Eléments de base de l'algorithmique

I. Généralités sur l'algorithmique et la programmation

I.1 Qu'est-ce que l'Algorithmique ?

I.1.1 L'origine du mot ''algorithme''

Le mot algorithme provient du nom d'un célèbre mathématicien, géographe, astronome,… musulman perse du IXème siècle appelé Mohammed Ibn Musa Al Khawarizmi.

I.1.2 Exemples d'algorithme

1. Une recette de cuisine.
2. Un mode d'emploi.
3. Une stratégie gagnante pour un jeu simple.
4. Une méthode systématique pour résoudre un problème mathématique, par exemple la résolution d'un système d'équations.

I.1.3 Définition d'un algorithme

C'est une suite finie d'instructions, qu'on applique à un nombre fini de données, dans un ordre bien déterminé pour résoudre un problème donné.

Autre définition

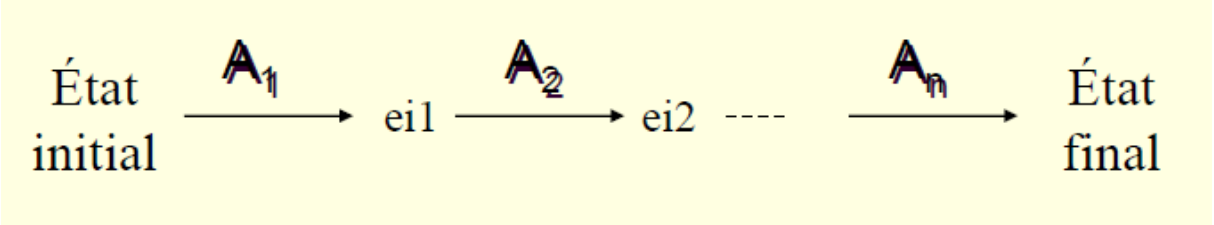
Un algorithme est la description de la méthode de résolution d'un problème posé.

L'algorithmique : L'algorithmique est la science qui étudie l'application des algorithmes à l'informatique.

I.2 Construction d'un algorithme

I.2.1 Définition

Construire un algorithme consiste à concevoir les actions qu'il faut organiser dans le temps, et à choisir la manière de les organiser pour obtenir le résultat escompté par leurs effets cumulés.



I.2.2 Étapes de construction d'un algorithme

• Position du problème et précision des conditions de travail: Objets caractérisant les informations ou les données à manipuler.
• Réfléchir à une méthode bien définie pour résoudre le problème.
• Décrire cette méthode sans ambiguïté.
• Évaluer la méthode décrite (temps d'exécution, nombre d'opérations, taille, etc..).
• Réfléchir aux qualités et aux défauts de la méthode choisie.
• Réfléchir à d'autres méthodes et les comparer selon les critères de qualité définis.

I.3 Algorithmique & programmation

En informatique, un programme est la traduction d'un algorithme dans un certain langage de programmation.

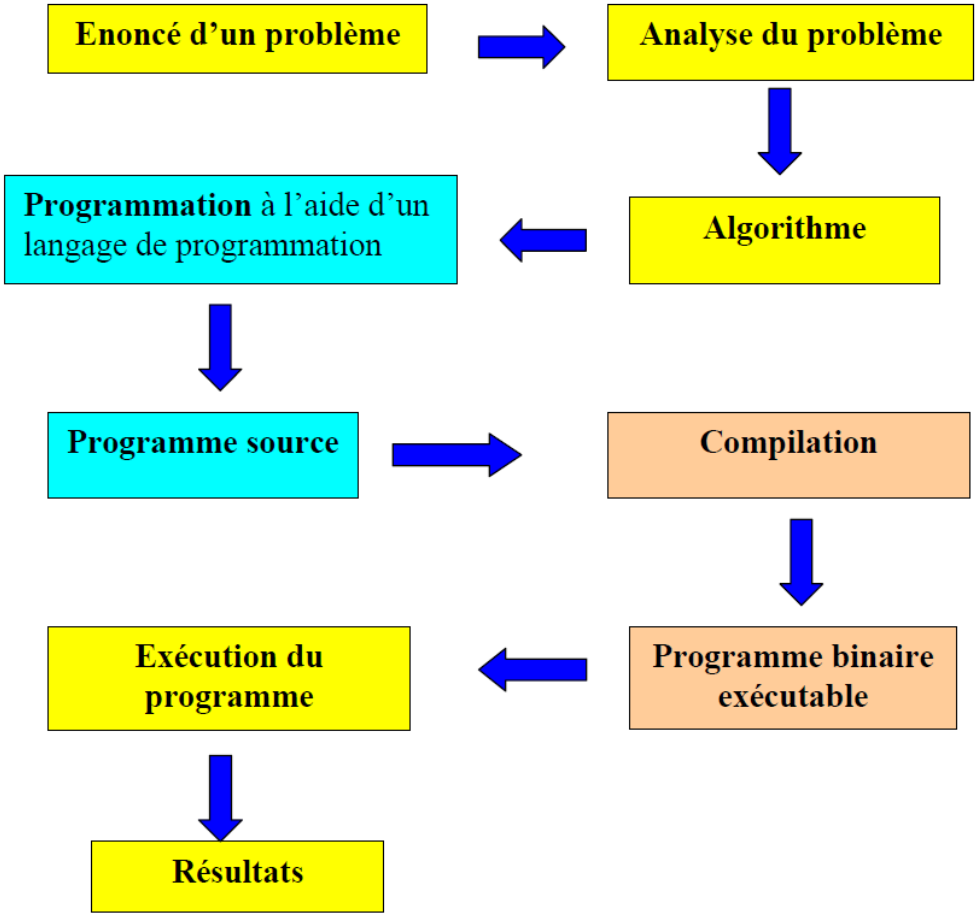
I.3.1 Objectifs principaux de la programmation

- Utiliser l'ordinateur pour traiter des données afin d'obtenir des résultats.
- La programmation joue le rôle d'intermédiaire entre le langage machine et le langage humain.

I.3.2 Démarche de la programmation

La résolution d'un problème passe généralement par les étapes suivantes :

• Phase d'analyse et de réflexion (algorithmique).
• Phase de programmation.
  - Choisir un langage de programmation.
  - Traduction de l'algorithme en programme.
  - Programme (ou code) source.
  - Compilation : traduction du code source en code objet.
  - Traduction du code objet en code machine exécutable et compréhensible par l'ordinateur.
• Phase de test.
• Phase d'exécution.



II. Éléments de base de l'algorithmique

II.1 Constantes

Définition : Les constantes sont des quantités fixées et invariantes au cours de l'algorithme.

• Déclaration d'une constante : Instruction permettant de réserver de l'espace mémoire pour stocker des données dont la valeur est fixée pour tout l'algorithme.

• Syntaxe (Règles d'écriture d'un langage de programmation informatique):

Constantes <identificateurs>=<expression>

Exemple :

constantes MAX=100

e = 2.72

II.2 Notion de variable

Dans un algorithme (ou programme informatique), le stockage des informations est primordial. Ces informations sont essentiellement :

- Les données fournies par l'utilisateur (les entrées);
- Les résultats obtenus par l'algorithme (les sorties);
- Les données intermédiaires.

Ces données sont de plusieurs types :

- des nombres (12, 5/9, 3.14, …);
- du texte ('a', 'Z', 'Bonjour', 'programmation',…)

Conséquence : Le stockage d'information nécessite l'utilisation de ce qu'on appelle variable.

II.2.1 Définitions

Définition d'une variable :

Une variable est un emplacement mémoire nommé, de taille fixée prenant au cours du déroulement de l'algorithme un nombre indéfini de valeurs différentes.

Identificateur : C'est le nom donné à une variable. Ce nom doit obligatoirement commencer par une lettre suivie d'une suite de lettre et de chiffre et il ne doit pas contenir de caractère accentuer ni d'espace.

Déclaration d'une variable :

C'est l'instruction permettant de réserver de l'espace mémoire pour stocker des données de différents types : entiers, réels, caractères, chaînes de caractères,…).

Syntaxe : variable<liste d'identificateurs> : type

Exemples :

- Variables val, unNombre: entiers
- nom, prénom : chaînes de caractères

Remarque :

• Une variable ne peut être utilisée que s'elle est déclarée.

• Le nom d'une variable doit être formé par des lettres alphabétiques, des chiffres et le caractère souligné (_).

• Le nom choisi doit être simple et reflète l'objet auquel il est associé. Ainsi, pour représenter le salaire d'un employé dans un algorithme, il vaut mieux choisir l'identificateur salaire au lieu de x ou y.

II.2.2 Types de variable

Le type d'une variable est l'ensemble des valeurs qu'elle peut prendre. Par exemple, une variable de type entier peut prendre entiers positifs ou négatifs.

Les différents types utilisés :

• **Type Entier :** pour manipuler les nombres entiers positifs ou négatifs. Par exemple : 25, -15, etc.

• **Type Réel :** pour manipuler les nombres à virgule. Par exemple : 3.14, -15.5, etc.

• **Type Caractère :** pour manipuler des caractères alphabétiques et numériques. Par exemple : 'a', 'A', 'z', ' ?', '1', '2' etc.

• **Type Chaîne :** pour manipuler des chaînes de caractères permettant de représenter des mots ou des phrases.

Exemple : "bonjour, Monsieur" (une chaîne de caractère est toujours notée entre guillemet.
Remarque : En pseudo-code, une chaîne de caractères est toujours notée entre guillemets " ", car, il peut y avoir une confusion entre des nombres et des suites de chiffres. Par exemple, 423 peut représenter :

- le nombre 423 (quatre cent vingt-trois),
- ou la suite de caractères 4, 2, et 3 notée : "423"

• **Type Booléen :** pour les expressions logiques. Il n'y a que deux valeurs booléennes : vrai et faux.

Exemple :

```
Variables   n : entier
            r : réel
            x: caractere
            nom_etudiant : chaîne de caractere
            a, b: booléens
```

II.3 Primitives

II.3.1 Opérateurs

Un opérateur est un symbole significatif qui relie deux objets (variables ou constantes), pour produire un résultat. Les opérateurs sont classés comme suit :

• **Opérateurs arithmétiques** : Ce sont des opérateurs qui opèrent sur les objets numériques (+, -, *, /, div, mod). On note que ‘’div’’ est la division entière qui donne la partie entière du quotient d’une division, et le mod (le modulo) donne le reste d’une division entière.

• **Opérateurs Logiques :** Ce sont des opérateurs qui opèrent sur les booléens (ET, OU, NON,…).

• **Opérateurs de comparaison :** Ce sont des opérateurs qui opèrent sur les types numériques (=, < >, <, >, <=, >=).

Type	Opérations possibles	Symbole ou mot clé correspondant
Entier	Addition	+
	Soustraction	-
	Multiplication	*
	Division	/
	Division Entière	DIV
	Modulo	MOD
	Exposant (puissance)	^
	Comparaisons	<, =, >, <=, >=, <>
Réel	Addition	+
	Soustraction	-
	Multiplication	*
	Division	/
	Exposant	^
	Comparaisons	<, =, >, <=, >=, <>
Caractère	Comparaisons	<, =, >, <=, >=, <>
Chaîne	Concaténation	+ , &
Booléen	Comparaison	<, =, >, <=, >=, <>
	Logiques	ET, OU, NON

Exemples :

5 / 2 = 2.5; 5 Div 2 = 2; 5 Mod 2 = 1; 5 ^ 2 = 25

"Bonjour" & " " & "Monsieur" donne "Bonjour Monsieur"

II.3.2 Instruction, expression, affectation, incrémentation

• **Instruction :** Elle traduit une ou plusieurs consignes (ou actions) portant sur une ou plusieurs variables.

• **Expression :** C'est une combinaison de variables, de constantes et d'opérateurs. Le type d'une expression est défini par celui des variables et des constantes qui la constituent.

Exemples : (x+y)*(x-y)/2; a div b; (x>=0) ET (y>=0)

• **Affectation :** C'est une instruction qui consiste à doter une variable d'une valeur appartenant à son domaine, c'est-à-dire à lui donner une première valeur ou à changer sa valeur courante. Elle se réalise au moyen de l'opérateur ⟵

Syntaxe :

Nom_variable ← Valeur

Nom_variable ← Nom_variable

Nom_variable ← Expression

Exemple :

x ← 4 signifie:

- mettre la valeur 4 dans la case mémoire identifiée par x (on dit aussi x reçoit 4).
- copier dans x la valeur 4.

Remarques :

• 4 ← x est une instruction fausse.

• x ← y x et y doivent être de même type.

• x+1 ← y est une instruction fausse car une variable doit toujours être à gauche de l'affectation.

Exemple d'algorithme :

```
Test /* nom de l'algorithme*/
 début
        x ← 12
        y ← x + 4
        x ← 3
 fin
```

• **Incrémentation :** C'est l'augmentation de la valeur d'une variable à chaque phase de l'exécution d'un programme (algorithme).

• **Décrémentation :** C'est la diminution de la valeur d'une variable à chaque phase de l'exécution d'un programme (algorithme).

Syntaxe : x ← x+step où ''step'' est le pas d'incrémentation (ou de décrémentation).

Exercice : Écrire un algorithme permettant d'échanger le contenu de deux variables.

Solution :

```
Algorithme Échange
        Variables x, y, z: entiers
Début
        lire(x,y)
                z ← x
                x ← y
                y ← z
        ecrire(x, y)
Fin.
```

II.4 Instructions d'entrée/sorties (lecture/écriture)

II.4.1 Lecture (saisie)

L'instruction **''lire''** permet de lire une ou plusieurs variables : l'utilisateur doit saisir une ou plusieurs valeurs au clavier et valider par la touche ''Entrée''.

Syntaxe : lire(variable1, variable2,…)

Exemples :

lire (Nom) ; lire(A); lire(X, Y)

Recommandation : Il est souhaitable que toute instruction de lecture d'une variable sera précédée par l'affichage d'un message qui invite l'utilisateur à saisir une variable.

II.4.2 L'écriture (sortie)

L'instruction **''Écrire (ou afficher)''** permet d'écrire à l'écran (console) les valeurs des variables et les messages.

Syntaxe : Écrire (variable1, variable2,…)
Écrire ('message')

Exemples :

Écrire ('Bonjour, la séance des TDs commence à 08h.') Cette instruction affichera à l'écran le message : Bonjour, la séance des TDs commence à 08h.

Écrire(X) Cette instruction affichera la valeur de la variable X.

II.5 Structures de contrôle

En général, trois structures de contrôle sont à distinguer :

- Structure séquentielle,
- Sélection (Choix ou test),
- Boucles (ou itérations).

II.5.1 Structure Séquentielle

Dans un algorithme, les instructions sont exécutées séquentiellement c'est-à-dire l'une après l'autre.

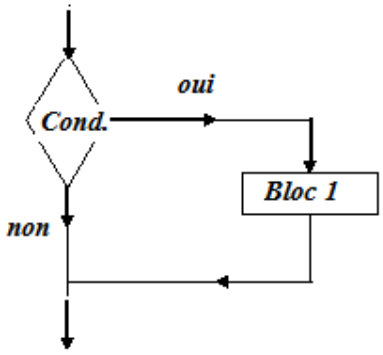
Un bloc est une suite d'instructions délimitée par ''début'' et ''fin'', qui s'exécutent séquentiellement.

II.5.2 Sélection (instruction alternative)

L'instruction Si : Cette instruction peut être représentée dans un algorithme sous deux formes:

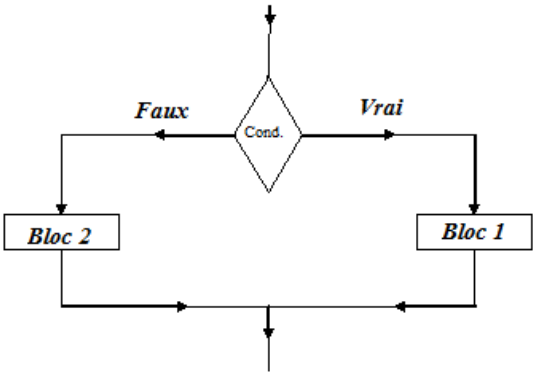
• Syntaxe 1 :

Si (Condition) alors
    Bloc 1
finSi.



• **Syntaxe 2 :**

Si (Condition) alors
    Bloc 1
Sinon
    Bloc 2
finSi.



Remarque : Si la condition est vraie le bloc 1 sera exécuté, sinon on passe à l'exécution du bloc 2.

Exemple 2 : Calcul de la valeur absolue d'un nombre réel x.

```
Algorithme Val_Abs_x
    Variable x, valabsx: réel
début
    ecrire('Donner la valeur de x')
    lire(x)
    Si (x>0) alors
        valabsx ← x
    Sinon
        Valabsx ← -x
    finsi
fin
```

II.5.3 Les boucles

II.5.3.1 Intérêt des boucles

Problème : Dans un algorithme, il arrive souvent que la même action soit répétée plusieurs fois. Ainsi, il est ennuyant d'écrire un algorithme qui contient de nombreuses fois la même instruction.

Solution : Pour pallier à ce problème, on fait appel à des instructions en boucle qui ont pour effet de répéter plusieurs fois une même action.

On distingue deux formes de boucles :

1. Répétitions inconditionnelles où le nombre de répétitions est connu avant l'exécution de l'algorithme,

2. Répétitions conditionnelles où le nombre de répétitions est inconnu avant l'exécution de l'algorithme.

Itération : C'est l'exécution de la liste des instructions dans une boucle.

II.5.3.2 Répétitions inconditionnelles : la boucle ''Pour''

Cette boucle est utilisée surtout si le nombre d'itérations à effectuer est connu au préalable.

• Syntaxe :

```
pour variable ← valeur initiale à variable ← valeur finale faire
        liste d'instructions
finpour
```

La variable va prendre successivement toutes les valeurs entières entre la valeur initiale et la valeur finale.
Pour chaque valeur prise par la variable, la liste des instructions est exécutée.
La variable utilisée pour énumérer les itérations est appelée variable d'itération, indice d'itération ou compteur.
L'incrémentation par 1 de la variable est implicite.

Exercice : Effectuer la somme de 0 à n (nombres entiers naturels).

Solution :

```
Algorithme Somme                        /* Nom de l'algorithme*/
Variable S, n, i: entiers               /* Déclaration des variables */
Début
    Lire(n)                             /* Lecture de la valeur de n */
    S← 0                                /* Initialisation de la somme */
    Pour i← 0 à i← n Faire              /* Début de la boucle */
        S← S+i                          /* Exécution de l'instruction */
    finPour                             /* Fin de la boucle */
    Ecrire(S)
Fin.
```

II. 5.3.3 Répétitions conditionnelles

a. La boucle ''Tant que...Faire''

Quand il est à priori impossible de connaître à l'avance au bout de combien d'itérations une condition cessera d'être satisfaite, on utilise Tant que…Faire
Syntaxe :

```
Tantque (condition) Faire
        Liste d'instructions
FinTant
```

La liste d'instructions est répétée autant de fois que la condition est vraie.

Exercice : Effectuer la somme des n premiers nombres entiers naturels.
Solution :

```
Algorithme Somme                        /* Nom de l'algorithme*/
Variable S, n, i: entiers               /* Déclaration des variables */
Début
    Lire(n)                             /* Lecture de la valeur de n */
    i ← 0                               /* Initialisation du compteur */
    S ← 0                               /* Initialisation de la somme*/
    Tant que (i<=n) Faire               /* Début de la boucle */
        S ← S+i                         /* Exécution de l'instruction */
        i← i+1                          /* Incrémentation */
    FinTant                             /* Fin de la boucle */
    Ecrire(S)
Fin
```

b. La boucle ‘’Répéter...Tant que’’

La syntaxe de cette boucle est donné par :

```
Répéter
        liste d’instructions
tantque (condition)
```

La liste d’instructions est tout d’abord exécutée, ensuite on examine la condition. Si elle est fausse, on refait le bloc d’instructions puis on réexamine la condition et ainsi de suite. Si la condition est vérifiée (vraie), on sort de la boucle.

Exemple : Saisir au clavier un nombre n compris entre 0 et 10.

```
Répéter
        ecrire(‘Donner un nombre n compris entre 0 et 10’)
        lire(n)
Tant que ((n<0) OU (n>10))
```

II. 5.3.4 Boucles imbriquées

On dit qu’une boucle est imbriquée s’elle contient d’autres boucles.

Exemple :

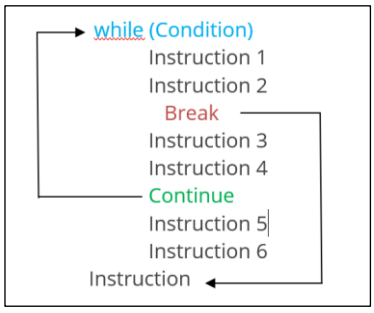
```
Variables i, j : entiers
Début
        Pour i← 1 à i← 10 Faire
                ecrire("Première boucle")
                Pour j ← 1 à j← 6 Faire
                        ecrire("Deuxième boucle") ;
                Finpour(j) ;
        Finpour(i) ;
Fin.
```

II. 5.3.5 Les instructions break et continue

Le principe des boucles est de parcourir un bloc de code jusqu’à ce que la condition soit fausse, mais parfois nous souhaitons :

- Mettre fin à l’itération en cours
- Mettre fin à la totalité de la boucle sans vérifier la condition.

Les instructions break et continue sont utilisées dans ces deux cas (voir figure ci-dessus).



L’instruction break est utilisé pour quitter une boucle while/for, alors que continue est utilisé pour ignorer le bloc actuel et revenir à l’instruction while/for.

Exemple avec la boucle for

```
for i in range(0,10):
    print(i, end='  ')
print("\n Au revoir")
```

L'exécution de ce code donne le résultat suivant :

```
0  1  2  3  4  5  6  7  8  9
 Au revoir
```

Effet de l'instruction continue

```
for i in range(0,10):
    if i == 5:
        continue
    print(i, end='  ')
print("\n Au revoir")
```

L'exécution de ce code donne le résultat suivant :

```
0  1  2  3  4  6  7  8  9
 Au revoir
```

Effet de l'instruction break

```
for i in range(0,10):
    if i == 5:
        break
    print(i, end='  ')
print("\n Au revoir")
```

L'exécution de ce code donne le résultat suivant :

```
0  1  2  3  4
 Au revoir
```

Exemple avec la boucle while

```
i = 10
while i > 0:
    i = i - 1
    print(i, end='  ')
print("\n Au revoir")
```

L'exécution de ce code donne le résultat suivant :

```
9  8  7  6  5  4  3  2  1  0
 Au revoir
```

Effet de l'instruction continue

```
n = int(input("Saisir un nombre : "))
print("les nombres naturels de 1 à ", n)
i=1
while(i<=7):
    if (i == 4):
        break
    print(i, end='  ')
    i=i+1
print("\n Au revoir")
```

```
9  8  7  6  4  3  2  1  0
 Au revoir
```

Effet de l'instruction break

```
i=10
while i > 0:
    i = i - 1
    if i == 5:
        break
    print(i, end='  ')
print("\n Au revoir")
```

L'exécution de ce code donne le résultat suivant :

```
9  8  7  6
 Au revoir
```

II. 5.3.6 Méthodologie pour écrire une boucle

• Si l'action considérée est répétitive, on utilise une boucle.

Pour choisir entre une boucle avec ou sans compteur, on pose la question suivante :

Peut-on prévoir/déterminer le nombre d'itérations ?

1. Si oui, on utilise une boucle avec compteur : la boucle pour
2. Si non, on utilise une boucle sans compteur.

Dans le deuxième cas, on pose la question faut-il tester la condition avant l'exécution ou l'inverse ?

1. si le teste de la condition précède l'exécution, on utilise la boucle TantQue
2. si l'exécution précède le teste de la condition, on utilise la boucle Répéter Tantque

• Écrire l'action répétitive et l'instruction de la boucle choisie.

• Initialiser les variables utilisées (si nécessaires).

• Écrire les conditions d'arrêt, voire l'incrémentation de la variable de contrôle.

• Exécuter pour les cas extrêmes et au moins un cas "normal".

Exercice : Écrire l'algorithme permettant d'imprimer le triangle (triangle de Pascal) ci-dessous, le nombre de lignes étant donné par l'utilisateur :

1
12
123
1234
12345
123456
1234567
……

Solution :

```
Triangle_Pascal
Variables nb_lignes,i,j: Entiers
Début
        ecrire(" Donner le nombre de lignes ")
        lire (nblignes)                                  /* lecture du nombre de lignes */
                Pour i ← 1 à i ← nblignes faire          /* Première boucle */
                        Pour j ← 1 à j ← i faire         /* Deuxième boucle */
                                Ecrire (j)               /* Ecriture de j */
                        Finpour(j)
                        retour à la ligne                /* Retour à la ligne */
                fpour(i)
fin
```

Lexique :

nblignes : entier, nombre de lignes à imprimer,
i : entier, indice d'itération sur les lignes,
j : entier, indice d'itération sur les éléments de la ième ligne.

II.6 Notions de bloc d'instructions et d'indentation

Un bloc de code (bloc d'instructions) est une suite d'instructions (formant un ensemble logique) qui s'exécutent tous successivement dans certaines conditions bien définies. Un bloc peut contenir d'autres blocs imbriqués.

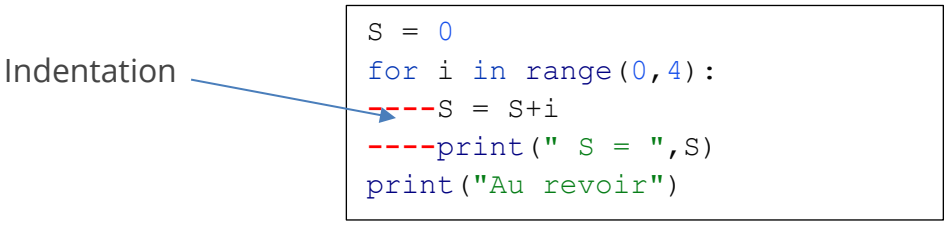
Selon le langage de programmation utilisé, un bloc d'instructions peut être délimité de différentes manières :

En C, C++ et Java il est délimité avec des accolades {} (accolades ouvrante et fermante).

En python on utilise **l'indentation**. Un bloc de code commence par une indentation et se termine par la première ligne non indentée.

Une indentation représente le fait d'ajouter un ou plusieurs espaces au début d'une ligne de code. En python il est recommandé d'indenter son code avec 4 espaces (équivalent à 1 tabulation).

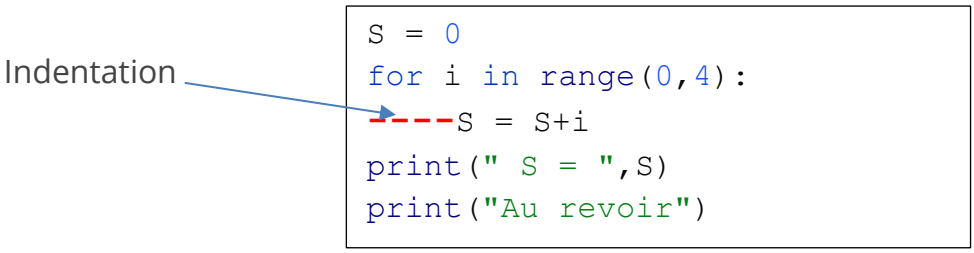
Exemple 1 : soit le code suivant :



L'exécution de ce code donne le résultat suivant :

```
 S  =   0
 S  =   1
 S  =   3
 S  =   6
Au revoir
```

Exemple 2 : soit le code suivant :



L'exécution de ce code donne le résultat suivant :

```
S =  6
Au revoir
```

II.7 Complexité

1- Introduction

Devant plusieurs versions d'algorithme, certaines questions se posent naturellement, en particulier à propos de la manière d'évaluer les performances d'un algorithme. Différents algorithmes ont des goûts différents en termes de :

- Temps d'exécution qui est lié aux nombres d'opérations effectués par l'algorithme.
- La taille mémoire nécessaire pour stocker les différentes structures de données pour l'exécuter.

Ces deux critères sont appelés la complexité de l'algorithme en temps et en espace.
La complexité est un concept fondamental pour tout programmeur, elle permet de mesurer la performance d'un algorithme et déterminer lequel des algorithmes disponibles résolvant le même problème, est meilleur et s'il est plus optimal.
Il existe deux types de complexité :

1. La complexité temporelle qui permet de quantifier la vitesse d'exécution.
2. La complexité spatiale qui permet de quantifier l'utilisation de la mémoire.

2- Quelques classes de complexités classiques

Type de complexité	Nature de complexité	Exemples d'algorithmes
O(1)	Constante	…
O(n)	Linéaire	- Recherche de min/max d'un tableau 1D
O(log(n))	Logarithmique	- Recherche dichotomique
O(nlog(n))	Quasi-linéaire	-Tri rapide -Tri fusion -Tri par Tas
$O(n^2)$	Quadratique	-Tri à bulle - Tri par insertion -Tri par sélection -Tableau à 2D : transposé d'une matrice
$O(n^3)$	Cubique	…
$O(n^\alpha)$	Polynomial	…
$O(x^n)$, n>1	Exponentiel	- Calcul de x^n

Remarque : L'ordre de grandeur asymptotique notée O (grand o) relève de la définition mathématique suivante :
Définition : Une fonction f(n) est en O(g(n)) (grand o de g(n)) si :

$$[\exists n_0, \exists c \in IR \text{ tel que : } \forall\, n \in IR+, n \geq n_0 \Rightarrow f(n) \leq cg(n)]$$

Exemple :

$f(n)=2 = O(1)$
$f(n)=3n+1 = O(n)$
$f(n)= 3n^2 + n+1 = O(n^2)$

1- Exemples d'algorithmes et complexité

a) **Algorithme de HÖrner : L**'Objectif est d'évaluer un polynôme de degré n en un point x de IR. Pour $p(x)=a_0+a_1x+\ldots+a_nx^n$ on pose $T[i]=a_i$ et on écrit la fonction qui calcule et renvoie la valeur du polynôme p au point x :

```
Fonction Horner(p : réel[0..max] ;n :entier ;x :réel) : réel
Variable res : réel
         i : entier
Début
   res←0
   Pour i←n à 0 [Pas=-1] faire
      res←res*x + T[i]
   Finpour
   retourner (res)
Fin
```

Dans l'analyse de cet algorithme, on :

- (n+2) affectation
- (n+1) addition
- (n+1) multiplication

La complexité est donc en O(n)

b)

Pour i←1 à n faire
  Pour j ← 1 à 2*n + 1 faire
  écrire("Bonjour")
  FinPour
FinPour
	Ici la première boucle fait (n-1) itérations et la deuxième boucle 2n itérations, donc un total de (n-1)*2n.
	Donc de complexité d'ordre $O(n^2)$.

c)

Pour i←1 à 10 faire
  Pour j ← 1 à n faire
  écrire("Bonjour")
  FinPour
FinPour
	O(n). Ici la première boucle fait 10 itérations et la deuxième boucle n itérations, donc un total de 10n. Attention, selon certain calcul cela ferait 9 itérations suivi de n-1 itérations. Heureusement, dans la notation de Landau cela ne change rien donc inutile de chipoter pour le comptage d'itérations !

d)

Pour i←1 à n faire
  Pour j ← i à n faire
  écrire("Bonjour")
  FinPour
FinPour
	$O(n^2)$. Très classique, chaque boucle fait n-1 itérations.

e)

Pour i←1 à n faire
  Pour j ← 1 à 2*i + 1 faire
  écrire("Bonjour")
  FinPour
FinPour
	La première boucle est classique. La deuxième va de 1 à 2i+1, donc si on fait quelques exemples : 3, 5, 7, …, 2n+1 (ici on fait fonctionner les deux boucles en même temps). On a donc la somme des n premiers termes inpairs donc quadratique. $O(n^2)$.

f)

Pour i← 1 à n*n do
pour j ← 1 à i faire
  écrire("Bonjour")
  FinPour
FinPour
	La réflexion est la même que pour l'algorithme b) Ici, nous avons les n premiers termes au carré. O(n4).

g)

Pour i← 0 à m faire
  t ← 1
  TantQue(t < m) faire
  écrire("Bonjour")
  t ← t*2
  FTQ
FinPour
	Pour la première boucle tout va bien, le problème va venir de la deuxième. Ici la valeur de t double à chaque itération et la boucle s'arrête quand t dépasse la valeur de m. On pose k le nombre d'itération réalisée. On s'arrête quand 2^k >m donc quand k > log(m). D'où la complexité est O(m log(m)).

Exercice :

1) Calculer la complexité de l'algorithme de conversion suivant :

```
Fonction fact(n :entier) : entier
Variable h,m,s : entier
Début
  h←T div 3600
  m←(n-3600*h) div 60
  s← n mod 60
  retourner h,m,s
 Fin
```

2) Calculer la complexité de la fonction récursive factorielle de n suivante :

```
Fonction conversion(T : entier) : entier
Début
  Si(n=0)
     retourner 1
 sinon
    retourner n*fact(n-1)
Fin
```

II.8 Chaîne de caractères : Quelques fonctions prédéfinies sur les chaînes de caractères

Le type chaîne de caractères permet de décrire des objets formés par la juxtaposition de plusieurs caractères. Dans la plupart des langages de programmation, il existe des « outils » pour les manipuler.

II.8.1 Concaténation de chaînes

Cette fonction permet d'assembler plusieurs chaînes de caractères en une seule. Elle est notée par **&** ou **Concat**.

Syntaxe :

Concat(Ch1, Ch2): chaîne ou Ch1&Ch2: chaîne

Exemple :

''Merci'' & '' '' & ''beaucoup'' donne ''Merci beaucoup''

II.8.2 Longueur d'une chaîne

La longueur d'une chaîne désigne le nombre de caractères qui constitue la chaîne.

Syntaxe : Longueur (''chaîne''): entier

Exemples :

- Longueur (''Bonjour'') = 7
- Longueur ("test ") = 5

II.8.3 Sous chaîne

Syntaxe :

SousChaîne (ch : chaîne, i : entier, L : entier) : chaîne

Cette instruction retourne une sous-chaîne de longueur L extraite de la chaîne ch, à partir de la position i.

Exemple :

SousChaine(" Informatique",6,2)= " ma" .

II.9.4 Accès au ième caractère :

Syntaxe :

ième (ch : chaîne, i : entier) : caractère

Cette instruction retourne le ième caractère de la chaîne ch.

Exemple :

ième (" Salam" , 2) ="a"

II.9. Notion de tableaux

Exercice

Sachant que votre groupe est composée d'environ 200 étudiants, calculer la moyenne des notes, relative au module d'informatique pour les étudiants en gardant en mémoire toutes les notes pour faire d'autres calculs (Écart type, Note minimale, Note maximale …):

Solution

La seule solution dont nous disposons actuellement est de :

- déclarer autant de notes que d'étudiants : Note1, Note2,…Note200.
- saisir les 200 notes,
- calculer la moyenne : moyenne ← (Note1+Note2+…+Note200)/200

Pratiquement cette solution est difficile à réaliser. Il faudrait alors pouvoir par l'intermédiaire d'une seule variable stocker plusieurs valeurs de même type.

C'est le rôle des tableaux.

II.9.1 Tableaux à une dimension (Vecteurs)

II.9.1.1 Définition

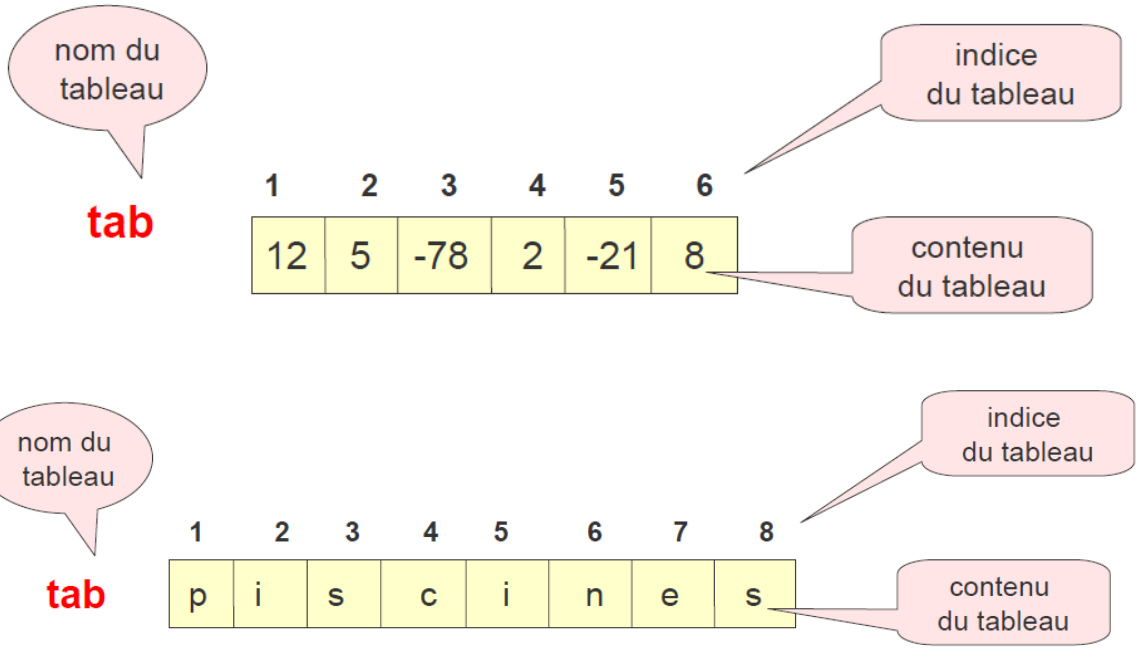
Un tableau est un ensemble d'éléments, ordonné, homogène et identifié par un nom:

- ordonné car les cases mémoires composant un tableau sont numérotés (souvent à partir de 0) ;
- homogène car un tableau ne peut contenir que des éléments de même type.

Remarques :

- Chacune des variables d'un tableau est numérotée, ce numéro s'appelle indice.
- Un indice sert à accéder à une valeur contenue dans un tableau.
- Les tableaux sont de types numériques, chaînes de caractères,…

Exemples :



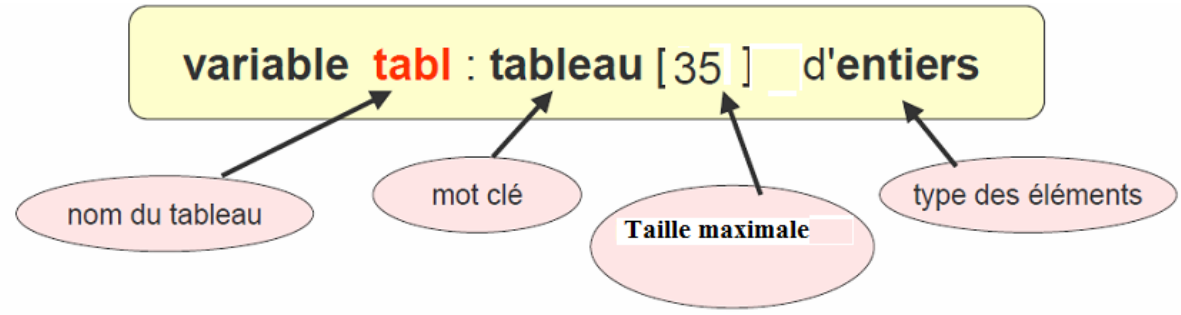
II.9.1.2 Déclaration d'un tableau

La déclaration d'un tableau à une dimension montre en particulier sa taille et le type de ces éléments.

Syntaxe :

<nom du tableau>: Tableau [Taille_Max] type_des_éléments

Exemple : Déclaration d'un tableau qui peut contenir jusqu'à 35 entiers.

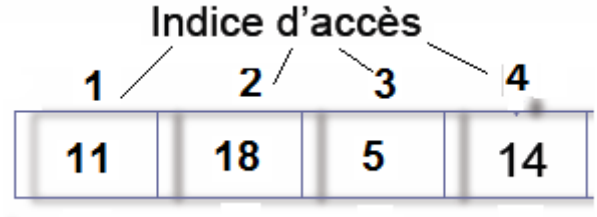


II.9.1.3 Accès aux éléments d'un tableau

Les éléments d'un tableau T ont des indices de 1 à n.

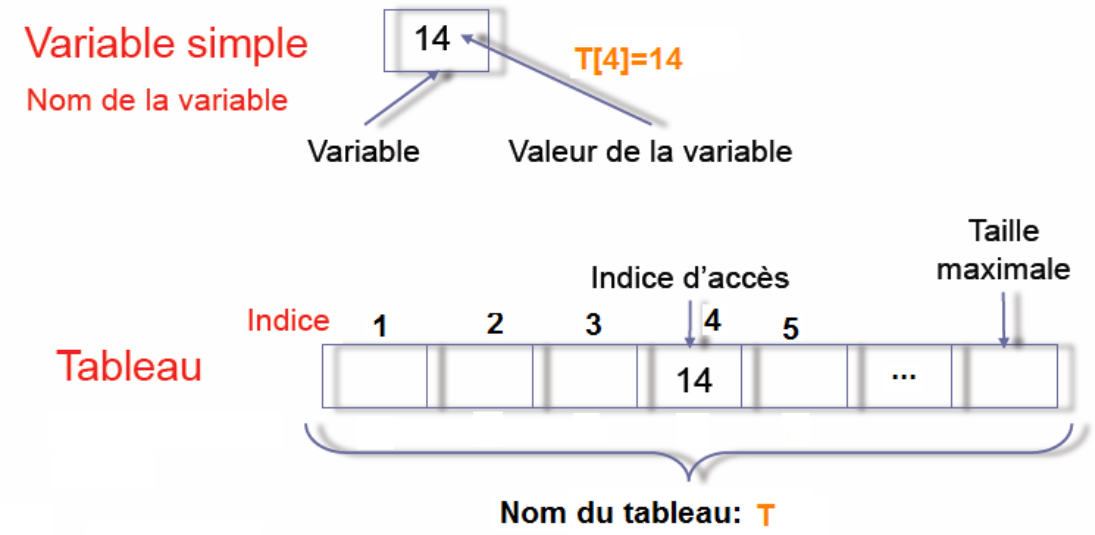
L'élément d'indice i du tableau T est noté T[i].

Exemple : Soit T un tableau formé des notes du module d'informatique pour 4 étudiants.



T[1]=11, T[2]=18, T[3]=5, T[4]=14

Résumé :



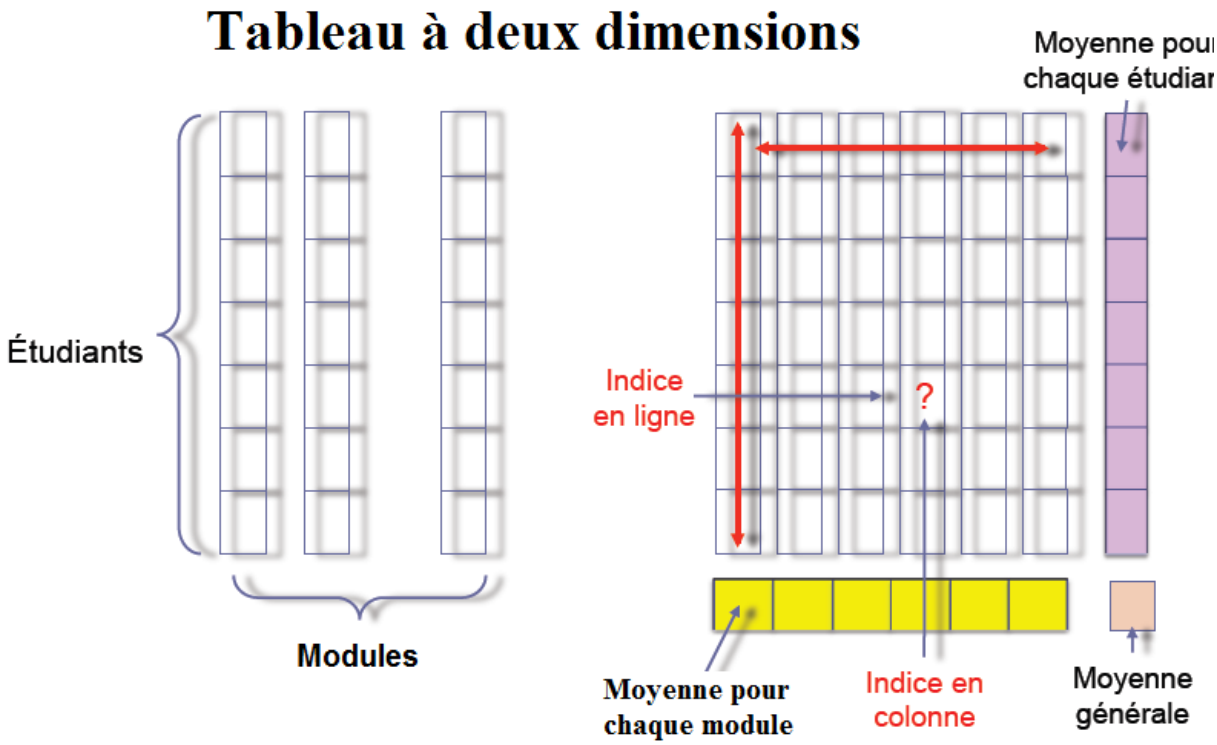
• Chaque variable du tableau est donc caractérisée par le nom du tableau et son indice.
• L'énorme avantage des tableaux, c'est qu'on va pouvoir les traiter à l'aide des boucles.

Exercice
Calculer la moyenne des notes du module d'informatique de votre groupe toute en gardant en mémoire les notes pour faire d'autres calculs (Écart type, Note minimale, Note maximale …):

Solution de l'exercice en utilisant les tableaux.

```
Algorithme Note_Moyenne_ST
Variable note: tableau[1..300] de réels
        moyenne, somme: réels
        i,n: entiers
Début
        ecrire(" Donner le nombre d'etudiants ")
        lire(n)
                somme ← 0                                   /* initialisation de la somme */
        Pour i ← 1 à n Faire                                /* la saisie des notes */
                Ecrire("Saisir la note de l'étudiant n° ", i)
                Lire(note[i])
                Somme ← somme+note[i]
        FinPour
        Moyenne ← somme/n
        Ecrire(" La moyenne est: " , moyenne)
Fin
```

II.9.2 Tableaux à deux dimensions
Problème :
•Comment sauvegarder et manipuler les notes des étudiants de votre groupe, pour les 4 modules du semestre 1?
• Calculer la moyenne de la promotion pour chaque matière.
• Calculer la moyenne pour chaque étudiant.
• Calculer la moyenne générale de tous les modules pour l'ensemble des étudiants.
Solution : Utilisation d'un tableau à deux dimensions



II.9.2.1 Définition

Un tableau à deux dimensions est défini selon le même principe qu'un tableau à une dimension sauf qu'on utilise deux indices au lieu d'un seul : un indice pour les lignes et un indice pour les colonnes.

II.9.2.2 Déclaration d'un tableau à deux dimensions

Syntaxe :

<nom du tableau>: Tableau [Max_lignes, Max_colonnes] type_des_éléments

Exercice : Calculer la somme de deux matrices carrées d'ordre n.