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.
- 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 :

Ligne 1 : Définition de la fonction 'shuffle' (mélanger) qui prend en paramètre d'entrée un tableau, noté tab dans la suite de la fonction.

Ligne 2 : Boucle pour qui parcours les indices i (entier) de 1 à la longueur du tableau (elle commence donc à la deuxième case du tableau.

Ligne 3 : L'entier j est affecté d'une valeur aléatoire comprise entre 0 et i (raison pour laquelle il n'est pas nécessaire de commencer la boucle pour par l'indice 0).

Ligne 4 : Condition booléenne : Si j est inférieur à i alors (faire)

Ligne 5 et suivantes : La condition est remplie (le booléen est vrai) alors faire :

la valeur i+1 du tableau (tab) est copiée dans la variable temp (pour temporaire)

la valeur j+1 du tableau (tab) est copiée dans le tableau (tab) à position i+1

la valeur temp est copiée dans le tableau (tab) à position j+1

Quel type de boucle est utilisé ?

Une boucle pour : (if)

Quel test est employé ?

Le test employé est une conditionnelle Si (booléen) d'un test de valeur : Si j est inférieur à i alors vrai donc faire instruction suivante ...

Quel type de donnée est traitée ?

La donnée principale traité est un tableau de valeurs.

Quelle fonction prédéfinie est appelée ? Quel est sont rôle ?

La fonction random.randint(0, i) permet de choisir aléatoirement un entier compris entre 0 et i.

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

Soit la fonction 'shuffle' (mélanger) qui prend en paramètre d'entrée un tableau de valeurs

Pour i (entier) allant de 2 à la longueur du tableau faire :

j est affecté d'une valeur entière aléatoire comprise entre 1 et i

Si j est inférieure à i alors :

la i+1ème valeur du tableau est copiée dans la variable temp

la j+1ème valeur du tableau est copiée à la i +1ème position du tableau

la variable temp est copiée à la j +1ème position du tableau

fin Si

fin Pour

Remarques :

- Dans cette version de l'algorithme la première valeur du tableau est bien ici stockée dans la 'case' d'indice 1, d'où un décalage des indices par rapport au programme en python. (tout est question de normalisation et de choix)

- La fin de la boucle Pour et de la condition Si sont bien indiquées par un : fin … , l'indentation ne sert ici qu'a faciliter la lecture de l'algorithme et n'a pas de rôle logique.

2-1. Comparaison de langages :

Tout d'abord lire le programme dans les différents langages suivants pour repérer leur structure.

```
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)

```
1 <?php
2 function MyShuffle($tab) {
3       for($i = 0; $i < sizeof($tab); ++$i) {
4             $r = rand(0, $i);
5             $tmp = $tab[$i];
6             $tab[$i] = $tab[$r];
7             $tab[$r] = $tmp;
8       }
9 };
10 ?>
```

Langage php (1994)

```
1 100 FOR I = 1 TO LONGUEUR
2 110 J = INT(RND(1) * I + 1)
3 120 IF I = J THEN GOTO 160
4 130 TMP = TAB(I)
5 140 TAB(I) = TAB(J)
6 150 TAB(J) = TMP
7 160 NEXT I
8 170 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)

```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 ton dans tous les programmes et ce quel que soit le langage ?

Le nom de la fonction se retrouve dans presque tous les langages (sauf en Basic ou c'est le numéro de ligne qui permet de réutiliser le code…)

Dans tous les langages on retrouve les éléments clefs de l'algorithme : la boucle Pour, la condition Si et les variables locale : i, j et temp

Classification logique :

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

- Pour les premiers langages (Fortran et Basic) la fin d'une boucle ou d'un test sont : soit explicitement écrit endif ou NEXT I , soit ont demande au programme de retourner à une ligne du programme, en basic : THEN GOTO…

- Pour les langages plus récent c'est l'utilisation d'accolades ouvrantes { et fermantes } qui délimite le début et la fin de l'instruction. On retrouve ici la notion de balises déjà rencontré en html.

- Comment sont structurées les boucles for? les tests ?

- Pour les premiers langages (Fortran et Basic …) la structure est une instruction par ligne sans délimitateur ni indentation (le numéro de ligne en Basic permet de suivre les instructions… très '*pratique*' si on doit rajouter ou supprimer une ligne de code…)

- Pour les langages suivant (à partir du C) l'utilisation du point virgule ; à la fin de chaque ligne permet de suivre et vérifier la syntaxe du programme.

- Les langages les plus récents utilisent de plus en plus la simple indentation des lignes pour simplifier l'écriture des programmes. (la colorisation syntaxique, disponible sur notepad++ et les éditeurs spécifiques, à aussi permit de faciliter l'écriture des programmes)

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

- La déclaration des variables locale en Fortran et Basic se fait au début du programme.

- Elle n'est plus forcément nécessaire pour les langages postérieurs au Basic.

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

- Pour les premiers langages (Fortran et Basic ...) les types de données sont entièrement spécifiés (au début du programme).

- Pour le C et le Java on doit spécifier la nature de la variable lors de sa première utilisation ou affectation.

- Les langages les plus récents ne nécessitent pas de déclaration particulière. (l'éditeur interprète automatiquement la nature des variables)

Classification chronologique :

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

- La notion de verbosité : Les premiers langages de programmations étaient assez verbeux : beaucoup de lignes de codes sont nécessaires (pour la structure, préciser les variables et leurs natures, ...) ; Avec les progrès des analyseurs de code un grand nombre de ces informations n'a plus à être écrite explicitement. De plus, la syntaxe des langages moderne est aussi allégée et simplifiée.

- Les variables locales : Avec le temps, la gestion des variables locale se simplifie considérablement. Au début, elles doivent toutes être déclarées au début de la fonction avec leur type. Ensuite, en fonction des langages: elles peuvent être déclarées « en cours de route » (voir pas du tout, et la spécification de leur type n'est plus systématiquement nécessaire.

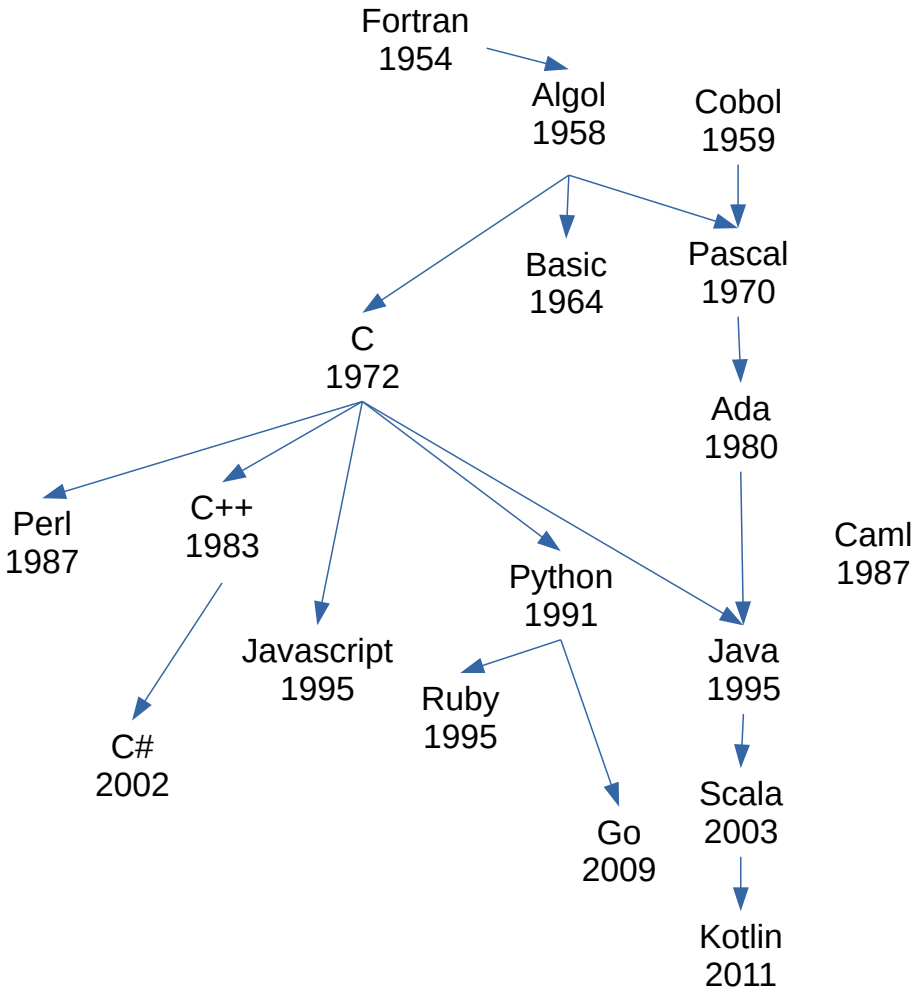

Figure : Généalogie (non exhaustive) des langages de programmation

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