S si	**Programmation**	***Cours et TD***

1. Introduction

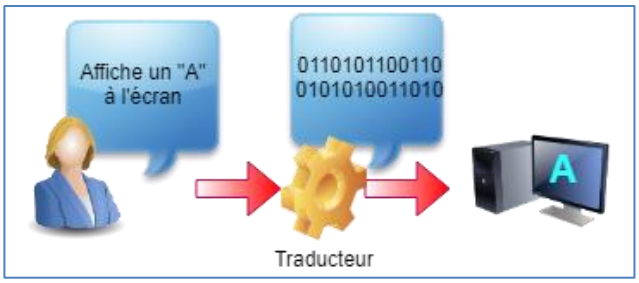
Nous sommes entourés au quotidien de systèmes programmés. Ils sont devenus incontournables dans de nombreux domaines.
L'objectif de ce cours est de vous aider à comprendre les principes de base de la programmation.

Programmation de sites Internet	Programmation de jeux video	Programmation d'applications pour Smartphone	Analyse de données	Programmation de modules industriels
				

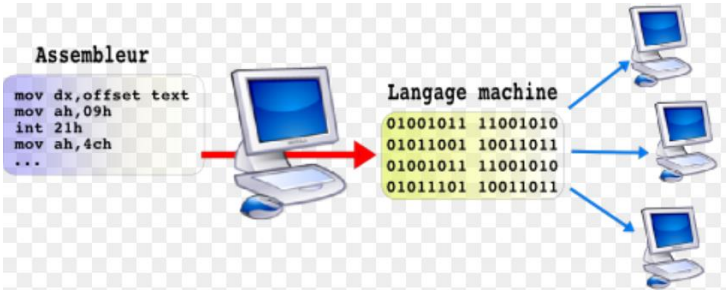
2. Principe de communication

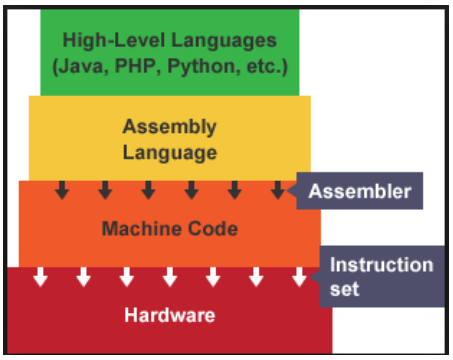
Un processeur d'ordinateur ne comprend que le binaire (suites de "0" et de "1"). Pour pouvoir demander à l'ordinateur d'exécuter des actions sans avoir à les écrire sous forme de "0" et de "1" (trop laborieux) il faut utiliser un "traducteur".

Il existe de nombreux « traducteur » et chaque « traducteur » possède son propre langage. Donc, vous allez écrire non pas dans le langage de l'ordinateur mais dans le langage du « traducteur ». Le traducteur, qui comprend ce que vous lui dite, convertira tout cela en 0 et 1 parfaitement compréhensible par l'ordinateur.



Un langage de **haut niveau** (exemple: Python, PHP, Java, ...) est proche du langage des êtres humains, il est par conséquent plus facile à utiliser. Cependant, cette facilité limite les possibilités d'interagir avec la machine selon ce que le langage met à disposition.





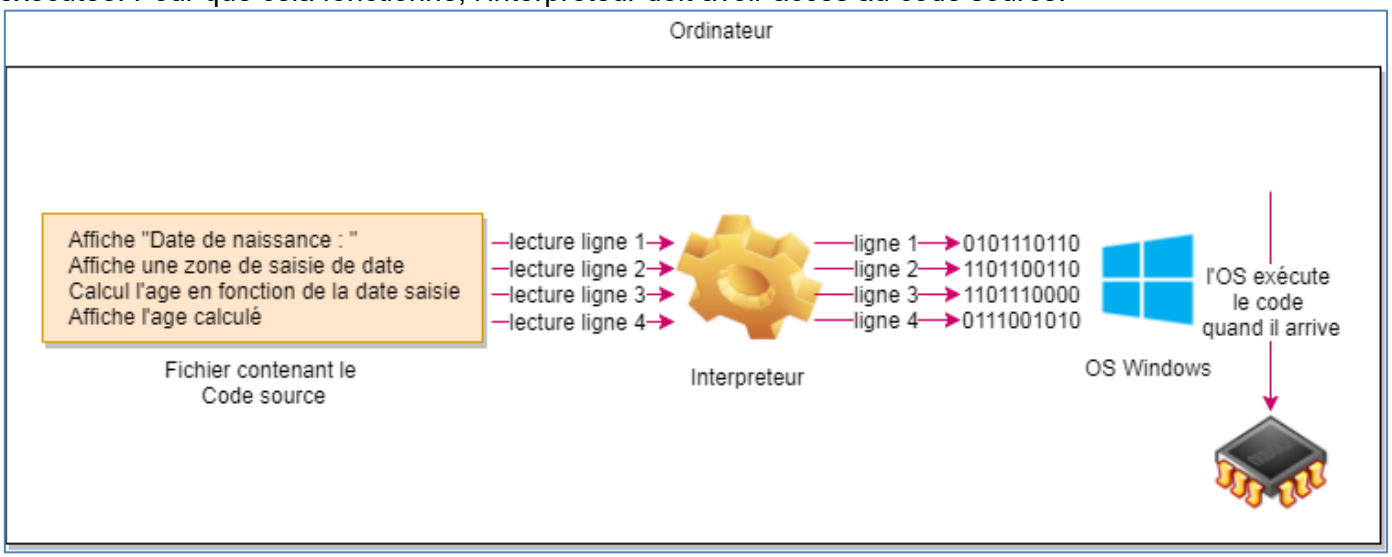
Par opposition, un langage de **bas niveau** est proche langage compris par la machine. Il est donc plus difficile à apprendre et à utiliser mais il permet plus de possibilité d'interaction avec le hardware de la machine.

3. Langages interprétés, compilés, mixtes

Un "traducteur" comme nous l'avons défini précédemment peut convertir le langage (code source) de deux façons différentes: il peut soit interpréter soit compiler le langage.

3.1. Langage interprété

Un langage interprété est un langage dans lequel chaque ligne d'instruction est lue et traduite pour être exécutée. Pour que cela fonctionne, l'interpréteur doit avoir accès au code source.



OS (Operating System): c'est le système d'exploitation installé sur la machine (exemples: Windows, Linux, Android, MacOS, ...)

Sur le schéma que peut-on voir :

- un fichier contenant le code source,
- un interpréteur qui lit chaque ligne du code source et la transforme en code binaire lisible par l'OS qui lui se chargera de l'envoyer au processeur.

On comprend donc que :

- les lignes sont lues et traduites au fur et à mesure des besoins,
- le code source doit être présent dans l'ordinateur,
- l'interpréteur doit lui aussi être présent dans l'ordinateur.

Exemples de langages interprétés: Matlab, PHP.

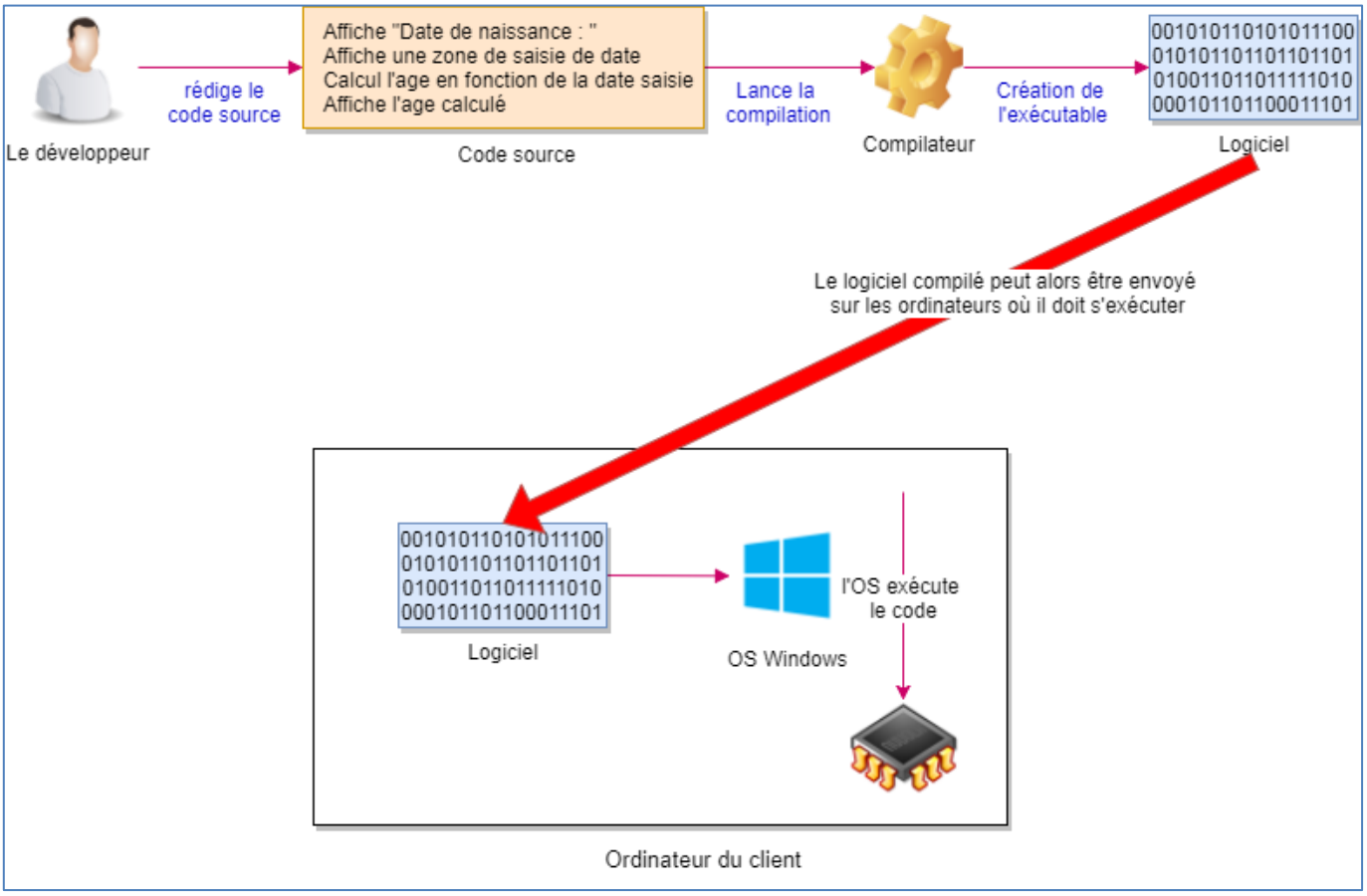
3.2. Langage compilé

Avec un langage compilé, c'est différent. Le développeur du code source va effectuer une opération qui s'appelle la compilation. Cette opération réalisée à partir d'un logiciel spécial appelé « compilateur » va convertir l'ensemble du code source en code machine. Sous Windows, cela donne un fichier avec l'extension « .EXE » (exe pour exécutable).

Pour exécuter le programme sur la machine, il n'y a besoin que de ce fichier exécutable (le code source n'est plus nécessaire).

A chaque modification du programme par le développeur, il faudra recompiler le programme.

Exemple de langage compilé: C++



3.3. Différence entre interprété et compilé

L'avantage du langage interprété c'est sa portabilité, c'est à dire sa capacité à fonctionner sur différents types d'OS. En effet, il suffit d'avoir l'interpréteur sur l'ordinateur sur lequel on veut faire tourner le programme et on peut en théorie le faire fonctionner.

Par contre, l'inconvénient majeur du langage interprété c'est sa rapidité. En effet, l'interpréteur traduit le code source ligne par ligne ce qui prend un peu de temps. Un langage interprété est toujours un peu plus lent qu'un langage compilé.

Et enfin, en envoyant le code source sur l'ordinateur sur lequel il va être exécuté, le programme est moins facilement protégé contre la copie et le code source peut-être modifié.

Un langage compilé est généralement plus « proche »de l'OS. Il est moins portable. La plupart des langages compilés s'adressent à un OS spécifique. Mais c'est aussi ce qui les rend plus performants en termes de rapidité et de fonctionnalités.

3.4. Langages mixtes

Il existe des langages « mixtes » (entre interprétation et compilation). Ces langages sont compilés en bytecode (le bytecode est du code compilé mais pas directement exécutable sur l'ordinateur) et ce bytecode sera interprété sur l'ordinateur distant. Cela permet de pallier au problème de lenteur et de protection du langage interprété tout en ayant une bonne portabilité entre les différents OS.

Exemple de langage mixte: JAVA

4. Les points communs à tous les langages de programmation

4.1. Constantes

Une constante est une donnée qui ne change pas au cours du programme.
On utilise des constante pour augmenter la lisibilité des programmes et pour en facilité l'écriture.
Exemple: PI = 3.14

4.2. Variables

Une variable est une donnée qui va être amenée à changer au cours du programme.
Il est nécessaire de définir de quel type sont les variables pour réserver un espace mémoire plus ou moins grand pour les stocker.
Présentation de quelques types de variables fréquemment utilisés :

int	« integers » les entiers relatifs, souvent codés sur 2 octets et alors compris entre « -32 768 ; ….-3 ; -2 ; -1 ; 0 ; 1 ; 2 ; ….32 767 »
unsignedint	« unsignedintegers » les entiers naturels qui lorsque codés sur 2 octets sont compris entre « 0 ; 1 ; 2 ; 3 ; …….. ; 65 534 ; 65 535 »
boolean	Variable logique, codée sur 1 bit, qui ne peut prendre que deux valeurs « true » et « false »
float	Nombres décimaux (nombres à virgule). Attention il faut utiliser un point (et non une virgule) avant les décimales
byte	Variable codée sur 8 bits (donc 1 octet) qui peut prendre les valeurs entières comprises entre 0 et 255

4.3. Algorithme

Un **algorithme** est la description de la méthode à utiliser pour répondre à un problème..
Des actions tel qu'une recette de cuisine, ouvrir une porte ou même saluer de la main peuvent être résolu grâce à un algorithme. En effet, les étapes nécessaire pour accomplir ces actions seront toujours les mêmes.
L'algorithmique exprime les instructions résolvant un problème donné indépendamment des particularités de tel ou tel langage.
Apprendre l'algorithmique, c'est apprendre à manier la structure logique d'un programme informatique

4.1. Instructions

Une instruction est un ordre donné à un ordinateur.

Les ordinateurs, quels qu'ils soient, ne sont fondamentalement capables de comprendre que quatre catégories d'ordres (en programmation, on n'emploiera pas le terme d'ordre, mais plutôt celui d'instructions). Ces quatre familles d'instructions sont :

- l'affectation de variables
- la lecture / écriture
- les tests
- les boucles
-

Un algorithme informatique se ramène donc toujours au bout du compte à la combinaison de ces quatre petites briques de base.

.

4.2. Pseudo-code

En programmation, le **pseudo-code**, également appelé LDA (pour Langage de Description d'Algorithmes) est une façon de décrire un algorithme en langage presque naturel, sans référence à un langage de programmation en particulier.

En pseudo-code, l'instruction d'affectation se note avec le signe ←

Ainsi :

```
Var_1 ← 24
```

Attribue la valeur 24 à la variable Var_1.

Langage C : syntaxe des tests

Écriture en pseudo code	Traduction en C

Si condition **alors**
instructions
Finsi
	if (condition) {
	instructions;
	}

Si condition **alors**
instructions1
Sinon
instructions2
Finsi
	if (condition) {
	instructions1;
	} else {
	instructions2;
	}

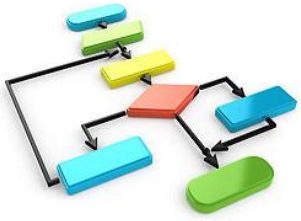
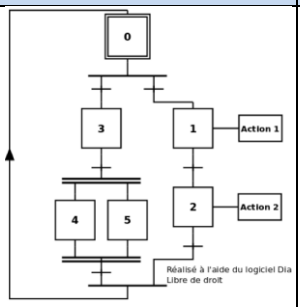
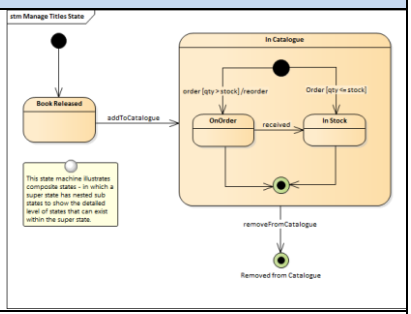
4.3. Structure de données

Une structure de données est une manière d'organiser un ensemble de données en mémoire. Les structures courantes sont les tableaux, les listes, les piles, les arbres, ...

5. Quel langage de programmation choisir ?

La plupart des langages ont des spécificités (simplicité d'écriture, portabilité, proximité avec le système, …). Ce qui fait, qu'il faut choisir le langage en fonction de ce que l'on veut faire.

Les programmes que vous allez utiliser cette année sont:

Pseudo-langage Algorithme	Programmation graphique Algorigramme / organigramme	GRAFCET	Diagramme états-transitions
Saisir un entier n compris entre 0 et 255 i=7 A="" tant que $i \geqslant 0$ faire si $n - 2^i \geqslant 0$ alors A = A + "1" $n = n - 2^i$ sinon A = A + "0" fin $i = i - 1$ fin Afficher A			
Il permet de décrire la structure logique d'un programme informatique	Il permet de décrire la structure logique d'un programme informatique de façon graphique	Ce langage graphique de programmation était initialement à destination des automates programmables industriels.	Ce diagramme permet de spécifier une séquence d'états qu'un objet traverse en réponse aux évènements qu'il subit.