Matière : Programmation informatique

Responsable de la matière : ELAGGOUNE Zakarya

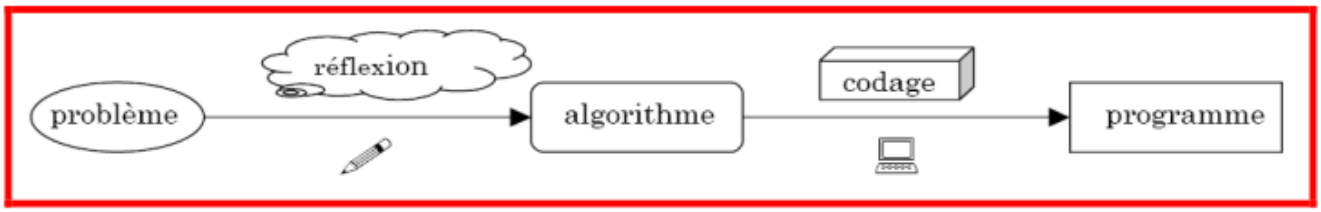
Cours / TD 01 : Découvrir l'algorithme

- **Algorithme :** est un terme d'origine arabe (vient du nom du célèbre mathématicien arabe Al Khawarizmi), composé d'une suite de raisonnement et des instructions élémentaires, qui en une fois exécutée correctement conduit à une résultats de certains problèmes.

- **Programme :** est un assemblage et enchaînement d'instruction élémentaire écrits dans un langage de programmation et exécuté par un ordinateur afin de traiter les données d'un problème et revoyer des résultats.

- **Langage de programmation :** fournit un ensemble de mots-clés et de régles de syntaxe qui peuvent s'exécuter sans soucis, sur une machine. Il existe plusieur langage de programmation tel que : C,C++, Python, Java, Php... etc. (Ps : le langage Python est utilisé dans ce TD).

- **Pourquoi apprendre l'algorithmique pour apprendre à programmer ?** Parce que 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. Cette dimension est présente quelle que soit le langage de programmation ; mais lorsqu'on programme dans un langage on doit en plus se colleter les problèmes de syntaxe, ou de types d'instructions, propres à ce langage. Apprendre l'algorithmique de manière séparée, c'est donc sérier les difficultés pour mieux les vaincre.



- **Critères d'un bon algorithme :** La bonne connaissance de l'algorithmique permet d'écrire des algorithmes exacts et efficaces, or c'est en écrivant des algorithmes corrects, qu'on devient un bon Programmeur.

  Un algorithme correct doit donc subit à certains critères qui sont :

être fini (achevé après un nombre fini d'actions élémentaires), être précis (la machine n'a pas à choisir), et mentionner les entrées (saisie de données) et les sorties (affichage des résultats)

- **Comment représenter un algorithme ?**

(1) En termes de langage naturel.

• Exemple:

* Supposons que le premier élément soit le plus grand. * Examinez chacun des éléments restants dans la liste et, s'il est plus grand que le plus grand élément trouvé jusqu'à présent, notez-le. * Le dernier élément noté sera le plus grand de la liste une fois le processus terminé.

• Avantages

√ Familiarisé

• Inconvénients

√ Verbeux

√ Imprécis → Ambiguïté

√ Rarement utilisé pour les algorithmes complexes ou techniques

(2) En termes de langage de programmation formel

• Exemple:

```
number = int(input("Saisissez un nombre: "));
if (number % 2 == 0) {
   print("Le nombre est pair") ;
else
   print("Le nombre est impair") ; }
```

• Avantages

√ Précis → Sans ambiguïté

√ Programmera à terme dans ces langages

• Inconvénients

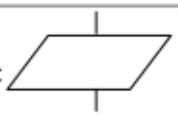
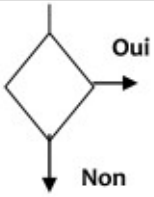
√ Difficile : vous devez d'abord maîtriser le langage de programmation

√ Comporte des détails syntaxiques qui ne sont pas importants lors de la conception d'algorithmes.

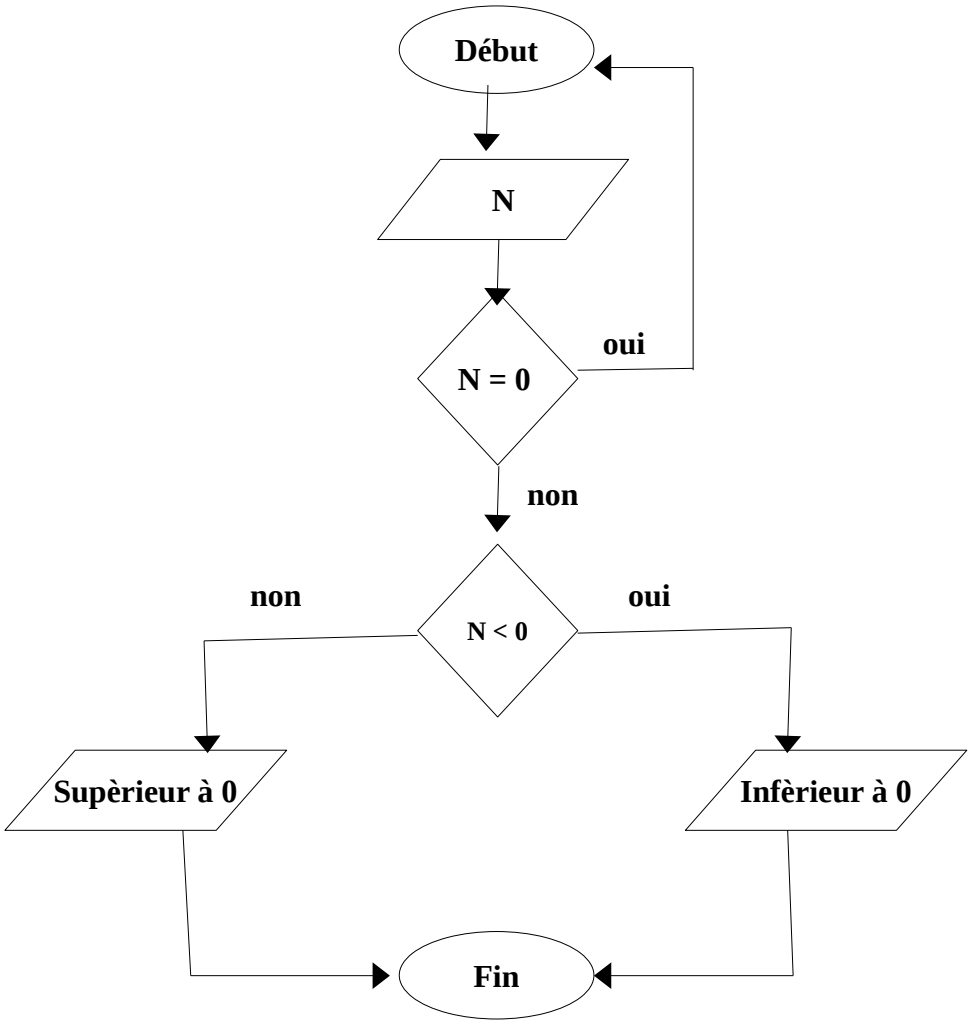
√ Inconnu pour la personne qui ne s'intéresse pas à ce langage de programmation.

(3) Représentation en organigramme

Un organigramme est une représentation graphique d'un algorithme, il permet de schématiser graphiquement la solution d'un problème. Un organigramme permet de mieux visualiser la démarche de résolution d'un problème, il est construit à partir d'un formalisme comprenant cinq simples symboles normalisés qui sont reliés entre eux par des lignes de liaisons. Ces symboles sont :

SYMBOLE	***DESIGNATION***
L'ovale : 	Il exprime le début ou la fin de l'organigramme.
Le parallélogramme : 	Il est utilisé pour les opérations d'entrée / sortie (Instruction ou groupe d'instructions pour lire les données ou écrire les résultats)
Le rectangle : 	Il est utilisé pour les opérations ou groupe d'opérations de traitement (calcul) sur des données.
Le losange : 	Il est utilisé pour la vérification d'une condition (un test). Instruction conditionnelle. la condition est évaluée pour pouvoir prendre le chemin correspondant
Cercle de conjonction : 	Il est utilisé pour la liaison de plusieurs critères

• *Exemple:* Ecrire un organigramme qui lit un nombre N non nul et affiche le message: inférieure à "0" ou supérieur à "0" suivant sa valeur.



• *Avantages*

✓ Logique claire, analyse efficace

✓ Programmation et test simplifiés

• *Inconvénients*

✓ Difficiles à utiliser pour les grands programmes

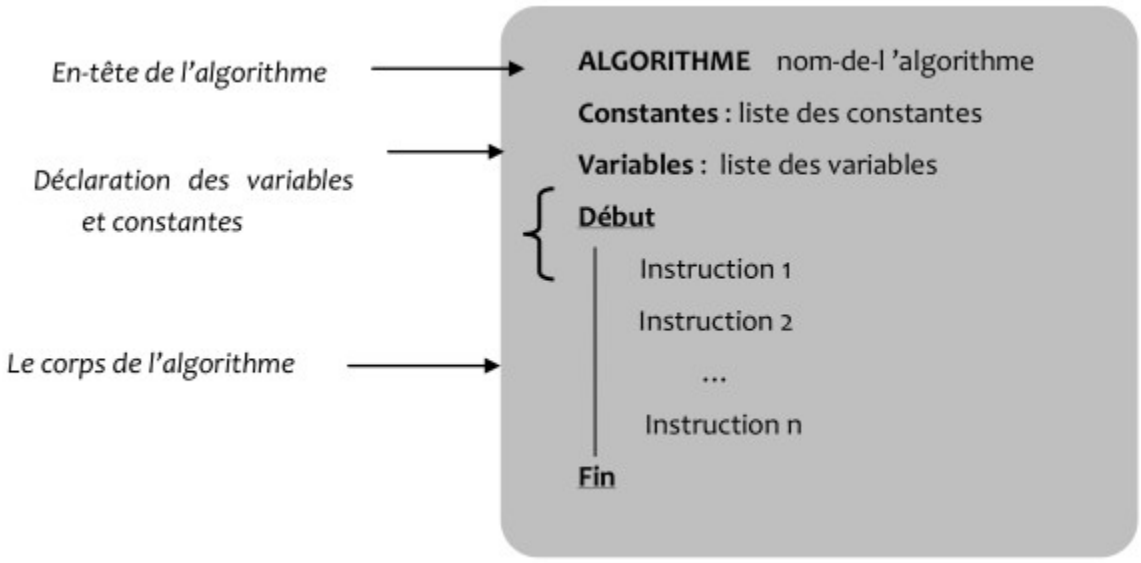
✓ Difficiles à modifier

(4) En termes de pseudocode (pseudo-langage de programmation)

Le pseudocode (dérivé de pseudo et code) est une description compacte et informelle de haut niveau d'un algorithme de programmation informatique qui utilise les conventions structurelles d'un langage de programmation, mais omet généralement les détails qui ne sont pas essentiels à la compréhension de l'algorithme, tels que les sous-routines, les déclarations de variables et le code spécifique au système.

Un algorithme s'écrit le plus souvent en pseudocode afin de faciliter ultérieurement sa traduction dans un langage de programmation.

• *Structure d'un pseudocode*



L'en-tête : permet tout simplement d'identifier l'algorithme.

Déclaration : liste de toutes les constantes et variables utilisées dans l'algorithme.

Le corps : cette partie contient les ordres (instructions) ou les tâches de l'algorithme.

• Quelques remarques

✓ Il n'existe pas de syntaxe standard pour le pseudocode.

✓ Un programme en pseudocode n'est pas un programme exécutable.

• *Avantages*

✓ Un compromis acceptable.

✓ Ressemble à de nombreux langages de programmation populaires.

✓ Relativement peu contraignant en termes de règles grammaticales.

✓ Seules les instructions bien définies sont incluses : « Un langage de programmation sans détails ».

• *Inconvénients*

✓ Comparé à un organigramme, il est plus difficile de comprendre la logique du programme.

amalgame de concepts

→ Les Mots-clés du Pseudo-code

Le pseudo-code n'est pas un langage de programmation rigide, mais il suit une structure fixe pour être lisible par tous. Voici les mots-clés essentiels :

- **Algorithme :** Nom de l'exercice.
- **Variables :** Zone de déclaration (on précise le type : Entier, Réel, Chaîne, Booléen).
- **Début / Fin :** Délimitent le corps de l'algorithme.

Exemple de structure :

```
Algorithme Calcul_Double
Variables
    x, resultat : Entiers
Début
    Écrire "Entrez un nombre :"
    Lire x
    resultat ← x * 2
    Écrire "Le double est : ", resultat
Fin
```

→ Faire tourner l'algorithme "à la main"

C'est une technique de vérification (le "debug" manuel). On crée un tableau avec les variables et on suit l'algorithme ligne par ligne pour voir comment les valeurs évoluent.

Exemple :

```
1. A ← 5
2. B ← A + 2
3. A ← A * 2
```

Étape	Valeur de A	Valeur de B
Ligne 1	5	/
Ligne 2	5	7
Ligne 3	10	7

→ La Récursivité

La récursivité, c'est quand une fonction s'appelle elle-même pour résoudre un problème. Un algorithme récursif doit toujours avoir une condition d'arrêt pour éviter de boucler à l'infini.

Exemple : Le calcul de la factorielle (n!)

- $3! = 3 \times 2 \times 1 = 6$

```
Fonction Factorielle(n)
    Si n == 0 Alors
        Retourner 1  // Condition d'arrêt
    Sinon
        Retourner n * Factorielle(n - 1) // Appel récursif
    FinSi
FinFonction
```

· Un algorithme possède à quatre types d'instruction considérercomme des briques de base pour résoudre un problème :

1 / L'affectation des variables

L'affectation est l'opération qui consiste à attribuer une valeur à une variable. On utilise généralement le symbole ←.

- Syntaxe : `Nom_Variable ← Valeur`
- Principe : La valeur à droite est stockée dans la "boîte" à gauche. Si la variable contenait déjà quelque chose, l'ancienne valeur est écrasée.

Exemples :

- `Âge ← 25` (On stocke le nombre 25 dans la variable Âge)
- `Nom ← "Alice"` (On stocke une chaîne de caractères)
- `Nb ← Nb + 1` (On augmente la valeur actuelle de Nb de 1)

2 / La Lecture et l'Écriture

Pour communiquer avec l'utilisateur, l'algorithme utilise deux instructions fondamentales :

- **LIRE :** Permet à l'utilisateur d'entrer une donnée via le clavier.
- **ÉCRIRE :** Permet d'afficher un message ou un résultat sur l'écran.

Exemple :

```
Écrire "Quel est votre nom ?"
Lire utilisateur
Écrire "Bonjour ", utilisateur
```

3 / Les structures conditionnelles (tests) : La Condition : Si... Alors... Sinon... FinSi

C'est la structure de décision. Elle permet d'exécuter un bloc de code uniquement si une condition est vraie.

Exemple 01 : Vérifier la majorité

```
Si âge >= 18 Alors
    Écrire "Accès autorisé"
Sinon
    Écrire "Accès refusé"
FinSi
```

Exemple 02 : Afficher la moyenne

```
Algorithme CalculMention
Variables
    note1, note2, moyenne : Réel
Début
    Afficher "Entrez la première note :"
    Saisir note1
    Afficher "Entrez la deuxième note :"
    Saisir note2

    moyenne ← (note1 + note2) / 2

    Si moyenne >= 10 Alors
        Afficher "Reçu avec une moyenne de ", moyenne
    Sinon
        Afficher "Ajourné avec une moyenne de ", moyenne
    FinSi
Fin
```

4 / Les Boucles (Structures Répétitives)

A. Boucle "Tant que... Faire"

On utilise cette boucle quand on ne sait pas à l'avance combien de fois on va répéter l'action. Elle continue **tant que** la condition est vraie.

Exemple : Demander un mot de passe jusqu'à ce qu'il soit correct

```
Tant que mdp != "1234" Faire
    Écrire "Mot de passe incorrect, réessayez :"
    Lire mdp
FinTantQue
```

B. Boucle "Pour... Faire"

On l'utilise quand on connaît exactement le nombre de répétitions.

Exemple : Afficher "Bonjour" 5 fois

```
Pour i allant de 1 à 5 Faire
    Écrire "Bonjour"
FinPour
```