

INTRODUCTION AUX ALGORITHMES

Qu'est ce que la programmation ?

Qu'est ce qu'un programme

Un programme est un ensemble d'instructions (ou d'étapes) destinées à être exécutées par un ordinateur.

Une instruction dicte à l'ordinateur l'action qu'il doit effectuer avant de passer à l'instruction suivante.

Qu'est-ce qu'un algorithme ?

“Un algorithme permet de décomposer un problème complexe en une suite d'opérations simples.”

3 principaux types d'actions d'un algorithme

1. Je prends une bouilloire
2. J'ajoute de l'eau
3. J'allume la bouilloire
4. Tant que l'eau ne bout pas
5. J'attends
6. Dès que l'eau bout, je verse l'eau dans le mug avec le thé
7. Tant que le thé n'est pas infusé
8. J'attends
9. Dès que le thé est infusé, j'enlève le sachet de thé
10. Si je préfère le thé sucré alors j'ajoute du sucre
11. Tant que le thé n'est pas assez sucré
12. J'ajoute du sucre
13. Je goûte
14. Si je préfère le lait au citron alors
15. J'ajoute du lait
16. Sinon
17. J'ajouter du citron

- **Actions simples**
- **Actions qui s'exécutent sous une condition**
- **Actions qui se répètent**

Comprendre les langages informatiques

Qu'est-ce qu'un langage informatique

Combien de langues connaissez-vous ? Sur Terre, il existe des milliers de langues qui nous permettent de communiquer.

Pour communiquer à un ordinateur des instructions qu'il devra exécuter par la suite, il vous faudra aussi utiliser une langue, mais une langue qu'il comprend. C'est ce qu'on appelle un **langage informatique**, ou du code. Et il en existe des milliers. (https://fr.wikipedia.org/wiki/Liste_de_langages_de_programmation)

Mais rassurez-vous, comme pour l'anglais, l'espagnol ou l'arabe, certaines langues sont plus populaires, et plus utilisées que d'autres. Chaque langage informatique est composé de sa propre syntaxe, ou règle, et sémantique ou signification.

Différences de syntaxe entre 2 langages humains

Prenons par exemple deux mots : " Bonjour ! ", et " ¡ Hola ! ". Ces deux mots sont différents, mais ils ont la même signification. On note qu'en espagnol, on utilisera deux points d'exclamation, à l'endroit et à l'envers, contrairement à un seul en français

C'est une règle de syntaxe différente entre ces deux langues. Il en va de même pour les langages informatiques.

Langues	Syntaxe
Français	Bonjour !
Espagnol	¡ Hola !

Différences de syntaxe entre 2 langues informatiques

Prenons deux langages informatiques : JavaScript et PHP.

Ces deux lignes de code font strictement la même chose. Ils affichent le texte "Bonjour à tous" sur une page.

Avez-vous remarqué la différence de syntaxe entre ces deux langages ? En JavaScript, on utilise `document.write`, parenthèses suivies du texte entre guillemets. Alors qu'en PHP, on utilise le mot `echo` suivi du texte "Bonjour à tous", entre guillemets.

Langages	Syntaxe
Javascript	<code>document.write("Bonjour à tous");</code>
PHP	<code>echo "Bonjour à tous";</code>

Qu'est-ce que le langage machine ?

Pourquoi existe-t-il autant de langages informatiques différents, surtout s'ils font la même chose ?

La réponse est très simple. Chaque langage a ses propres forces et faiblesses.

Certains sont idéaux pour des petits programmes, tandis que d'autres ont été conçus pour gérer des calculs mathématiques complexes.

Quel que soit le langage que vous choisissiez, ce langage sera en réalité décomposé en un autre langage que seuls les ordinateurs peuvent comprendre : **le langage machine**.

Le langage machine est la langue la plus proche de l'ordinateur, c'est du binaire : une succession de 0 et de 1, parfaitement incompréhensible pour l'humain. C'est pour cela que les langages de programmation ont été créés.



L'objectif final d'un langage informatique est donc d'être un intermédiaire entre notre langage à nous, le langage humain, et le langage machine, que parle l'ordinateur.

Plus un langage est bas niveau, plus il se rapproche du langage machine, ce qui le rend plus difficile à comprendre pour les développeurs.

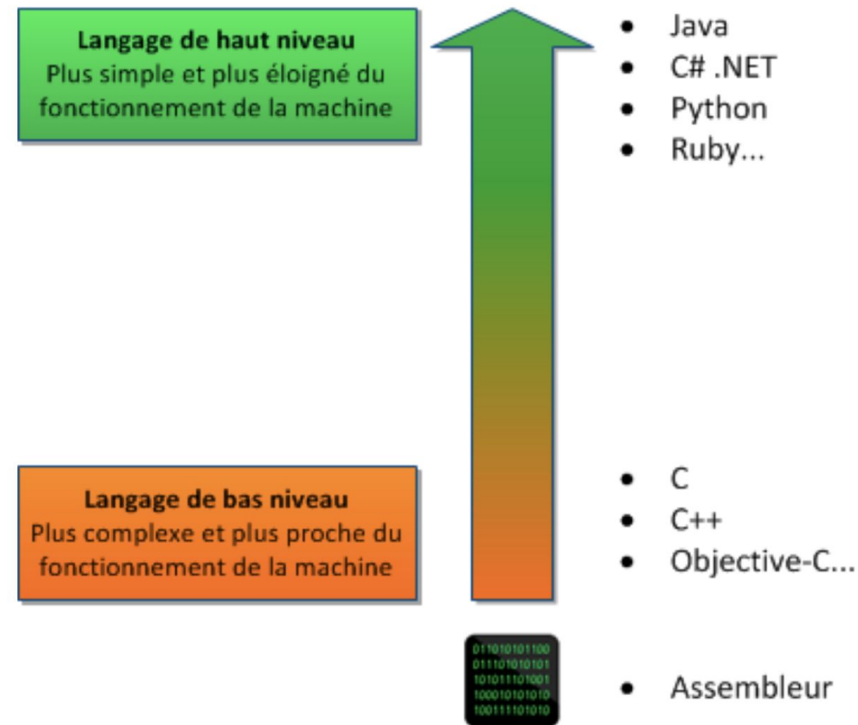
Plus le langage est haut niveau, plus il se rapproche des langages humains, c'est-à-dire qu'il sera composé de mots-clés, d'une structure et d'une syntaxe plus facile à comprendre et surtout à apprendre.

Il est important de noter que chaque langage de programmation possède une documentation, qui est un peu comme le "mode d'emploi" du langage.

Documentation Javascript :

<https://developer.mozilla.org/fr/docs/Web/JavaScript>

LANGAGE HUMAIN



LANGAGE MACHINE (Binaires)

Environnement de développement

Langages	Extension du fichier
Javascript	.js
Python	.py
PHP	.php
Java	.java
Ruby	.rb
C	.c
C++	.cpp

Comprendre la compilation et
l'interprétation d'un code

Trois façons de traduire du code source

- Compiler, c'est-à-dire le traduire en binaire
- Interpréter, c'est-à-dire le lire en temps réel et exécuter les instructions
- Hybride (compiler + interpréter)

Quelle est la différence entre compiler et interpréter ?

Prenons un exemple. Vous essayez de monter une maquette, mais la notice est dans une langue que vous ne comprenez pas. Vous pouvez, bien sûr, traduire la notice avant. Dans ce cas, c'est un langage compilé.

Mais vous pouvez aussi demander à quelqu'un qui parle cette langue de lire en traduisant la notice pendant que vous montez le modèle. Dans ce cas, c'est un langage interprété.

Trois façons de traduire du code source

Quelle est la différence entre compiler et interpréter ?

Prenons un exemple. Vous essayez de monter une maquette, mais la notice est dans une langue que vous ne comprenez pas. Vous pouvez, bien sûr, traduire la notice avant. Dans ce cas, c'est un langage compilé.

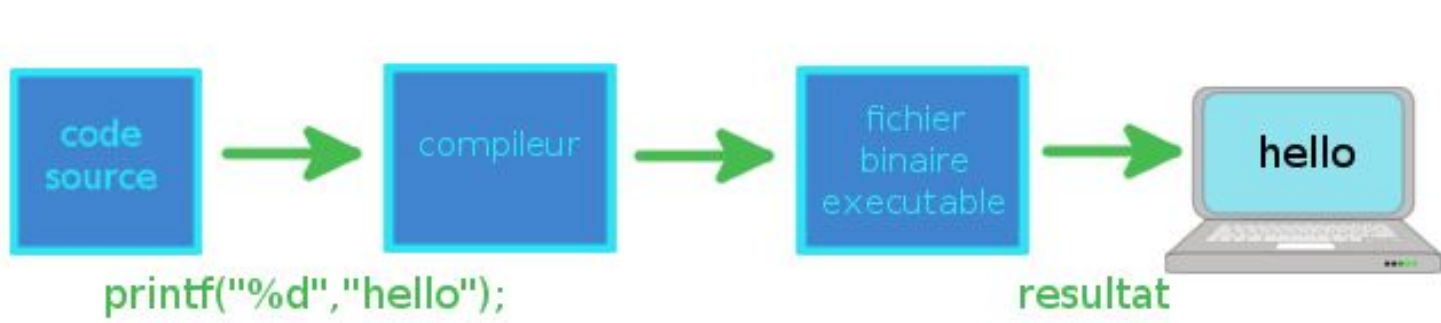
Certains langages sont considérés comme hybrides car ils requièrent d'être préalablement compilés pour ensuite être interprétés. C'est le cas, par exemple, de Java, C# ou Python. À présent, voyons comment exécuter du code.

Langage compilé

Si un langage est de type compilé, il aura besoin d'un compilateur pour être traduit.

Les compilateurs traduisent le code en langage machine. Une fois cette traduction effectuée, vous pouvez distribuer le fichier résultant sur tous les postes des utilisateurs.

Exemple : C, C++ et Objective-C



Langage compilé

Avantages :

- Programme immédiatement disponible à démarrer
- Plus “rapide”, car optimisé
- Code source est privée

Inconvénients :

- Non multi-plateforme
- Nécessite des étapes supplémentaires pour tester

Langage interprété

Concernant les langages interprétés comme PHP ou JavaScript, ils auront besoin d'un interpréteur.

Les interpréteurs informatiques, quant à eux, traitent votre code source à chaque exécution, ligne par ligne, et c'est à l'autre utilisateur d'avoir l'interpréteur nécessaire disponible sur sa machine

Exemple : PHP et JavaScript



Langage interprété

Avantages :

- Multi-plateforme
- Simple à tester
- Facile à débbugger
- Code source public (Communauté open source)

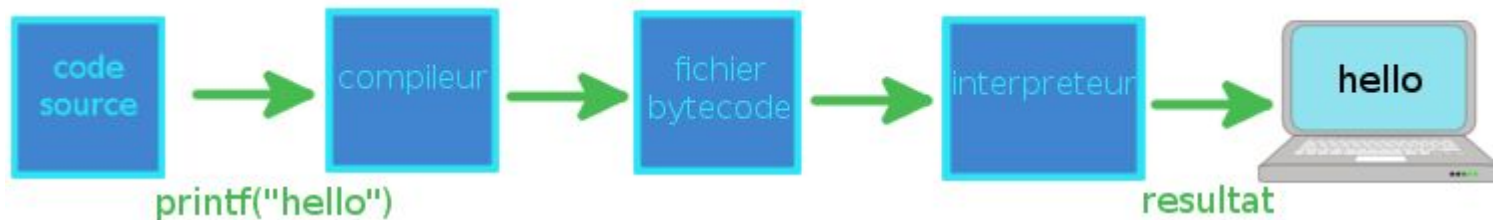
Inconvénients :

- Requier un interpréteur
- Pour chaque exécution, le programme doit être interprété préalablement

Langage hybride

Certains langages sont considérés comme hybrides car ils requièrent d'être préalablement compilés pour ensuite être interprétés

Exemple : Java, C# et Python



Langage hybride

Avantages :

- Multi-plateforme
- Simple à tester
- Facile à débbugger
- Code source compilé seulement si version différente de version compilé
- Code source public (Communauté Open Source)

Inconvénients:

- Requiert un interpréteur

Débogger du code

Erreurs courantes en programmation

- Erreur de syntaxe
- Erreur sémantique (ou erreur de logique)
- Erreur d'exécution

Erreur de syntaxe

Les règles du langage (sa syntaxe) n'ont pas été respecté

Erreur de syntaxe

Bonne syntaxe

```
console.log("Hello World");
```

Mauvaise syntaxe

```
console.log("Hello World')
```

```
console.log("Hello World";
```

```
Console.log("Hello World");
```

```
consolelog("Hello World");
```

Erreur sémantique (ou de logique)

La sortie du programme n'est pas celle que vous attendiez

Erreur sémantique (ou de logique)

Code correct

```
let prenom = "Bob";  
  
alert( "Hello " + prenom );
```

Résultat : Hello Bob

Code avec erreur sémantique

```
let prenom = "Bob";  
  
alert( "Hello prenom" );
```

Résultat : Hello prenom

Erreur d'exécution (ou Runtime error)

Erreur de programme qui se produit pendant l'exécution du programme. L'ordinateur n'a pas pu exécuter une partie de votre code.

Exemple:

- Mémoire insuffisante
- votre programme essaie de lire un fichier qui n'existe plus
- Erreur de calcul

Erreur d'exécution (ou Runtime error)

```
let i = 1;  
  
while ( i < 10 ) {  
    console.log( i );  
}
```

Résultat: Boucle infini

NOTION 1 : LES VARIABLES

Comprendre la notion de variable

Les variables sont des conteneurs dans lesquels on peut stocker des valeurs.

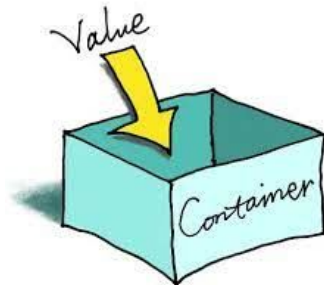
En Javascript, il faut déclarer une variable avec le mot-clé `let` en le faisant suivre de son nom :

```
let maVariable;
```

// Je déclare ma variable pour la première fois. On lui donne une valeur :

```
let maVariable;
```

```
maVariable = "Bob";
```



Comprendre la notion de variable

// On peut faire les deux opérations sur une même ligne :

```
var maVariable = 'Bob';
```

// Vous retrouvez la valeur en appelant simplement la variable par son nom :

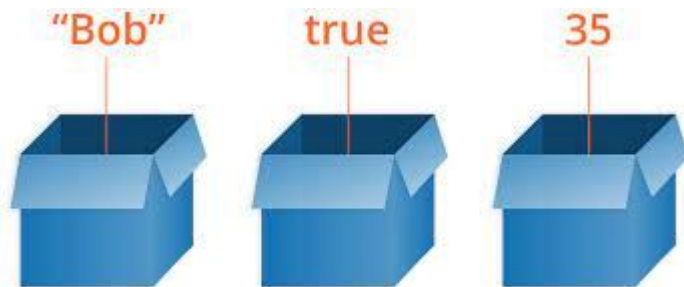
```
maVariable;
```

// Une fois qu'on a donné une valeur à une variable, on peut la changer plus loin :

```
let maVariable = 'Bob';  
maVariable = 'Étienne';
```

Les différents types de variables

- String (ou chaîne de caractère): `"Bob"` ou `'Bob'`
- Number (nombre): `35`
- Boolean (booléen): `true`, `false`
- Array (tableau): `["Bob", 12345]`
- Object (objet): `{ pseudo: "Bob", email: "mon-adresse-mail@exemple.com"`
`}`



Norme de nommage d'une variable en Javascript

- Utiliser la forme de nommage camelCase (Exemple: maVariable)
 - première lettre en minuscule
 - la première lettre de chaque mot suivant en Majuscule
- sensible à la casse (Exemple : mavariable ≠ maVariable)
- Ne pas utiliser les mots réservés (Exemple : let, const, function, return, if, else, while etc....)

Les variables à travers les langages informatiques

Javascript



```
let name = "John";  
let myNum = 5;  
document.write(name);  
document.write(myNum);
```

Résultat :

John

5

PHP



```
$name = "John";  
$my_num = 5;  
echo $name;  
echo $my_num;
```

Résultat :

John

5

Les variables à travers les langages informatiques

PHP



```
$name = "John";  
$my_num = 5;  
echo $name;  
echo $my_num;
```

Résultat :

John
5

Python



```
name = "John"  
myNum = 5  
print(name)  
print(myNum)
```

Résultat :

John
5

Les variables à travers les langages informatiques

Python



```
name = "John"  
myNum = 5  
print(name)  
print(myNum)
```

Résultat:

John

5

C#



```
string name = "John";  
int myNum = 15;  
Console.WriteLine(name);  
Console.WriteLine(myNum);
```

Résultat:

John

5

Variantes pour les chaînes de caractère

"abcABC123 . \$"

'abcABC123 . \$'

Les opérateurs arithmétiques

Opérateurs	Symboles
Addition	+
Soustraction	-
Multiplication	*
Division	/
Modulo (reste)	%
Exponent	**

Les règles de priorité

Voici l'ordre de priorité des opérations qu'il faut respecter :

1. Les Parenthèses
2. Les Exposants
3. Les Multiplications et les Divisions (de la gauche vers la droite)
4. Les Additions et les Soustractions (de la gauche vers la droite)

Incrémentation / Décrémentation

let calcul = 10;

Opérations	Équivaut à	Résultat
calcul += 2	calcul = calcul + 2 <i>calcul = 10 + 2</i>	12
calcul -= 2	calcul = calcul - 2 <i>calcul = 10 - 2</i>	8

Incrémentation / Décrémentation

let calcul = 10;

Opérations	Équivaut à	Résultat
calcul *= 2	calcul = calcul * 2 <i>calcul = 10 * 2</i>	20
calcul /= 2	calcul = calcul / 2 <i>calcul = 10 / 2</i>	5

Incrémentation / Décrémentation

let calcul = 10;

Opérations	Équivaut à	Résultat
calcul++	calcul = calcul + 1 <i>calcul = 10 + 1</i>	11
calcul--	calcul = calcul - 1 <i>calcul = 10 - 1</i>	9

Les opérateurs comparaison

Opérateurs	Symboles
Supérieur	>
Supérieur ou égale	>=
Inférieur	<
Inférieur ou égale	<=
Différent de	!= ou !==
Égale	== ou ===

Les opérateurs de comparaison

Opérateurs	Description	Symboles
Égale	compare valeur seulement	==
Égale (<i>stricte</i>)	compare valeur + type	===
Différent de	compare valeur seulement	!=
Différent de (<i>stricte</i>)	compare valeur + type	!==

NOTION 2 : LES CONDITIONS

Définition d'une condition sans alternative

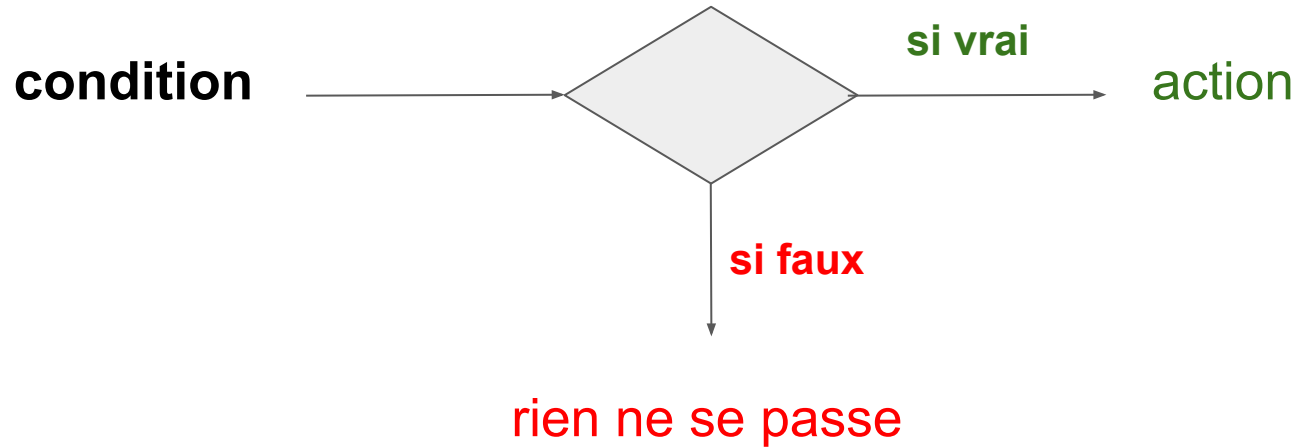
Nous prenons des décisions tous les jours. Ces décisions, généralement, sont prises selon des conditions. S'il pleut, je prends mon parapluie. S'il fait beau, je prends mes lunettes de soleil.

En programmation, il est possible d'exécuter du code selon des conditions similaires. Une condition sera soit vraie, soit fausse. Si cette dernière est vraie, cela déclenchera une action.

Un code s'exécutera dans notre programme.

Cependant, si cette condition est fausse, rien ne se passera. Aucun code ne sera exécuté.

Condition simple en programmation



Syntaxe d'une condition simple

```
if ( condition ) {  
    // Le code s'exécute si la condition est vraie  
}
```

Définition d'une condition sans alternative

Pour créer des conditions, on utilise les instructions if mais, jusqu'à présent, si la condition était fausse, nous ne faisons rien.

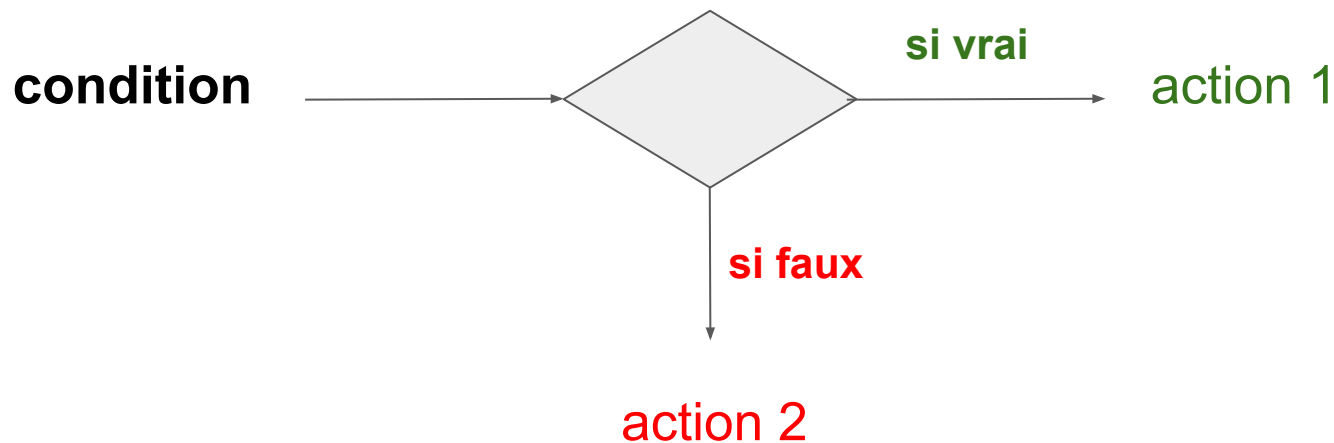
Nous allons pouvoir prendre des mesures pour que ce ne soit plus le cas. Pour cela, il faudrait ajouter une alternative. Si ma condition est vraie, il faut qu'une action se passe mais, si celle-ci est fausse, il faut indiquer au programme qu'une autre action doit s'exécuter.

Prenons un autre exemple. S'il fait chaud, je mets un tee-shirt, sinon, je prends mon manteau.

Si ma condition est fausse, s'il ne fait pas chaud, au lieu de ne rien faire, je fais une action par défaut qui consiste à mettre mon manteau. Nous venons de créer une condition avec une alternative.

En programmation, nous allons pouvoir ajouter cette alternative grâce au mot-clé else.

Condition avec une alternative



Syntaxe d'une condition avec une alternative

```
if ( condition ) {  
    // Le code s'exécute si la condition est vraie  
} else {  
    // Le code s'exécute si la condition est fausse  
}
```

Syntaxe d'une condition avec plusieurs alternatives

```
if ( condition 1 ) {  
    // Le code s'exécute si la condition 1 est vraie  
} else if ( condition 2 ) {  
    // Le code s'exécute si la condition 2 est vraie  
} else {  
    // Le code s'exécute si aucune condition n'est vraie  
}
```

Les conditions composées avec le ET

Conditions composées	Résultats
true & true	true
true & false	false
false & true	false
false & false	false

Les conditions composées avec le OU

Conditions composées	Résultats
true & true	true
true & false	true
false & true	true
false & false	false

Les conditions à travers les langages informatiques

Javascript



```
let age = 29;

if (age < 18) {
  document.write('Vous êtes mineur');
} else {
  document.write('Vous êtes majeur')
}
```

Résultat :

Vous êtes majeur.

C++



```
int age = 29;

if (age < 18) {
  cout << "Vous êtes mineur.";
} else {
  cout << "Vous êtes majeur";
}
```

Résultat :

Vous êtes majeur.

Les conditions à travers les langages informatiques

C++



```
int age = 29;
if (age < 18) {
    cout << "Vous êtes mineur.";
} else {
    cout << "Vous êtes majeur";
}
```

Résultat :

Vous êtes majeur.

Python



```
age = 29
if age < 18:
    print("Vous êtes mineur.")
else:
    print("Vous êtes majeur")
```

Résultat :

Vous êtes majeur.

Les conditions à travers les langages informatiques

Python



```
age = 29
if age < 18:
    print("Vous êtes mineur.")
else:
    print("Vous êtes majeur")
```

Résultat :

Vous êtes majeur.

Ruby



```
age = 29
if age < 18
    print 'Vous êtes mineur.'
else
    print 'Vous êtes majeur.'
end
```

Résultat :

Vous êtes majeur.

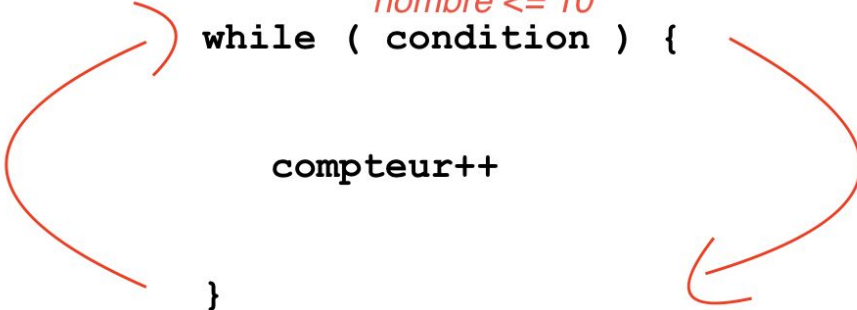
NOTION 3 : LES BOUCLES

Syntaxe d'une boucle while

```
while ( condition ) {  
    // Le code s'exécute si la condition est vraie  
}
```

Syntaxe d'une boucle while

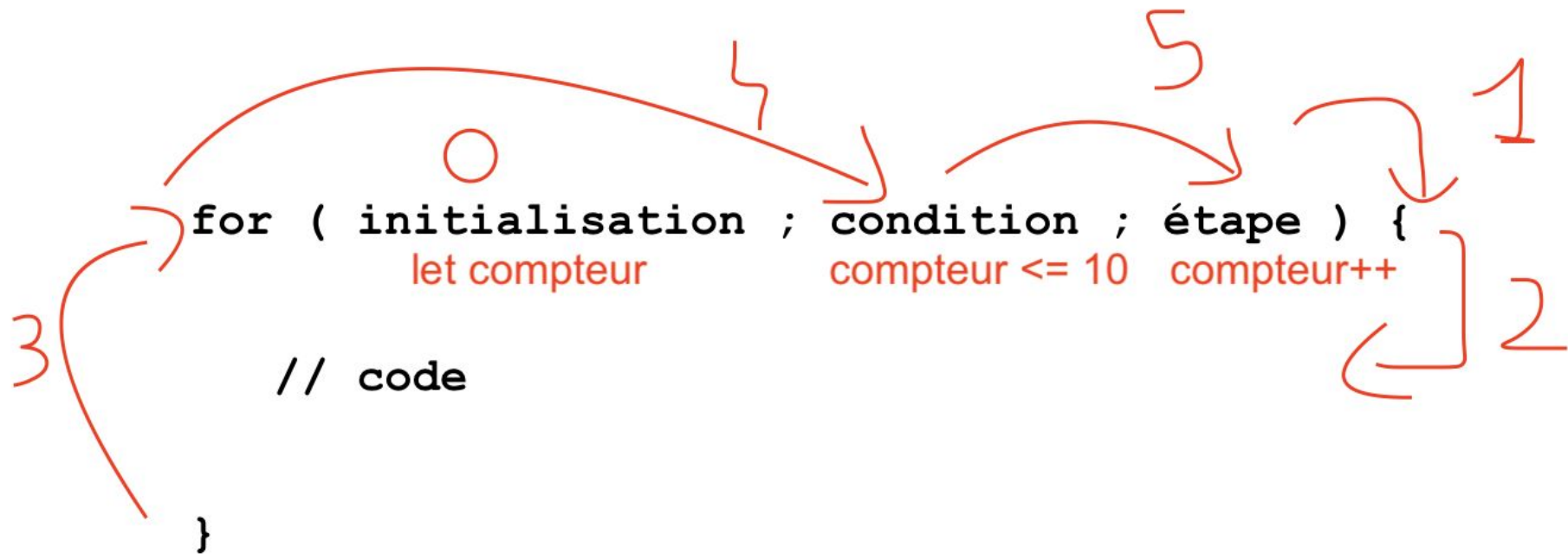
```
let compteur = 1;  
?  
while ( nombre <= 10 condition ) {  
    compteur++  
}
```



Syntaxe d'une boucle for

```
for ( initialisation ; condition ; étape ) {  
    // Le code s'exécute  
}
```

Syntaxe d'une boucle for



Les boucles à travers les langages informatiques

Javascript



```
// Boucle while  
let i = 0;  
while (i < 5) {  
    document.write(i);  
    i++;  
}
```

Java



```
// Boucle while  
int i = 0;  
while (i < 5) {  
    System.out.println(i);  
    i++;  
}
```

Les boucles à travers les langages informatiques

Java



```
// Boucle while  
int i = 0;  
while (i < 5) {  
    System.out.println(i);  
    i++;  
}
```

Python



```
// Boucle while  
i = 0  
while i < 5:  
    print(i)  
    i += 1
```

Les boucles à travers les langages informatiques

Python



```
// Boucle while  
i = 0  
while i < 5:  
    print(i)  
    i += 1
```

Ruby



```
// Boucle while  
$i = 0  
while $i < 5  
    puts($i)  
    $i +=1  
end
```


Exemples d'autres types de boucles

Boucle do..while

```
do {  
    // Instructions  
} while ( condition );
```

Boucle for...in

```
for variable in objet {  
    // Instructions  
}
```

NOTION 4 : LES FONCTIONS

Le principe DRY

Don't

Repeat

Yourself

Syntaxe d'une fonction

```
function maFonction() {  
    // instructions de la fonction  
}  
  
// On appelle la fonction pour que le code s'execute  
maFonction();
```

Syntaxe d'une fonction avec des paramètres

```
function maFonction(argument1, argument 2) {  
    // instructions de la fonction  
}
```

// On indique les informations relatifs aux arguments
lors de l'appel de la fonction

```
maFonction(donneeArgument1, donneeArgument2);
```

Les fonctions à travers les langages informatiques

Javascript



```
// Crée la fonction
function direBonjour() {
    document.write("Bonjour !");
}

// Appel la fonction
direBonjour();

// Résultat : Bonjour !
```

Python



```
// Crée la fonction
def dire_bonjour():
    print("Bonjour !")

// Appel la fonction
dire_bonjour()
```

Les fonctions à travers les langages informatiques

Python



```
// Crée la fonction
```

```
def dire_bonjour():  
    print("Bonjour !")
```

```
// Appel la fonction
```

```
dire_bonjour()
```

Perl



```
// Crée la fonction
```

```
sub direBonjour {  
    print("Bonjour !");  
}
```

```
// Appel la fonction
```

```
direBonjour();
```

Les fonctions à travers les langages informatiques

Perl



```
// Crée la fonction
sub direBonjour {
    print("Bonjour !");
}

// Appel la fonction
direBonjour();
```

C++



```
// Crée la fonction
void direBonjour() {
    cout << "Bonjour !";
}

// Appel la fonction
int main() {
    direBonjour();
    return 0;
}
```


NOTION 5 : LES TABLEAUX ET OBJETS

Syntaxe d'un tableau

```
let liste = ["Bob", "Emmanuelle", "Tony"];
```

Récupérer un élément dans un tableau

```
let liste[index];
```

Récupérer un élément dans un tableau

```
let liste = ["Bob", "Emmanuelle", "Tony"];
```

```
liste[0]; // "Bob"
```

```
liste[2]; // "Tony"
```

Récupérer un élément dans un tableau

```
           0           1           2
let liste = ["arbre", "écureuil", [7, 8, 9]];
let liste[2]; // "[7, 8, 9]"
```

```
           0     1     2
let liste = ["arbre", "écureuil", [7, 8, 9]];
let liste[2][1]; // 8
```

Découvrir la méthode forEach

```
var tableau = ["item1", "item2", "item3"]

tableau.forEach(function(elementTableau) {
    console.log(elementTableau);
});

// "item1"
// "item2"
// "item3"
```

Exemple d'un objet informatique

```
let stylo = {  
  marque: "Bic",  
  couleur: "bleu",  
  type: "bille",  
  ecrire: function () {  
    console.log("J'écris ...");  
  }  
};
```

Exemple d'un objet informatique

```
let stylo = {
```

```
  marque: "Bic",  
  couleur: "bleu",  
  type: "bille",
```

Propriétés de l'objet
= caractéristiques (appelés aussi attributs)

```
  ecrire: function () {  
    console.log("J'écris ...");  
  }
```

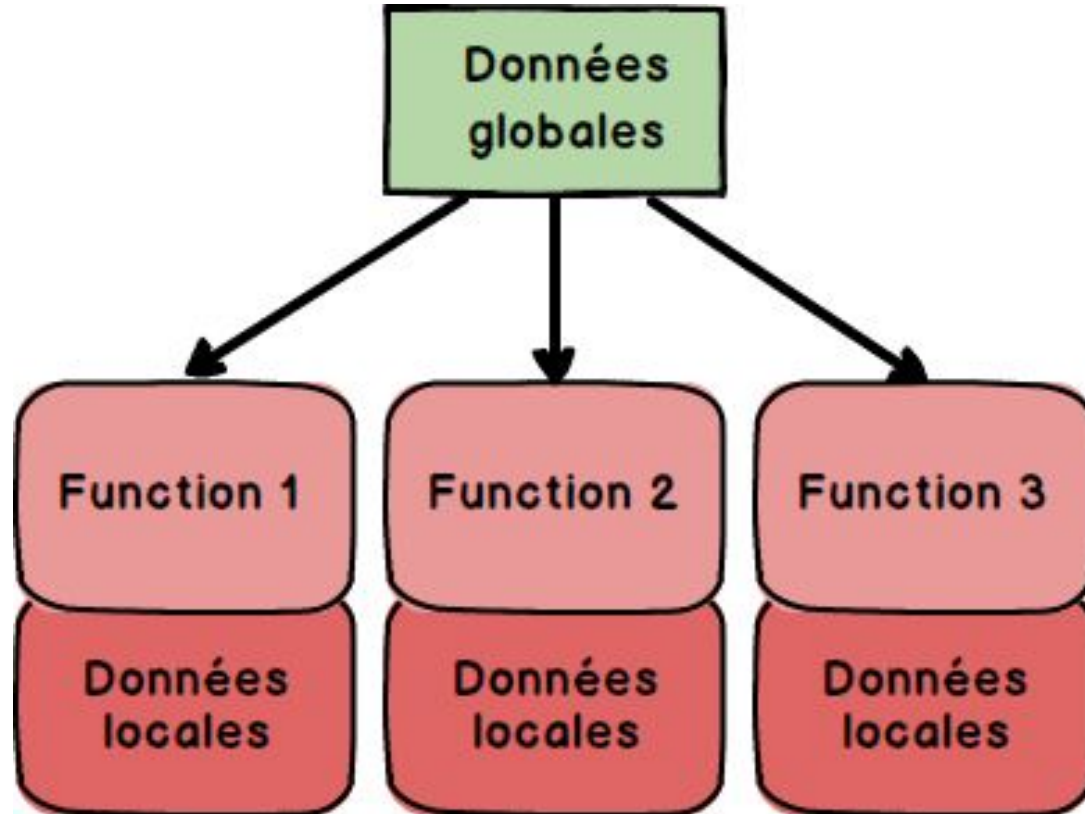
Méthode de l'objet
= comportements

```
};
```

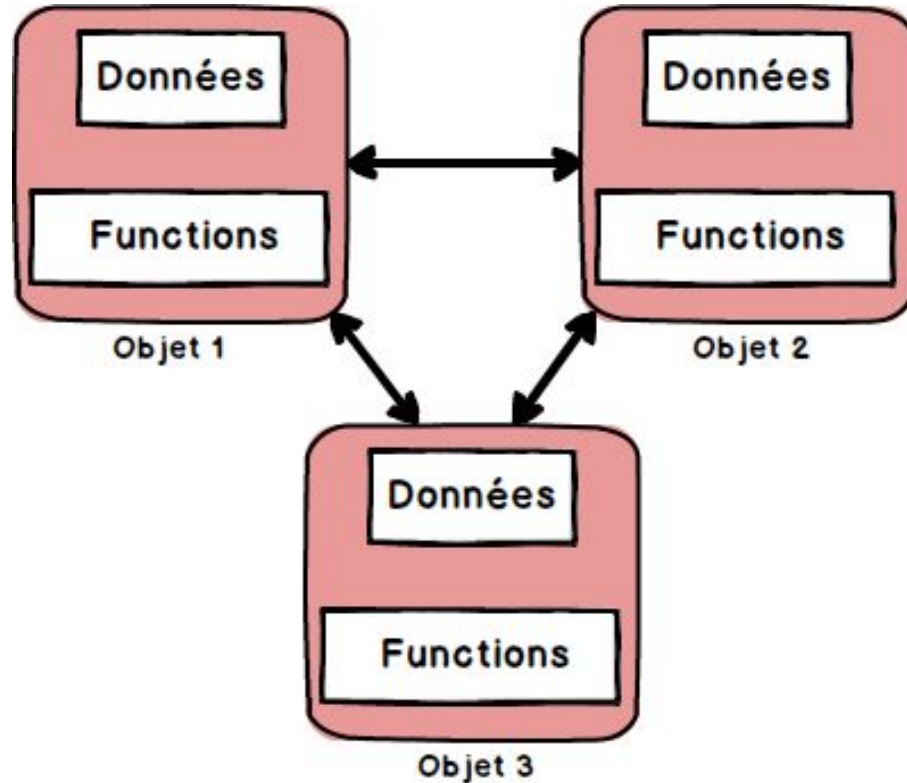

Définition d'un objet informatique

- Un objet informatique est une entité qui possède des propriétés et des méthodes.
- Chaque propriété définit une caractéristique, une information de l'objet (exemple : la couleur)
- Chaque méthode définit un comportement, une action de l'objet (exemple : écrire)

Programmation procédurale



Programmation orienté objet



Problématique...

```
let voiture1 = {  
  couleur : "blanc",  
  marque : "suzuki celerio",  
  prix: 8000,  
  rouler: function () {  
    console.log("Je roule");  
  }  
}
```

```
let voiture2 = {  
  couleur : "rouge",  
  marque : "renault capture",  
  prix: 23400,  
  rouler: function () {  
    console.log("Je roule");  
  }  
}
```

Solution : La classe

```
class Vehicule {  
  constructor(arg1, arg2, arg3) {  
    this.couleur = arg1;  
    this.marque = arg2;  
    this.prix = arg3;  
  }  
  rouler() {  
    console.log("Je roule");  
  }  
}  
  
let voiture1 = new Vehicule("blanc", "suzuki celerio", 8000);  
let voiture2 = new Vehicule("rouge", "renault capture", 23400);
```

Instanciation = créer un objet à partir d'une classe.

Ici on dit que voiture1 et voiture2 sont des instances de la classe Vehicule