Algorithmique partie 1

Plan

- Introduction et définition

- Algorithmique partie 1
  - Données : informations élémentaires
    - Le type
    - Le nom
    - Variables et constantes
  - Données : les actions élémentaires
    - La déclaration de variables
    - Les affectations
    - La lecture
    - L’écriture
  - Les structures
    - Les structures conditionnelles
    - Les structures itératives

Avant-propos

Les nouvelles technologies évoluent sans cesse et à un rythme effréné, qu'elles concernent les appareils connectés ou les sites web, cette course en avant semble inarrêtable et difficilement prévisible, en raison de la multiplicité des facteurs de corrélation ou de causalité.

Ainsi, l'apprentissage se doit également d'être moderne.
Être un rat de bibliothèque adepte de l'apprentissage par coeur est désuet, "old school" depuis plusieurs années.

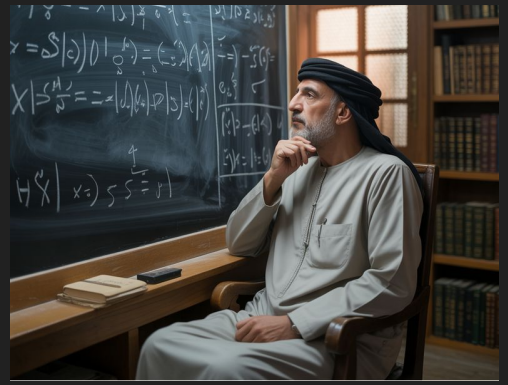
L'objectif de ce cours est de transmettre les connaissances nécessaires, mais surtout de développer l'autonomie, car comme cité précédemment les technologies évoluent bien trop vite pour permettre de capitaliser uniquement sur les acquis d'une formation.

Introduction

L'algorithmique est une notion ancienne non propre à l'informatique, les scribes égyptiens et les savants de la Mésopotamie utilisaient déjà l'algorithmique pour résoudre des problèmes arithmétiques.

Le terme algorithme vient du "latin médiéval algorithmus, latinisation du nom d'un mathématicien de langue arabe, avec influence du grec arithmos, nombre" (Dictionnaire Larousse).

=> Vient d'un mathématicien (Muḥammad ibn Mūsā al-Khwārizmī) qui a rédigé un traité sur les nombres indiens, traduit en latin au XII^{e} siècle sous le titre Algoritmi de numero Indorum, a donné naissance au mot algorithme (latinisation de al-Khwārizmī).



Algorithmique : définition et application

Une première définition de l'algorithmique ressort : l'art de décomposer un problème en événements simples.

Exemples d'algorithmes quotidiens : une todolist pour préparer un voyage, une recette de cuisine, etc...

En informatique, le rôle de l'algorithme est fondamental.
Sans algorithme, il n'y aurait pas de programme. Celui-ci n'est que sa traduction dans un langage compréhensible par l'ordinateur.
Les algorithmes sont indépendants à la fois :

- de l'ordinateur qui les exécute ;
- des langages dans lesquels ils sont énoncés et traduits.

=> L'algorithme permet à partir de données d'obtenir un résultat.

Algorithmique : définition et application

Contexte informatique :

Un **langage de programmation** est un langage formel conçu pour permettre à un humain de décrire des instructions, des algorithmes et des structures de données sous forme de **code source**. Ce code peut ensuite être traduit ou exécuté par un ordinateur afin de réaliser des tâches précises.

Un **compilateur** est un programme qui traduit un code source écrit dans un langage de programmation (langage source) vers un autre langage (langage cible), généralement le **langage machine.**

Un **interpréteur** est un logiciel (ou parfois un matériel spécialisé) qui lit, analyse et exécute directement un programme écrit dans un langage source, **instruction par instruction**, sans générer de fichier exécutable intermédiaire.

Algorithmique : langage naturel et pseudo-code

Langage naturel	Pseudo-code
1. Additionner 2 nombres 2. Afficher le résultat	Début somme ← 2 + 3 ecrire(somme) Fin

Nous partons ainsi d'une **description textuelle** (algorithme) qui définit les étapes et on la traduit en **instructions structurées** (pseudo-code).

Cet exemple est purement illustratif, en pratique nous pourrions soit décomposer davantage en ajoutant la déclaration de la variable somme ou passer directement ecrire(2 + 3).

Algorithmique : les éléments d’un algorithme

Un algorithme est généralement constitué de plusieurs éléments :

- Données
- Actions / événements
- Résultat / retour

Les données comportent des informations élémentaires, elles sont désignées par un type, un nom et une valeur.

- Données : informations élémentaires
  - Le type
  - Le nom
  - Variables et constantes

Algorithmique : les données

Le type

- Soumis aux lois de l'électricité, une machine informatique ne connaît que 2 valeurs : éteint ou allumé (0 ou 1), c’est à dire une donnée binaire. Néanmoins une suite de données binaires peut constituer des chiffres et des lettres et tout un grand nombre d’informations suivant le type de codage binaire (8bits, 16bits, etc…). Les données binaires peuvent être converties en hexadécimal.

Principaux types en JavaScript : number, string, boolean, etc… (nous aborderons les autres types par la suite

Algorithmique : les données

Le nom

Dans l'hypothèse d'un algorithme, le nom d'une information élémentaire peut-être librement choisi.

En revanche, dans le cadre d'une application informatique/Web, en fonction des langages et de la nature de l'information différentes nomenclatures existent.

Principales conventions de dénomination	Format
Flat case	nomdefamille
Camel case	nom**D**e**F**amille
Pascal case	**N**om**D**e**F**amille
Snake case ou underscore case	nom_de_famille
Kebab case (spinal case, dash case, lisp case)	nom-de-famille

Algorithmique : les données

Le nom

Autres conventions de dénomination	Format
Upper flat case	NOMDEFAMILLE
Screaming snake case (Macro Case, Constant Case)	NOM_DE_FAMILLE
Camel snake case	nom_De_Famille
Pascal snake case	Nom_De_Famille
Cobol case (Screaming kebab case)	NOM-DE-FAMILLE
Train Case (HTTP header case)	Nom-De-Famille

Algorithmique : les données

Variables et constantes

La détermination du choix entre variable et constante peut être primordial dans le cadre d'un algorithme pour éviter les erreurs.

Exemple : algorithme permettant de calculer le montant TTC avec une TVA de 20%.

Le taux de TVA est défini dès le début et ne changera pas au cours de l'algorithme.

Algorithmique - les données : les actions

Les actions élémentaires

- La déclaration de variables
- Les affectations
- La lecture
- L’écriture

Algorithmique - les données : les actions

La déclaration de variables

Précédemment nous avons pu observer que les variables et les constantes peuvent contenir des valeurs.
La première action élémentaire consiste à déclarer le nom des variables et leur type

```
Déclaration :
variables : nom: caractère (chaîne de caractères);
TVA : entier;
```

Algorithmique - les données : les actions

Les affectations

Précédemment nous avons pu observer que les variables et les constantes peuvent contenir des valeurs.
L'affectation est une opération qui permet d'attribuer (affecter) une valeur à la variable.

L'affectation peut s'effectuer avec le signe ← ou = :

```
A = 2
nom = "Martin"
TVA = 20
```

ou

```
A ← 2
nom ← "Martin"
TVA ← 20
```

Algorithmique - les données : les actions

La lecture

C’est une opération qui fait entrer une valeur dans une variable par l’intermédiaire d’un périphérique d’entrée (le clavier, la souris, un scanner, etc...).

```
lire(variable)
lire(variable1, variable2, variable3)
```

La lecture est une forme particulière d’affectation. Voici un petit exemple pour comprendre la différence entre affectation et lecture :

```
A=5
lire(A)
```

Algorithmique - les données : les actions

L’écriture

L’opération d’écriture imprime sur un périphérique de sortie (écran, imprimante,...) le contenu d’une variable. C’est l’action complémentaire de la lecture.

L’écriture ne modifie pas la valeur de la variable, donc nous pouvons l’utiliser aussi souvent que nécessaire sans craindre la perte de valeurs.

```
écrire(variable)
écrire(variable1, variable2, variable3, …)
```

Algorithmique - les données : exemple complet

Notation

```
Déclaration :
        variable : nom : chaîne de caractères
Début
        écrire(“Quel est votre nom”)
        lire(nom)
        nom = “Martin”
        ecrire(“Bonjour”)
        ecrire(nom)
Fin.
```

Algorithmique - les opérateurs de comparaison

`==` : teste l'**égalité des valeurs** (conversion de type possible).
`===` : teste l'**égalité stricte** → la valeur **et** le type doivent être identiques.
`!=` : teste la **différence de valeurs** (conversion de type possible).
`!==` : teste la **différence stricte** → soit la valeur, soit le type (ou les deux) sont différents.
`>` : signifie **strictement supérieur à**.
`>=` : signifie **supérieur ou égal à**.
`<` : signifie **strictement inférieur à**.
`<=` : signifie **inférieur ou égal à**.

Algorithmique - les structures conditionnelles

Structure conditionnelle : SI…ALORS

Jusqu'à présent nos algorithmes n'intégraient aucune condition, ils se contentaient de donner un résultat quels que soient les données de départ ou les données saisies par l'utilisateur.

Cependant, il est très fréquent de souhaiter que l'algorithme se comporte différemment en fonction des données fournies.

```
Si (condition) alors
    action  {actions à exécuter si condition remplie}
Fsi
```

(Fsi = Fin si)

Algorithmique - les structures conditionnelles

Structure conditionnelle : SI…ALORS

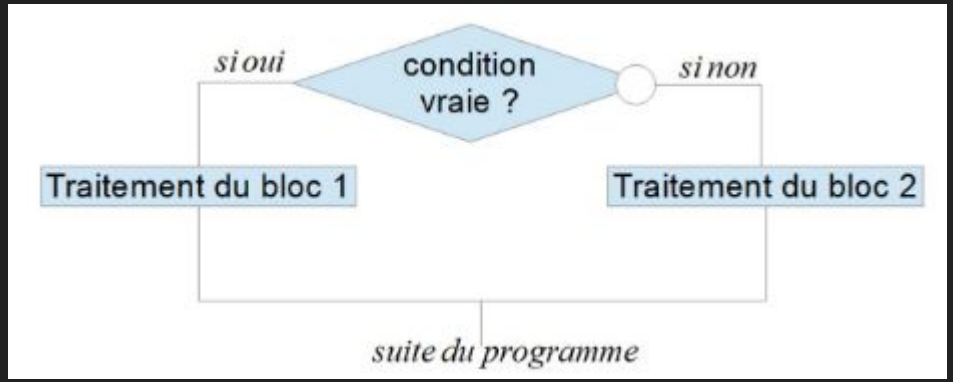
Imaginons un algorithme qui écrit “solde négatif” si le solde bancaire est inférieur à 0.

```
Algorithme : Solde négatif
Déclaration :
        Variable : solde : entier
lire(solde) {l'utilisateur saisit le solde bancaire)
Si solde < 0 alors
   écrire(“Solde négatif”)
Fsi
```

Algorithmique - les structures conditionnelles

Structure conditionnelle : SI…ALORS…SINON

L'algorithme précédent prévoit des actions à exécuter uniquement si la condition est remplie, en réalité il faudrait également prévoir une ou plusieurs actions le cas contraire.



Algorithmique - les structures conditionnelles

Structure conditionnelle : SI…ALORS…SINON

Si (**condition**) **alors**
    actions {actions à exécuter si condition remplie, sinon actions ignorées}
Sinon
    actions {actions à exécuter dans le cas contraire)
Fsi

Algorithmique - les structures conditionnelles

Structure conditionnelle : CAS PARMI

L'algorithme sur le solde et celui sur les âges acceptent un grand nombre de valeurs différentes, néanmoins dans certaines hypothèses il existe un certain nombre limité de valeurs connues. Pour ces situations la structure **cas parmi** est parfaitement adaptée :

```
Cas Expression Parmi
      valeur1 : actions
      valeur2 : actions
      valeur3 : actions
      etc…
      defaut: actions (optionnel)
Fcas
```

Algorithmique - les structures conditionnelles

Structure conditionnelle : CAS PARMI

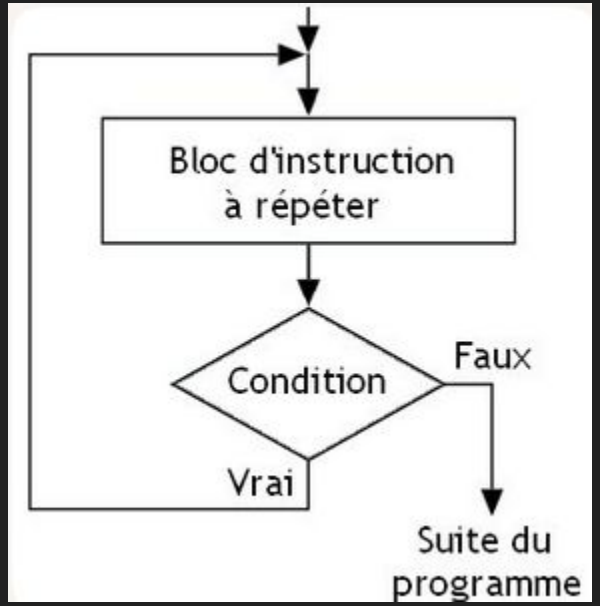
Exemple :

```
Algorithme : navigateurs Web
Déclaration :
        Variable : navigateur : caractère (chaîne)
Début
        lire(navigateur)
        Cas navigateur Parmi
                "Edge"      : écrire("Microsoft")
                "Chrome" : écrire("Google")
                "Firefox"   : écrire("Mozilla")
                defaut : écrire("Inconnu")
        Fcas
Fin
```

Algorithmique - les structures itératives

Structure itérative :

Structure itérative signifie plusieurs itérations. Une itération est une répétition d'un ensemble d'instructions).



Algorithmique - les structures itératives

Structure itérative POUR

La structure précédente (cas parmi) est très adaptée lorsque les données entrantes sont limitées. Cependant, quant est-il lorsqu'il s'agit par exemple d'afficher un grand nombre d'informations ou lorsqu'il s'agit de répéter plusieurs fois les mêmes actions ?

Imaginons que nous souhaitons créer un algorithme écrivant les 10 premiers chiffres, au stade actuel de nos connaissances nous procéderions de la façon suivante :

```
Algorithme : 10 chiffres
Début
        écrire(0)
        écrire(1)
        écrire(2)
        ……….
        écrire(9)
Fin
```

Algorithmique - les structures itératives

Structure itérative POUR

Nous observons ici une répétition de la l'action écrire et seul le chiffre augmente de 1 (incrémentation).
Il s'agit donc d'une itération (répétition) avec incrémentation.

Lorsque la même action se répète il s'agit d'une boucle, et si nous connaissons le nombre d'itérations, la boucle **pour** est parfaitement adaptée.

```
Algorithme : 10 chiffres
Déclaration :
        Variable : i : entier
Début
        Pour i de 0 à 9 par +1 faire
                écrire(i)
        Fpour
Fin
```

Algorithmique - les structures itératives

Structure itérative TANT QUE

La **boucle pour** est adéquate dans toutes les situations où le **nombre d'itération est connu**, pourtant il existe de nombreuses situations pour lesquelles le nombre d'itérations est difficilement déterminé ou déterminable.

Prenons un exemple simple : la liste de course. Nous avons un budget à ne pas dépasser et des achats à faire tant que ce budget n'est pas atteint. Le nombre d'itérations dépendra du prix de chaque article.
La **boucle tant que** permettrait d'ajouter des articles tant que le budget n'est pas atteint

```
tant que (condition)
faire
        actions
Ftant
```

Algorithmique - les structures itératives

Structure itérative TANT QUE

```
Algorithme : courses
Déclaration :
Variable : total, prix_article, budget : entier
Début
        total = 0 (initialisation)
        tant que (total < budget)
        faire
                ecrire(“Votre budget restant est de : “)
                ecrire(budget - total)
                ecrire(“Quel est le montant de l’article à ajouter ?”)
                lire(prix_article)
                total = total + prix_article
        Ftant
Fin
```

Ceci est un exemple, en pratique il faudrait vérifier si total + prix_article <= budget avant d’ajouter l’article.

Algorithmique - les structures itératives

Structure itérative REPETER…JUSQU'A

La boucle répéter…jusqu'à exécute au moins une fois le bloc d'instructions, puis teste la condition de fin.

```
Algorithme : courses
Déclaration :
        Variables : total, prix_article, budget : entier
Début
        total ← 0   (initialisation)
        Répéter
                ecrire("Votre budget restant est de : ")
                ecrire(budget - total)
                ecrire("Quel est le montant de l'article à ajouter ?")
                lire(prix_article)
                total ← total + prix_article
        Jusqu'à (total ≥ budget)
Fin
```