Types de langages

Le microprocesseur d'un ordinateur n'effectue que des opérations simples sur des séquences de 0 et des 1, appelées **binaire** [1]. Comment fait-il pour lire des données que l'on souhaite traiter (les textes, les images, les sons, les nombres, etc.) ou exécuter des logiciels, des programmes par exemple écrits en Python, etc. ?

Langages de bas niveau

Le langage natif du microprocesseur est appelé langage machine, c'est-à-dire le seul qu'il puisse traiter. Il est composé d'instructions et de données codées en binaire.

Cours

Langage machine : longue suite de 1 et de 0 (les "**bits**"[5]) souvent traités par groupes de 8 (les « octets »), 16, 32, ou même 64 en fonction de l'architecture du microprocesseur.

Le langage assembleur associe des noms aux mots binaires : placer un nombre en mémoire, faire une addition, etc… La transformation du code assembleur en langage machine est accomplie par un programme nommé aussi assembleur. L'opération inverse (retrouver l'assembleur équivalent à un morceau de code machine) est le désassemblage.

Cours

Assembleur[6] : langage qui représente le langage machine sous une forme lisible par un humain. Les combinaisons de bits du langage machine sont représentées par des symboles faciles à retenir.

Par exemple, un processeur de la famille x86[2] reconnaît une instruction en langage machine du type :

```
10110000 01100001
```

Cette même instruction s'écrit en langage assembleur d'une façon plus facile à comprendre pour le programmeur (`10110000` s'écrit `movb %al` et `01100001` s'écrit `$0x61`) :

```
movb $0x61,%al
```

ce qui signifie : « écrire le nombre 97 (la valeur est donnée en hexadécimal : $61_{16} = 97_{10}$) dans le registre `AL` ».

Cours

Le **langage machine** et l'**assembleur** sont des langages dit de « **bas niveau** » constitués d'instructions très élémentaires. Chaque processeur possède son **propre langage machine et assembleur**. Un programme en langage machine ou assembleur ne peut s'exécuter que sur la machine pour laquelle il a été écrit.

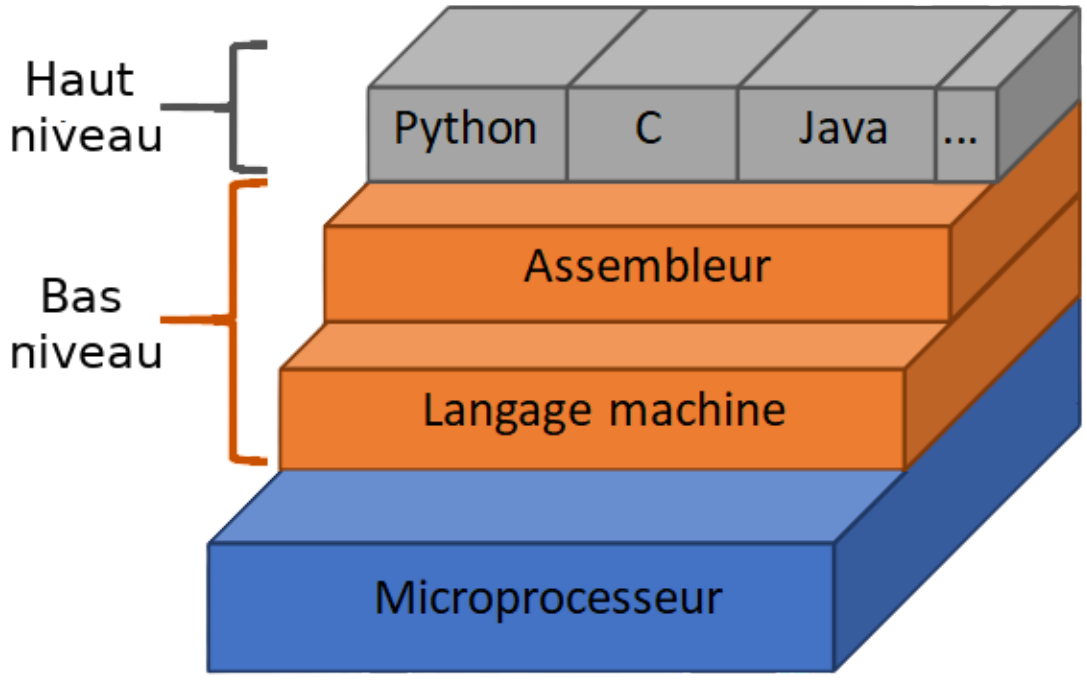
Langages de haut niveau

Même si coder en assembleur et plus rapide qu'en binaire, cela reste une étape longue et fastidieuse. De plus que chaque processeur possède un jeu d'instructions différent des autres, un code en assembleur qui fonctionne sur un processeur Pentium ne marchera pas forcément sur un AMD ou sur un processeur ARM, on dit que le programme n'est pas portable.

Les langages de haut niveau ont permis de palier à ces deux inconvénients : ils permettent une programmation rapide et plus simple que l'assembleur et ils sont portables. Le même programme écrit en Python ou en C fonctionnera sur différents ordinateurs.

Cours

Les langages de « **haut niveau** » (Python, C, Java…) comportent des instructions plus abstraites ou plus « puissantes qui seront traduites en un grand nombre d'instructions machine élémentaires.



Langage de « bas niveau »	Langage de « haut niveau »
Directement exécuté par le processeur	Doit être « traduit » en langage machine pour être exécuté (complexe et long)

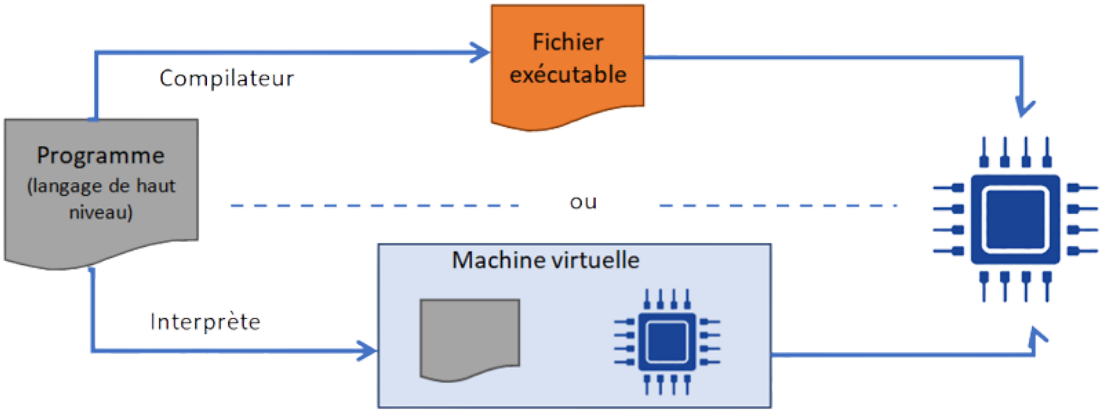
Langage de « bas niveau »	Langage de « haut niveau »
Plus difficile à écrire, maintenir (modifier), debugger	Plus facile et plus rapide à écrire, maintenir, et debugger
Ne fonctionne que sur un seul type de machine (pour qu'une autre l'accepte, il faut le réécrire entièrement)	Portable (fonctionne sans trop de modifications sur des machines ou des systèmes d'exploitation différents)

Langages interprétés ou compilés

Le seul langage compris par le processeur est le langage machine il faut donc utiliser une sorte de traducteur entre le code source écrit dans un langage de programmation de haut niveau et le langage machine compris par le processeur. Il en existe de deux types :

– un **compilateur** est un programme qui traduit un code écrit dans un langage de haut niveau en un fichier exécutable[3] en assembleur compatible avec le processeur.

– un **interprèteur** est un programme qui simule une machine virtuelle dont le langage machine serait le langage de programmation lui-même. Le code va être lu, analysé et exécuté instruction par instruction par l'interprète.



Cours

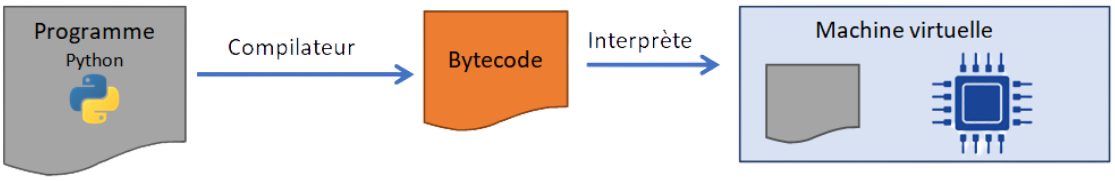
Un programme écrit dans un langage de haut niveau peut être traduit en langage machine exécutable par la machine de deux façons : par **interprétation** ou **compilation**.

En général, un programme compilé est nettement plus rapide qu'un programme interprété puisque toute la phase d'analyse et de vérification du code source a déjà été faite lors de la phase de compilation.

Python est entre un langage interprété et un langage compilé.

Cours

Un script Python est d'abord **compilé** en un **langage intermédiaire**, appelé ***bytecode***, qui est lui-même ensuite **interprété** par une machine virtuelle.



On peut observer le Bytecode d'un petit programme en utilisant le module dis de python :

`import dis`	
`dis.dis('x = 25; x = x + 1')`	
`>>>`	
`Remote Interpreter Reinitialized`	
`1 0 LOAD_CONST 0 (25)`	Le nombre 25 est copié dans un registre
`2 STORE_NAME 0 (x)`	Le registre est copié à l'adresse mémoire de x
`4 LOAD_NAME 0 (x)`	La valeur de x est copiée dans un registre
`6 LOAD_CONST 1 (1)`	Le nombre 1 est copié dans un registre
`8 BINARY_ADD`	Les deux sont additionnés
`10 STORE_NAME 0 (x)` <	Le résultat est copié à l'adresse mémoire de x
`12 LOAD_CONST 2 (None)`	
`14 RETURN_VALUE`	

Le bytecode est portable, il peut s'exécuter sur différentes machines, il suffit de disposer pour chacune d'elles d'un interpréteur adapté.

L'interprèteur Python le plus utilisé est CPython ; il est écrit en C[4].

Comparaison entre interprétation, compilation et code intermédiaire

Interprétation	Compilation	Intermédaire
Un interpréteur analyse au fur et à mesure chaque ligne du programme source et la traduit	Un compilateur lit une seule fois toutes les lignes du programme source et produit un programme objet	Le programme source est d'abord compilé pour produire un code intermédiaire, similaire à un

Interprétation	Compilation	Intermédaire
en langage machine qui est ensuite directement exécutée. Aucun programme objet n'est généré.	(ou code objet) qui peut être exécuté indépendamment du compilateur et être conservé dans un fichier exécutable.	langage machine, que l'on appelle bytecode. Le bytecode est transmis à un interpréteur pour l'exécution dans une machine virtuelle.
Idéal en phases de développement pour tester les modifications d'un programme sans compilation qui demande du temps	Préférable lorsqu'un programme doit s'exécuter rapidement, un programme compilé est plus rapide puisque l'ordinateur n'a plus à traduire chaque instruction en binaire.	L'interpréteur permet de tester rapidement un programme tout en gardant la rapidité d'un code compilé, même si l'interprétation du bytecode compilé n'est pas aussi rapide que celle d'un véritable code binaire.
Basic, PHP, Javascript	C, C++, Cobol, Fortran, Pascal	Python, Java, C#

1. En pratique le microprocesseur utilise des signaux électriques qui prennent deux états (un potentiel électrique maximum ou minimum) sur une logique logique du type « ouvert ou fermé », qu'on assimile à des nombres ne prenant que les deux valeurs 0 et 1. ↩
2. La famille de processeurs x86 désigne les processeurs fonctionnant en 32 bits, la famille x64 ceux en 64 bits. ↩
3. En pratique le compilateur créé le code assembleur qui est ensuite assemblé en binaire pour créer un fichier exécutable. ↩
4. Mais d'autres existent (Jython, écrit en Java par exemple). ↩
5. ⚠️ En anglais octet se dit « *byte* », ne pas confondre avec « bit ». ↩
6. Le Langage assembleur est étudié dans le chapitre Architectures matérielles et Systèmes d'exploitations. ↩