Chapitre 1 : Diversité et unité des langages de programmation

1. Introduction

Il n'existe pas de langage de programmation « universel ». En effet, si l'on peut, en général, tout programmer avec n'importe quel langage, certains langages seront plus adaptés à la programmation de telle ou telle tâche. Le langage Python, par exemple, est très utilisé pour la mise en place rapide de programmes. Cependant, dès lors que l'on a besoin de performances plus élevées, ce n'est plus le langage le plus adapté. Un exemple typique est la bibliothèque numpy qui est programmée en langage C (bien qu'elle soit prévue pour être ensuite utilisée avec Python). En parallèle, l'évolution de techniques concernant différents aspects des langages de programmation (comme la programmation orientée objets, la gestion de la mémoire, des exceptions, des adresses mémoires) fait que l'on désire définir de nouveaux langages (ou d'en faire évoluer d'autres) pour inclure de nouvelles techniques.

Il en résulte que depuis la création de l'informatique, on a vu sans cesse des nouveaux langages être créés, souvent inspirés de langages déjà existant et incorporant ou développant de nouveaux aspects. L'étude de l'évolution des langages de programmation permet ainsi d'illustrer le développement et l'évolution de différentes techniques ainsi que l'émergence de nouvelles idées.

Petit listing ...

Voici la liste, non exhaustive de quelques langages de haut niveau :

- 1954, Fortran premier langage de haut niveau. le traducteur de formules (FORmula TRANslator), inventé par John Backus et al. Principalement utilisé pour le calcul scientifique.
- 1958, Algol (ALGorithmic Oriented Language).
- 1959, Cobol spécialisé dans la programmation d'application de gestion (COmmon Business Oriented Language)
- 1964, BASIC (Beginner's All-purpose Symbolic Instruction Code)
- 1970, Pascal, héritier d'Algol en plus convivial : utilisé pour l'enseignement
- 1972, C : langages de programmation système, développé par Dennis Ritchie et Ken Thompson pour le développement d'Unix aux laboratoires Bell entre 1969 et 1973.
- 1983, C++ :langage compilé ; et l'objective-C.
- 1987, PERL : langage interprété pour traiter l'information de type textuel.
- 1990, le HTML : Hypertext Markup Language se traduit littéralement en langage de balisage d'hypertexte.
- 1991, Python langage orienté objet et multi-plateformes.
- 1994, php : langage de programmation libre, principalement utilisé pour produire des pages Web dynamiques via un serveur HTTP, mais pouvant également fonctionner comme n'importe quel langage interprété de façon locale. C'était à l'origine une bibliothèque logicielle en C.
- 1995, Java et le Javascript. Deux langages distincts! Java est multi-plateformes. JavaScript s'exécute sur les pages web, côté client mais aussi côté serveur depuis quelques temps.
- 2000, C# , langage orienté objet distribué par Microsoft. Il est dérivé du C++ et très proche du Java dont il reprend la syntaxe générale ainsi que les concepts.
- 2003, Scala.
- 2009, Go , langage compilé et développé par google.
- 2011, ruby langage orienté objet et multi-paradigme.
- 2011, Kotlin ,est un langage de programmation orienté objet et fonctionnel, en 2017 Kotlin devient le second langage de programmation officiellement pris en charge par Android après Java.

Tous ces langages sont différents, mais ils ont aussi des points communs, on peut même dire qu'ils ont plus de points communs que de différences.

Le langage C a été créé par Dennis Ritchie (1941-2011) et Ken Thompson (1943-) en 1972. Le langage C est une évolution du langage B (langage B a été créé par Ken Thompson à la fin des années 60). Le langage C est encore très utilisé aujourd'hui (dans le top 10 des langages de programmation les plus utilisés), par exemple, le noyau du système d'exploitation Linux est écrit en C. Tout informaticien qui se respecte doit avoir, un jour ou l'autre (au moins pendant ses études), écrit des programmes en C.

Compilateur et interpréteur

Le C est un **langage compilé**, c'est-à-dire qu'un programme appelé "compilateur" transforme le code source (le code écrit par le programmeur) en langage machine. Cette opération, appelée "compilation", doit être effectuée à chaque fois que le programmeur modifie le code source, cette phase de compilation peut prendre des dizaines de minutes pour de très gros programmes.

Il existe une autre méthode pour passer du code source au langage machine : **l'interprétation**. En simplifiant à l'extrême, l'interpréteur assure une traduction "à la volée" des instructions. (une ligne est traduite en langage machine puis immédiatement exécutée), alors que dans le cas de la compilation l'ensemble du code source est transformé en langage machine avant le début de l'exécution du programme. Les langages compilés (comme le langage C) sont réputés plus rapides à l'exécution que les langages interprétés.

Il existe une troisième voie qui a le vent en poupe : le code source est compilé en **pseudocode** (appelé **bytecode**) qui n'est pas encore du langage machine, mais s'en rapproche par rapport au code source de départ. Ce bytecode est ensuite interprété (l'interprétation est beaucoup plus rapide que pour des langages 100% interprétés). Python utilise cette technique.

Le java est compilé en bytecode et ce bytecode est ensuite interprété grâce à une machine virtuelle (JVM) dépendante du système d'exploitation. Lors des procédures de compilation, un certains nombres d'erreurs sont détectées et signalés. Le programme doit être corrigé.

Sources: https://fr.wikipedia.org/wiki/Histoire_des_langages_de_programmation
https://pixees.fr/informatiquelycee/n_site/nsi_prem_langageC.html

2- Un exemple : Le mélange de Fischer-Yates

2-1. Principe :

L' algorithme que nous allons utiliser est l'algorithme de mélange de Fischer-Yates, aussi appelé algorithme de Knuth. Cet algorithme prend en entrée un tableau (on considérera, au besoin, qu'il contient des entiers) et mélange l'ordre de ses éléments. Il ne renvoie rien, mais le tableau a été modifié (on parle d'algorithme en place).

Une implémentation possible en **Python** est la fonction suivante :

```
1 def shuffle(tab):
2     for i in range(1, len(tab)):
3         j = random.randint(0, i)
4         if j < i:
5             temp = tab[i]
6             tab[i] = tab[j]
7             tab[j] = temp
```

Questions :

Déterminer le rôle de chaque ligne du programme :

……

……

……

……

Quel type de boucle est utilisé ?

……

……

Quel test est employé ?

……

……

Quel type de donnée est traitée ?

.....

.....

Quelle fonction prédéfinie est appelée ? Quel est son rôle ?

.....

.....

Écrire l'algorithme (ou pseudo-algorithme) de mélange de Fischer-Yates ?

.....

.....

.....

.....

.....

.....

2-1. Comparaison de langages :

Tout d'abord lire le programme dans les différents langages suivants pour repérer leurs structures.

```
1 subroutine shuffle (tab, n)
2       integer n, tab(*)
3       integer i, j, temp
4       real r
5
6       do 10 i = 2, n
7               call random_number(r)
8               j = int(r * i) + 1
9               if (j < i) then
10                      temp = tab(j)
11                      tab(j) = tab(i)
12                      tab(i) = temp
13              endif
14 10 continue
15      return
16 end
```

Langage Fortran (1954)

```
100 FOR I = 1 TO LONGUEUR
110 J = INT(RND(1) * I + 1)
120 IF I = J THEN GOTO 160
130 TMP = TAB(I)
140 TAB(I) = TAB(J)
150 TAB(J) = TMP
160 NEXT I
170 END
```

Langage Basic (1964)

```
1 void shuffle(int tab[], int n) {
2       int i;
3       for (i = 1; i < n; ++i) {
4               int j = random(i + 1);
5               if (j < i) {
6                       int tmp = tab[i];
7                       tab[i] = tab[j];
8                       tab[j] = tmp;
9               }
10      }
11 }
```

Langage C (1972)

```php
<?php
function MyShuffle($tab) {
    for($i = 0; $i < sizeof($tab); ++$i) {
        $r = rand(0, $i);
        $tmp = $tab[$i];
        $tab[$i] = $tab[$r];
        $tab[$r] = $tmp;
    }
};
?>
```

Langage php (1994)

```javascript
function shuffle(tab) {
    for (var i = 1; i < tab.length; i++) {
        var j = Math.floor((i + 1) * Math.random());
        if (j < i) {
            var temp = tab[rand];
            tab[rand] = tab[i];
            tab[i] = temp;
        }
    }
}
```

Javascript (1995)

```java
public static void shuffle (int[] tab) {
  for (int i = 1; i < tab.length; i++) {
    int j = gen.nextInt(i + 1);
    if (j < i) {
      int temp = tab[i];
      tab[i] = tab[j];
      tab[j] = temp;
    }
  }
}
```

Java (1995)

```go
func shuffle(tab []int) {
  for i := 1; i < len(tab); i++ {
    j := rand.Intn(i + 1)
    if j < i {
      tmp := tab[i]
      tab[i] := tab[j]
      tab[j] := tmp
    }
  }
}
```

Langage Go (2009)

```kotlin
fun shuffle(tab: Array<Int>)
{
  for (i in 1 until tab.size) {
  val j = (0..i).random()
  if (j < i) {
      val tmp = tab[i]
      tab[i] = tab[j]
      tab[j] = tmp
    }
  }
}
```

Langage Kotlin (2011)

Questions et classements :

Les similitudes :

- Quels éléments de structure retrouve-t-on dans tous les programmes et ce quel que soit le langage ?

.....

.....

.....

.....

Classification logique :

- Comment les blocs d'instructions à l'intérieur d'une boucle ou d'un test sont-ils délimités ?

……

……

……

……

- Comment sont structurées les boucles for? les tests ?

……

……

……

- Est-il nécessaire de déclarer les variables locales ? Si oui, où ?

……

……

……

……

- Les types de données sont-ils spécifiés explicitement ? Doit-on indiquer que telle variable correspond à un entier ou un tableau ?

……

……

……

……

Classification chronologique :

Quelles sont les évolutions générales observées au cours de l'évolution chronologique des différents langages ?

……

……

……

……

……

……

Source : https://cache.media.eduscol.education.fr/file/NSI/77/2/RA_Lycee_G_NSI_lang_diversite_1170772.pdf