodes pour toute une série d'objets ayant les mêmes caractéristiques.

<u>En Python</u>

On définit une classe Point. On peut ensuite créer des points en une ligne. Ils auront tous les mêmes attributs et méthodes.

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def get_coords(self):
        return self.x, self.y

p1 = Point(2, 3)
p2 = Point(4, -1)
```

Quelques langages orientés objets

- Simula, crée en 1967, est le premier langage à classes et fournit les premiers ingrédients qui serviront à développer la POO.
- Smalltalk, développé durant les années 70 et publié en 1980, est un des premiers langages connus orienté objet.
- C++ est un C amélioré (incrémenté !) sorti en 1983. Il reprend les principes du C mais lui rajoute de nouveaux éléments, notamment les classes et les objets.
- Python, PHP, Java (années 90) sont également orientés objet. La langage Java est couramment utilisé pour développer des applications sur smartphone Android.
- Swift apparu en 2014 est un langage proche du C servant notamment à développer des applications sur iOS (iPhone).

Nous reviendrons plus longuement sur la POO qui fait l'objet du prochain cours.

UN EXEMPLE DE PROGRAMMATION EN PYTHON AVEC LES TROIS PARADIGMES

Les habitants d'un archipel veulent construire des ponts entre leurs îles, de sorte que l'on puisse aller de toute île à une autre en passant par ces ponts. Pour chaque paire d'îles (I_1; I_2), ils calculent le coût C(I_1; I_2) pour construire un pont entre elles.

On suppose que les coûts de construction des ponts ont été entrés dans un tableau *couts*, une liste de listes de flottants, de taille $n \times n$: couts[i][j] est le coût d'un pont entre l'ile i et l'ile j.
Ce tableau contient des float('inf') sur les éléments diagonaux et lorsque la construction n'est pas possible.

Programmer un algorithme glouton résolvant ce problème (en commençant par le paradigme impératif, sauf si vous savez déjà programmer en POO).