Chapitre 1

Introduction aux langages

1.1 Langages informatiques

On peut distinguer deux types de langages informatiques :

— les langages de bas niveau qui ne fournissent que peu d'abstraction et se concentrent sur les aspects machines (gestion de la mémoire, instructions machines, etc) ;

— les langages de haut niveau qui fournissent plus d'abstraction, sont proches des langages naturels comme l'anglais ou les mathématiques et sont généralement indépendants de la machine.

L'assembleur fut le premier vrai langage de programmation mais c'est aussi un des langages les plus proches du langage machine car il permet de traiter directement avec le processeur. C'est un langage de bas niveau et par conséquent relativement compliqué à comprendre par un humain. Depuis le tout premier langage haut niveau A-0 conçu par Grace Hopper au début des années 50, de nombreux langages sont apparus et on peut les regrouper en trois catégories :

— les langages de programmation,

— les langages de requêtage,

— les langages de description.

1.1.1 Langages de programmations

Ils ont pour but de faciliter la tâche de programmation des applications. Ils sont beaucoup moins lourds à manipuler que l'assembleur. Ces langages synthétiques sont totalement symboliques : le programmeur n'a qu'à attribuer des noms de variables aux données qu'il veut manipuler sans se préoccuper des adresses-mémoire où elles seront implantées. Ils font accomplir par la machine des actions de calcul, d'entrée de données et de sortie de résultats, de prise de décisions et de répétition de séquences, en somme tout ce qu'on fait en algorithmique. Ces langages sont très nombreux : FORTRAN, COBOL, BASIC, PASCAL, C, C#, C++, Java, Python ...
Parmi ces langages, on distingue les langages compilés et les langages interprétés. La compilation est une étape pendant laquelle le code est transformé en un fichier exécutable par la machine. À chaque modification, il faut compiler le code pour pouvoir, dans un deuxième temps, l'exécuter.

Les langages interprétés sont quant à eux traduits en exécutable au fur et à mesure de l'exécution.

Dans tous les langages de programmation, on retrouve les **instructions élémentaires** suivantes :

— la gestion des variables,

— l'instruction conditionnelle,

— la boucle non bornée,

— la boucle bornée.

On appelle **séquence** toute une suite d'instructions.

Exemple : le programme ci-dessous est une fonction Python permettant de calculer la moyenne d'une liste de nombres.

```
def moyenne(liste : list) -> float :

    assert type(liste) == list, "Erreur : l'argument n'est pas une liste."
    assert len(liste) > 0, "Erreur : la liste prise en entrée est vide."

     somme = 0

     for valeur in liste :
          somme += valeur

     return somme / len(liste)
```

1.1.2 Langages de requêtage

Ces langages qualifient le plus souvent les langages propres aux bases de données, ils sont représentés notamment par le SQL (Structured Query Language) : il permet tout simplement de gérer une base de données par exemple l'interroger, y insérer des données ou en supprimer d'autres, lui demander de faire ressortir des données selon des critères fixés.

Exemple : le programme ci-dessous est un code SQL permettant d'effectuer une recherche dans une base de données en croisant les informations de plusieurs des tables de données qu'elle contient.

```
SELECT moves.identifier, types.identifier, moves.generation_id, moves.power
FROM moves
INNER JOIN types
ON moves.type_id = types.id
WHERE moves.generation_id = 1
ORDER BY moves.power DESC
LIMIT 15 ;
```

1.1.3 Langages de description

Dans ces langages, on décrit ce qu'on veut obtenir. Ils reposent sur ce qu'on appelle des balises ou tags, ces derniers sont des étiquettes avec lesquelles on peut étiqueter des données (mots, texte etc.) pour produire un effet chez eux tant en sens (leurs donner du sens : ceci est un paragraphe, ceci est un titre, citation etc.) qu'en rendu visuel (italique, gras, couleur du texte etc.), on peut étiqueter des données en les encadrant par ces balises. Parmi ce type de langages, il y a le HTML (HyperText Markup Language) accompagné du CSS, le XML et ses dérivés.

Exemple : le programme ci-dessous est le début d'un code HTML permettant l'édition d'une page Web.

```
<!DOCTYPE html>
<html lang="fr"> <!--commentaires -->

<head>
     <title> Mouette </title>
     <meta charset="utf-8"/>
     <link rel="stylesheet" href="style.css" type="text/css"/>
</head>

<body>
     <header>
          <h1>Mouette 2 <br/>
          le retour</h1>
```

1.2 Fonctions

Dès qu'un programme devient un peu long et/ou que des blocs d'instructions ont tendance à se répéter, on a intérêt à utiliser les fonctions. Une **fonction** est un sous-programme plus ou moins indépendant du programme principal mais que l'on peut développer de manière autonome.
Tout comme une fonction mathématique, les fonctions de programmation prennent des arguments sous la forme de variables, sur lesquelles elles exercent une certaine tâche et produisent un résultat.
Un **prototype** est la signature d'une fonction. Il indique le nom de la fonction, le type de la valeur de retour et le type des arguments. Les variables utilisées au sein d'une fonction ont une portée limitée à la fonction.
En programmation, une fonction, au sens strict, doit renvoyer une valeur lorsqu'elle se termine. Sans valeur de retour, on appelle cela une **procédure**.

1.3 Bonnes pratiques

1.3.1 Phase de développement

Quelque soit le langage de programmation utilisé, un algorithme bien codé doit avoir les propriétés suivantes.

▷ **Avoir une organisation logique et évidente**

Quelque soit le chemin emprunté pour résoudre la problématique posée, chaque étape logique dans l'avancement du programme doit être commentée. Il ne faut également pas hésiter à laisser quelques lignes entre chaque étape pour aérer et faciliter la lecture du code.

▷ **Être facile à lire, par soi-même mais aussi par les autres**

Indenter un programme, c'est-à-dire insérer des espaces blancs en début de ligne, est un moyen de visualiser le niveau d'imbrication auquel une instruction se trouve. Il est aisé de lire un code où les blocs d'instructions du même niveau sont précédés du même nombre d'espaces. Alors que dans de nombreux langages l'indentation n'est qu'une aide visuelle, en Python, elle est primordiale. Elle change la signification du programme.

▷ **Être explicite**

Les noms des variables et des fonctions contribuent à la lisibilité de votre code. Ils doivent être explicites et de préférence en français ou en anglais si vous travaillez avec des personnes de nationalités différentes.

1.3.2 Analyse

« Tester un programme peut démontrer la présence d'un bug, jamais son absence. » (Dijkstra)

Analyse dynamique

Un programme est correct s'il effectue sans se tromper la tâche qui lui est confiée et ce dans tous les cas possibles. Programmer sans erreur est une tâche difficile en raison de la taille des logiciels, du nombre de personnes impliquées dans leur confection et de leur historique.
La méthode la plus répandue est l'analyse dynamique . Elle consiste à exécuter le code ou à le simuler en vue de faire apparaitre d'éventuels bugs. Il s'agit de comparer le résultat d'un programme avec le résultat attendu. Pour établir un plan de test, il faut énumérer les cas à tester et établir un test par cas. Quand le progamme est constitué de plusieurs modules, chacun doit être testé indépendamment avant de tester la globalité dans une série de tests dits d'intégration.
Comme on ne peut que tester un nombre fini de valeurs, cette méthode n'est en général pas exhaustive.

Autres méthodes

Les méthodes formelles permettent d'obtenir une très forte assurance de l'absence de bug dans les logiciels. Utilisées dans le développement des logiciels les plus critiques, elles permettent de raisonner rigoureusement, à l'aide de logique mathématique, sur les programmes informatiques afin de démontrer leur validité.