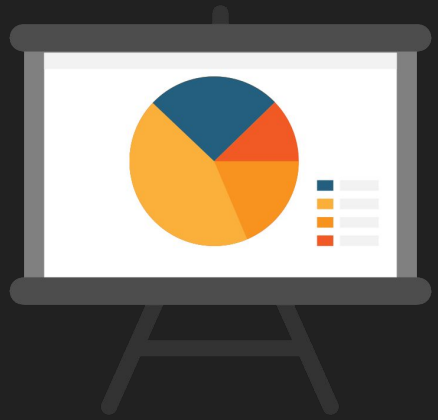
Algorithmique

Rachid EDJEKOUANE (edjek@hotmail.fr)

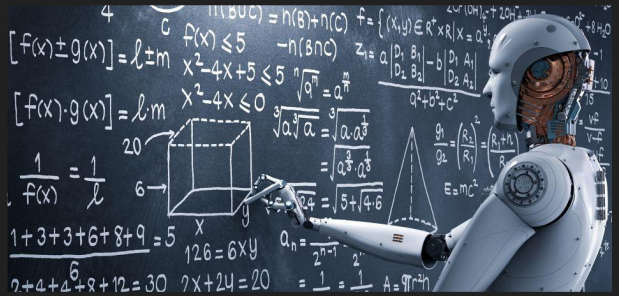
Introduction



Qu’est-ce qu’un algorithme ?

Un algorithme est la description d'une suite d'étapes permettant d'obtenir un résultat à partir d'éléments fournis en entrée.

Par exemple, une recette de cuisine est un algorithme permettant d'obtenir un plat à partir de ses ingrédients!



Algorithmique et programmation

Pourquoi apprendre l'algorithmique pour apprendre à programmer ?

Si on trouve des algorithmes dans la vie de tous les jours, pourquoi en parle-t-on principalement en informatique ?

Parce que l'algorithmique exprime les instructions résolvant un problème donné indépendamment des particularités de tel ou tel langage.

La raison est très simple : les ordinateurs sont très pratiques pour effectuer des tâches répétitives. Ils sont rapides, efficaces, et ne se lassent pas.

Algorithmique informatique

Pour qu'un algorithme puisse être mis en œuvre par un ordinateur, il faut qu'il soit exprimé dans un langage informatique, sous la forme d'un logiciel (souvent aussi appelé « application »).

Un logiciel combine en général de nombreux algorithmes : pour la saisie des données, le calcul du résultat, leur affichage, la communication avec d'autres logiciels, etc…

Programmation

Un algorithme exprime la structure logique d'un programme informatique et de ce fait est indépendant du langage de programmation utilisé.

Par contre, la traduction de l'algorithme dans un langage particulier dépend du langage choisi et sa mise en œuvre dépend également de la plateforme d'exécution.

Programmation

Langage de programmation

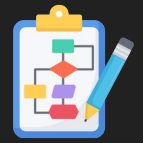
- Un langage de programmation est un langage informatique, permettant à un humain d’écrire un code source qui sera analysé par un ordinateur.

Compilateur

- Un compilateur est un programme informatique qui traduit un langage, le langage source, en un autre, appelé le langage cible.

Interpréteur

- Un interpréteur est un outil informatique (logiciel ou matériel) ayant pour tâche d’analyser et d’exécuter un programme écrit dans un langage source.



Découverte

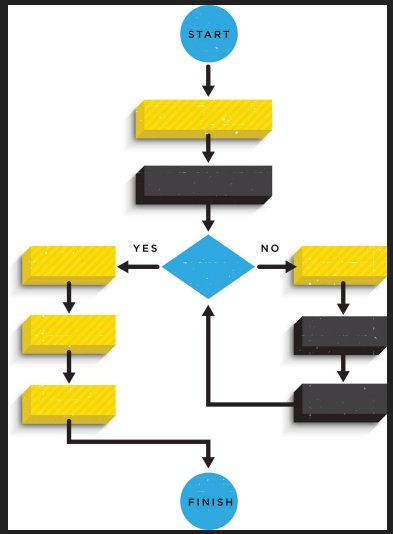
Algorithmique : comment ça marche ?

Pleins de possibilités pour décrire un algorithme :

Le pseudo-code :

```
si condition alors
    bloc 1
sinon
    bloc 2
```

Une autre manière de visualiser cette instruction, sous forme d'algorigramme :



Algorithmique : en Javascript

Dans un langage informatique :

Javascript (chaque langage a sa syntaxe) :

```
1  let age = prompt('quel ages as-tu?');
2
3  if (age >= 18) {
4      console.log('tu es majeur!');
5  } else {
6      console.log('tu es mineur');
7  }
```

Algorithmique : résolution d'un problème

- Établir la liste des données en entrée(données à saisir), la liste des données en sortie(résultats : données à afficher) et les liens entre elles.
- Construire un chemin de résolution qui permet d'obtenir les données en sortie à partir des données en entrée. C'est ce qu'on appelle un schéma de résolution.
- Décrire le schéma de résolution en termes d'instructions élémentaires acceptées par ordinateur. C'est l'Algorithme.

Algorithmique : comment ça marche ?

Validité d’un algorithme

- La validité d’un algorithme est son aptitude à réaliser exactement la tâche pour laquelle il a été conçu.

Robustesse d’un algorithme

- La robustesse d’un algorithme est son aptitude à se protéger de conditions anormales d’utilisation.

Réutilisabilité d’un algorithme

- La réutilisabilité d’un algorithme est son aptitude à être réutilisé pour résoudre des tâches équivalentes à celle pour laquelle il a été conçu.

Algorithmique : comment ça marche ?

Complexité d’un algorithme

- La complexité d’un algorithme est le nombre d’instructions élémentaires à exécuter pour réaliser la tâche pour laquelle il a été conçu.

Efficacité d’un algorithme

- L’efficacité d’un algorithme est son aptitude à utiliser de manière optimale les ressources du matériel qui l’exécute.

Algorithmique : conventions de nommage

Il existe plusieurs conventions de nommage utilisées dans le développement logiciel pour garantir une lisibilité et une compréhension cohérentes du code :

- PascalCase : MaVariable, MonObjet.
- camelCase : maVariable, monObjet.
- snake_case : ma_variable, mon_objet.
- kebab-case : ma-variable, mon-objet.
- UPPERCASE : MA_VARIABLE, MON_OBJET.

Algorithmique : conventions de nommage

Il est important de noter que les conventions de nommage ne sont pas seulement une question de préférence personnelle, mais également une question de normes dans les communautés de développement.

Il est donc important de suivre les conventions de nommage utilisées dans les projets et les langages de programmation spécifiques.

En pratique

Algorithmique : les variables

Dans un programme informatique, on va avoir en permanence besoin de stocker provisoirement des valeurs.

Il peut s'agir de données issues du disque dur, fournies par l'utilisateur (frappées au clavier), bases de données. Il peut aussi s'agir de résultats obtenus par le programme, intermédiaires ou définitifs.

Ces données peuvent être de plusieurs types (on en reparlera) : elles peuvent être de type number, string, boolean, table...

Toujours est-il que dès que l'on a besoin de stocker une information au cours d'un programme, on utilise une Variable.

Algorithmique : les variables

Une variable est une entité qui contient une information, elle possède :

- un nom, on parle d'identifiant (On préfère que ce nom soit significatif)
- une valeur
- un type qui caractérise l'ensemble des valeurs que peut prendre la variable

L'ensemble des variables est stocké dans la mémoire de l'ordinateur.

Algorithmique : les types de données

Qq exemples de types de variables :

- number (1, 78, 0.5, …)
- string 'hello', 'un message'
- boolean (true, false)
- array [12, 'test', true]
- object {
  firstName: 'julien',
  lastName: 'Dupont'
  }

Algorithmique : les instructions conditionnelles

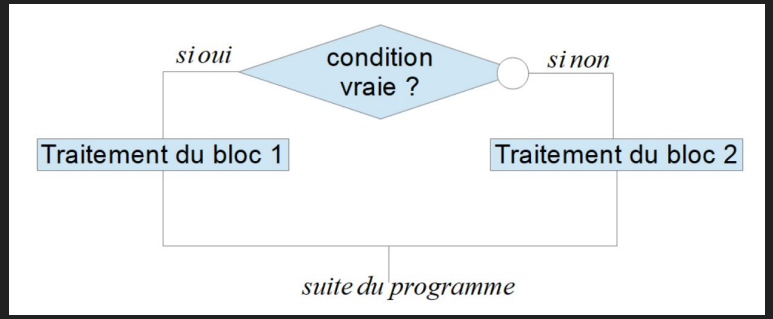
Une condition est une expression qui relie deux valeurs et/ou variables à l'aide d'un opérateur de comparaison.

On peut aussi écrire des conditions composées.

Une instruction conditionnelle est une instruction qui n'est exécutée que si une condition est validée.

Lorsque l'Algorithme est exécuté, les conditions sont testées. Le résultat du test peut valider ou non la condition.

Algorithmique : instructions conditionnelles




```
let condition = 18;

if (condition == 18) {
    console.log('tout est ok');
} else {
    console.log('Houston, nous avons un problème');
}
```

Algorithmique : opérateurs de comparaison

Voici la liste des opérateurs de comparaison qui existent :

== est l'opérateur d'égalité.
=== est l'opérateur d'égalité et de type.
!= est l'opérateur de différence.
!== est l'opérateur de différence.
> est l'opérateur « strictement supérieur à ».
>= est l'opérateur « supérieur ou égal à ».
< est l'opérateur « strictement inférieur à ».
<= est l'opérateur « inférieur ou égal à ».

Algorithmique : les conditions composées

On peut écrire des conditions composées, en fusionnant deux (ou plusieurs) conditions simples avec les opérateurs logiques ET et OU.

Avec l'opérateur ET, il faut que les deux conditions simples soient validées pour que la condition composée soit validée.

```
prix > 15 ET prix < 30.
```

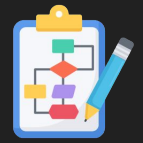
Avec l'opérateur OU, il faut qu'au moins l'une des deux conditions simples soit validée pour que la condition composée soit validée.

```
age ≤ 26 OU age ≥ 60.
```

Algorithmie : opérateurs logiques

```
// Les opérateurs logique OU (||) et ET (&&)
let permis = true;
if (condition >= 18 || permis == true) {
    console.log('Tu es majeur et tu as le permis');
} else {
    console.log('Houston, nous avons 2 problèmes');
}
```

Algorithmie : les boucles



Une boucle avec condition de bouclage permet d'indiquer qu'une même instruction, ou série d'instructions, doit être répétée tant qu'une condition est vraie.

Algorithmie : boucles à itérations définies

Une boucle à itérations définies (for) permet d'indiquer qu'une même instruction, ou série d'instructions, doit être répétée un certain nombre de fois.

Le nombre de répétitions est indiqué par un compteur, dont on précise la valeur de départ, la condition et la valeur de fin.

La boucle à itérations définies s'écrit en javascript :

```
1  for (let i = 0; i < 5; i++) {
2
3  }
```

Algorithmique : boucles à itérations non définies

Une instruction while permet d'exécuter une instruction tant qu'une condition donnée est vérifiée.

Si la condition n'est pas vérifiée, l'instruction n'est pas exécutée (ignorée) et le contrôle passe directement à l'instruction suivant la boucle.

```
let age = prompt('quel ages as-tu?');

while (age < 18) {
    age = prompt('Tu es mineur, tu ne peux pas rentrer!');
}
```

Algorithmique : fonctions et procédures

Une procédure est un sous-algorithme qu'on écrit à part. Une procédure peut être utilisée depuis l'algorithme principal, on dit alors qu'elle est appelée.

Une fonction est une procédure, un ensemble d'instructions effectuant une tâche ou calculant une valeur.

Afin d'utiliser une fonction, il est nécessaire de l'avoir auparavant définie au sein de la portée dans laquelle on souhaite l'appeler.

Algorithmique : déclaration de fonction

Une définition de fonction ou déclaration de fonction est construite par :

- un nom de la fonction.
- une liste d'arguments à passer à la fonction, entre parenthèses et séparés par des virgules.
- les instructions définissant la fonction.

Algorithmique : fonction en pratique

Un bon informaticien est un informaticien fainéant !

Cela veut dire tout simplement, qu'il ne faut jamais faire 2 fois le même travail. Un bon informaticien doit savoir réutiliser quelque chose déjà existant plutôt que de le refaire à nouveau.

```
1  function add(x, y) {
2      return x + y;
3  }
4
5  let result = add(7, 3);
6  console.log(result);
```

Algorithmique : portée des variables (scope)

La portée d'une variable ou scope désigne l'espace du script dans lequel elle va être accessible.

En effet, toutes nos Variables ne sont pas automatiquement disponibles à n'importe quel endroit dans un script et on ne va donc pas toujours pouvoir les utiliser partout.

Algorithmique : portée globale (scope)

Scope global (portée globale) : une variable définie en dehors de toutes les fonctions ou blocs est une variable globale.

Elle peut être accédée et modifiée de n'importe où dans le code, y compris à l'intérieur des fonctions.

```
var x = 10; // variable globale

function foo() {
  console.log(x); // accéder à la variable globale
}

foo(); // affiche 10
```

Algorithmique : portée block ou fonction

Une variable définie à l'intérieur d'une fonction ou d’un bloc est une variable locale et ne peut être accédée qu'à l'intérieur de ce bloc ou de cette fonction.

```
1  function foo() {
2      var x = 10; // variable locale
3      console.log(x); // accéder à la variable locale
4    }
5
6    foo(); // affiche 10
7    console.log(x); // erreur : la variable x n'est pas définie
8
```

```
if (true) { echo 'merci'; }
```