CHAPITRE 2 : ALGORITHMIQUE DE BASE

« Un programme informatique ne fait jamais ce que vous voudriez qu'il fasse…
Il fait seulement ce que vous lui dites de faire… »

I. Définitions :

I-1. Notion d'algorithme :

Les algorithmes sont présents dans notre vie, nous les exécutons ou nous les faisons exécuter tous les jours.
En informatique, l'ordinateur exécute aussi des algorithmes sauf qu'il est une machine non intelligente, c'est-à-dire qu'il n'est pas capable de résoudre des problèmes sans une description détaillée des actions à faire.

> « Un algorithme est une description en langage naturel d'une suite finie d'actions (ou d'instructions) à appliquer dans un ordre déterminé sur des données afin de résoudre un problème ».

I-2. Notion de programme :

Un ordinateur pour qu'il puisse effectuer des tâches aussi variées il suffit de le programmer.
Effectivement l'ordinateur est capable de mettre en mémoire un programme qu'on lui fournit puis l'exécuter. Plus précisément, l'ordinateur possède un ensemble limité d'opérations élémentaires qu'il sait exécuter. Un programme est constitué d'un ensemble de directives, nommées instructions, qui spécifient :

- les opérations élémentaires à exécuter
- la façon dont elles s'enchaînent.

Pour s'exécuter, un programme nécessite qu'on lui fournisse ce qu'on peut appeler « informations données » ou plus simplement « données ». En retour, le programme va fournir des « informations résultats » ou plus simplement résultats.

***Exemple* : Le programme PGCD (Le plus grand diviseur commun)**

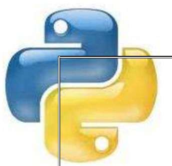
- Informations données: deux entiers a et b
- Les résultats: pgcd(a,b)

I-3. Programmation :

La programmation consiste, avant tout, à déterminer la démarche permettant d'obtenir, à l'aide d'un ordinateur, la solution d'un problème donné.
Le processus de la programmation se déroule en deux phases :

- Dans un premier temps, on procède à ce qu'on appelle l'analyse du problème posé ou encore la recherche d'un algorithme qui consiste à définir les différentes étapes de la résolution du problème. C'est la partie essentielle dans le processus de

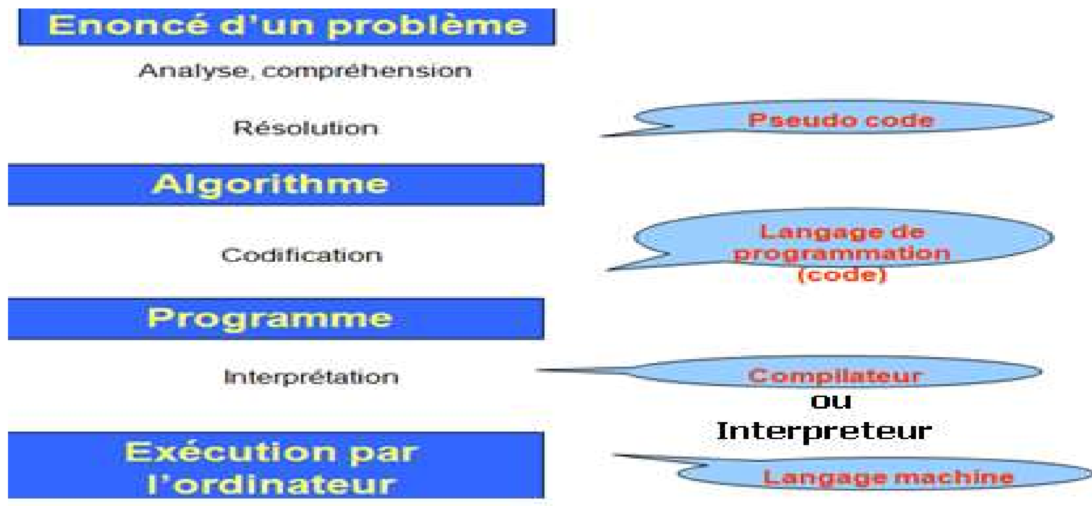
Prof : ZBAKH ABDEL ALI <zbakhabdou@gmail.com>



programmation. Elle permet de définir le contenu d'un programme en termes de données et d'actions.

- Dans un deuxième temps, on exprime dans un langage de programmation donné, le résultat de l'étape précédente. Ce travail, quoi qu'il soit facile, exige le respect strict de la syntaxe du langage de programmation.

Lors de l'étape d'exécution, il se peut que des erreurs syntaxiques soient signalées, ce qui entraîne des corrections en général simple ou des erreurs sémantiques plus difficiles à déceler. Dans ce dernier cas, le programme produit des résultats qui ne correspondent pas à ceux escomptés : le retour vers l'analyse sera alors inévitable.



Donc, la résolution d'un problème passe tout d'abord par la recherche d'un algorithme.

I-4. Langage de programmation :

C'est le langage utilisé par un programmeur pour créer des applications exécutables par un ordinateur. Les programmeurs utilisent divers langages de programmation pour écrire des applications. Il existe des centaines. Un certain nombre de ces langages sont directement compréhensibles par la machine. D'autres langages exigent des étapes intermédiaires de traduction.

a. Langage interprété :

Un langage informatique est par définition différent du langage machine. Il faut donc le traduire pour le rendre intelligible du point de vue du processeur. Un programme écrit dans un langage interprété a besoin d'un programme auxiliaire (l'interpréteur) pour traduire au fur et à mesure les instructions du programme.

Exemple : langage Python

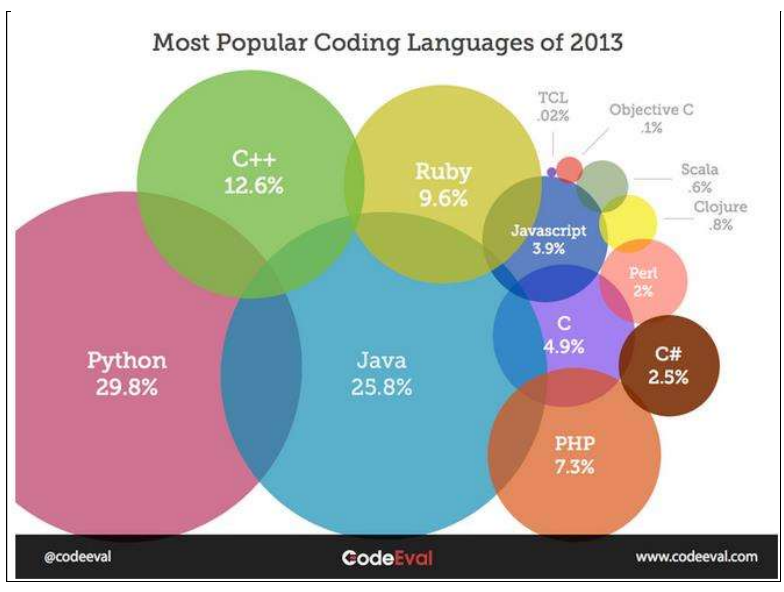
b. Langage compilé :

Un programme écrit dans un langage dit "compilé" va être traduit une fois pour toutes par un programme annexe (le compilateur) afin de générer un nouveau fichier qui sera autonome, c'est-à-dire qui n'aura plus besoin d'un programme autre que lui pour s'exécuter (on dit d'ailleurs que ce fichier est exécutable).

Un programme écrit dans un langage compilé a comme avantage de ne plus avoir besoin, une fois compilé, de programme annexe pour s'exécuter. De plus, la traduction étant faite une fois pour toute, il est plus rapide à l'exécution. Toutefois il est moins souple qu'un programme écrit avec un langage interprété car à chaque modification du fichier source (fichier intelligible par l'homme: celui qui va être compilé) il faudra recompiler le programme pour que les modifications prennent effet. D'autre part, un programme compilé a pour avantage de garantir la sécurité du code source. En effet, un langage interprété, étant directement intelligible (lisible), permet à n'importe qui de connaître les secrets de fabrication d'un programme et donc de copier le code voire de le modifier. Il y a donc risque de non-respect des droits d'auteur. D'autre part, certaines applications sécurisées nécessitent la confidentialité du code pour éviter le piratage (transaction bancaire, paiement en ligne, communications sécurisées, ...).

Exemple : langage C, Pascal

c. Langages couramment utilisés:



II. Représentation des algorithmes :

Historiquement, plusieurs types de notations ont représenté des algorithmes. Il y a eu notamment une représentation graphique, avec des carrés, des losanges, etc. qu'on appelait des **organigrammes**. Aujourd'hui, cette représentation est quasiment abandonnée, pour deux raisons. D'abord, parce que dès que l'algorithme commence à grossir un peu, ce n'est plus pratique du tout. Ensuite parce que cette représentation favorise le glissement vers un certain type de programmation, dite non structurée, que l'on tente au contraire d'éviter.

C'est pourquoi on utilise généralement une série de conventions appelée « **pseudo-code** », qui ressemble à un langage de programmation authentique dont on aurait évacué la plupart des problèmes de syntaxe. Ce pseudo-code est susceptible de varier légèrement d'un

Prof : ZBAKH ABDEL ALI <zbakhabdou@gmail.com>

livre à un autre. C'est bien normal : le pseudo-code, encore une fois, est purement conventionnel ; aucune machine n'est censée le reconnaître.

II-1. Organigramme :

C'est une représentation graphique de l'algorithme. Pour le construire, on utilise des symboles normalisés :

Quelques symboles utilisés dans la construction d'un algorigramme :

SYMBOLE	DESIGNATION	SYMBOLE	DESIGNATION
Symboles de traitement		Symboles auxiliaires	
	Symbole général Opération ou groupe d'opérations sur des données, instructions, pour laquelle il n'existe aucun symbole normalisé.		**Renvoi** Symbole utilisé deux fois pour assurer la continuité lorsqu'une partie de ligne de liaison n'est pas représentée.
	Sous-programme Portion de programme considérée comme une simple opération.		**Début, fin , interruption** Début, fin ou interruption d'un algorigramme.
	Entrée-Sortie Mise à disposition d'une information à traiter ou enregistrement d'une information traitée.		**Commentaire** Symbole utilisé pour donner des indications sur les opérations effectuées.
Symbole de test			
	Branchement Exploitation de conditions variables impliquant un choix parmi plusieurs.	Les différents symboles sont reliés entre eux par des lignes de liaisons.	

II-2. Pseudo-code :

Cette notation utilise un ensemble de mots clés et de structures permettant de décrire de manière complète, claire, l'ensemble des opérations à exécuter sur des données pour obtenir des résultats.

Structure d'un algorithme :

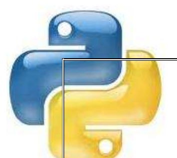
Un algorithme se compose de trois parties :

- ***L'en-tête*** : comprend le nom de l'algorithme
- ***Les déclarations*** : comprend la liste des données : des variables et des constantes
- ***Le corps*** : dans cette partie sont placées les instructions à exécuter (Les actions).

Exemple : *Un algorithme écrit en pseudo-code et qui permet de calculer et d'afficher la somme de deux nombres réels :*

Algorithme *Addition*	} ***L'en-tête***
Variables *A,B,S : réel*	} ***Les déclarations***

Prof : ZBAKH ABDEL ALI <zbakhabdou@gmail.com>



Début *Afficher('Donner la valeur de A')* *Saisir(A)* *Afficher('Donner la valeur de B')* *Saisir(B)* *S←A+B* *Afficher(''La somme de A et B est :'',S)* ***Fin***	***Le corps***

III. Eléments de base d'un algorithme

Les algorithmes agissent sur des données qui peuvent être de types différents et qui peuvent varier, ou rester constantes.

III-1. Les constantes :

Une constante est une donnée fixe qui ne varie pas tout le long de l'algorithme. Elle est caractérisée par : son ***nom*** et sa ***valeur*** (fixe)

Déclaration :

Constante nom_constante = valeur_constante

Exemple :

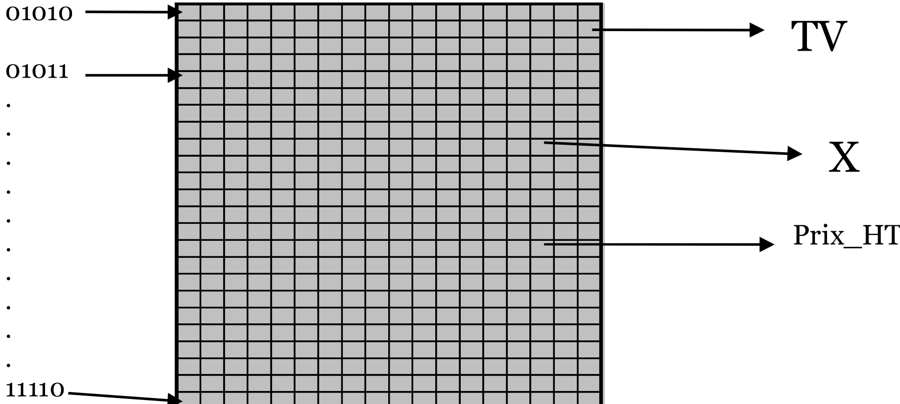
- constante Pi=3.14
- constante g=9.8

III-2. Les variables :

Dans un programme informatique, on va avoir en permanence besoin de stocker provisoirement en mémoire des valeurs. Il peut s'agir de données issues du disque dur ou fournies par l'utilisateur (frappées au clavier). Il peut aussi s'agir de résultats obtenus par le programme, intermédiaires ou définitifs. Ces données peuvent être de plusieurs types : elles peuvent être des nombres, du texte, etc. Dès que l'on a besoin de stocker une information au cours d'un programme, on utilise une variable.

a. Définition :

Une variable est un nom qui sert à repérer un emplacement donné de la mémoire. Cet emplacement est réservé pour contenir une valeur d'un type bien déterminé et qui pourra changer tout au long de l'algorithme



Cette notion contribue considérablement à faciliter la réalisation des programmes. Elle permet de manipuler des données sans avoir à se préoccuper de l'emplacement qu'elles occupent effectivement en mémoire. Pour cela, il vous suffit tout simplement de leur choisir un nom. Bien entendu, la chose n'est possible que parce qu'il existe un programme de traduction (compilateur, interpréteur) de votre programme dans le langage machine ; c'est lui qui attribuera une adresse à chaque variable. Le programmeur ne connaît que les noms TVA , X, Prix_HT… Il ne se préoccupe pas des adresses qui leur sont attribuées en mémoires.

b. Caractéristiques :

Une variable est caractérisée par : son **nom**, son **type** et sa **valeur** :

- ***Le nom :*** (on dit aussi identificateur) d'une variable, dans tous les langages, est formé d'une ou plusieurs lettres ; les chiffres sont également autorisés à condition de ne pas apparaître au début du nom. La plupart des signes de ponctuation sont exclus en particulier les espaces. Par contre, le nombre maximum de caractères autorisés varie avec les langages
- ***La valeur :*** A un instant donné, une variable ne peut contenir qu'une seule valeur. Bien sûr, cette valeur pourra évoluer sous l'action de certaines instructions du programme.
- ***Le type :*** Le type d'une variable définit la nature des informations qui seront représentées dans les variables (numériques, caractères…).

c. Déclaration :

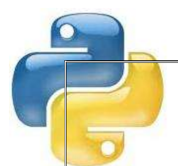
La première chose à faire tout au début de l'algorithme, avant de pouvoir utiliser des variables, c'est de faire la déclaration des variables.
Lorsqu'on déclare une variable, on lui attribue un nom et on lui réserve un emplacement mémoire. La taille de cet emplacement mémoire dépend du type de variable. C'est pour cette raison qu'on doit préciser lors de la déclaration le type du variable.

La syntaxe d'une déclaration de variable est la suivante :

VARIABLE nom_variable : TYPE_variable

ou

VARIABLES nom1_variable, nom2_variable,… : TYPE_variable



III-3. Les types de variables :

a. Type numérique :

Tous les langages, quels qu'ils soient offrent un « bouquet » de types numériques, dont le détail est susceptible de varier légèrement d'un langage à l'autre. On retrouve cependant les types suivants :

Type Numérique	Plage
Byte (octet)	0 à 255
Entier simple	-32 768 à 32 767
Entier long	-2 147 483 648 à 2 147 483 647
Réel simple	$-3,40 \times 10^{38}$ à $-1,40 \times 10^{45}$ pour les valeurs négatives $1,40 \times 10^{-45}$ à $3,40 \times 10^{38}$ pour les valeurs positives
Réel double	$1,79 \times 10^{308}$ à $-4,94 \times 10^{-324}$ pour les valeurs négatives $4,94 \times 10^{-324}$ à $1,79 \times 10^{308}$ pour les valeurs positives

Exemple :

VARIABLES i, j, k : **ENTIER**

b. Type alphanumérique :

Les variables peuvent contenir bien d'autres informations que des nombres. On dispose également du type chaîne (également appelé caractère ou alphanumérique).

Dans une variable de ce type, on stocke des caractères, qu'il s'agisse de lettres, de signes de ponctuation, d'espaces, ou même de chiffres. Le nombre maximal de caractères pouvant être stockés dans une seule variable chaîne dépend du langage utilisé.

Exemple :

VARIABLE nom : *CHAINE*

c. Type booléen :

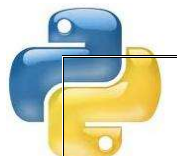
Une variable de type booléen prend uniquement deux valeurs VRAI ou FAUX.

Exempe :

VARIABLE x : *BOOLEEN*

III-4. Les instructions de base : L'affectation :

L'affectation est une attribution d'une valeur à une variable. Elle se note généralement en algorithmique avec le signe ←



Syntaxe :

nom_variable ← *valeur*
nom_variable ← *EXPRESSION*

Remarque : Une instruction d'affectation ne change que ce qui est situé à gauche de la flèche.

Exemple :

Soit l'instruction a ← 23 ; Attribue la valeur 23 à la variable a
Soit l'instruction b ← 2+5 ; Attribue la valeur 7 à la variable b
Soit l'instruction a ← b ; Attribue la valeur de la variable b à la variable a.

III-5. Les instructions de Lecture et Ecriture :

Pour programmer la communication entre l'ordinateur et l'extérieur on utilise les instructions de lecture et écriture.

a. Ecriture (Affichage):

C'est l'action qui permet à l'algorithme d'afficher pour son utilisateur des messages ou des résultats de calculs

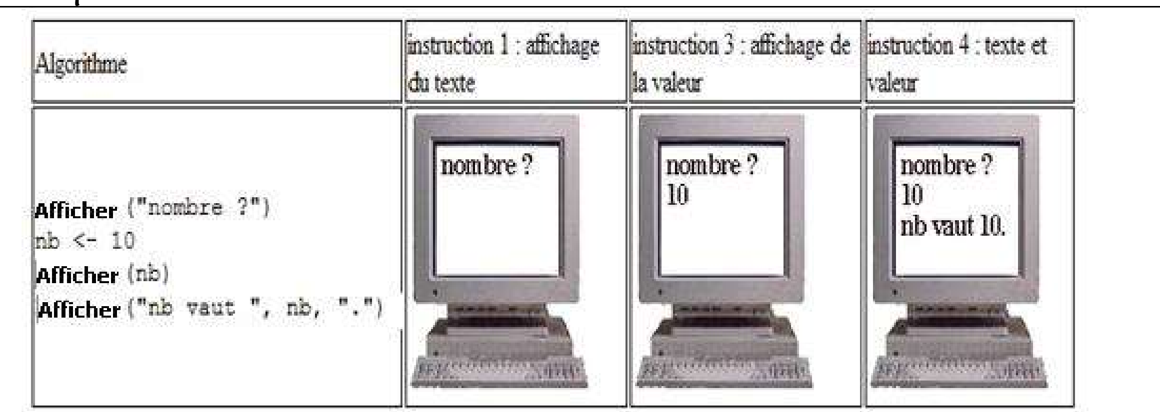
Syntaxe :

- Afficher un texte :
  Afficher ("texte à afficher")
- Afficher la valeur d'une variable :
  Afficher (nom_de_la_variable)
- Mélange de texte et de valeurs :
  Afficher("texte",nom_de_la_variable,"texte", nom_de_la_variable)

Exemple 1 :

Afficher (''La surface ?'')	affiche La ***surface ?*** à l'écran
Afficher (S)	affiche le contenu de la variable S à l'écran

Exemple 2 :



Algorithme	instruction 1 : affichage du texte	instruction 3 : affichage de la valeur	instruction 4 : texte et valeur
Afficher ("nombre ?") nb <- 10 **Afficher** (nb) **Afficher** ("nb vaut ", nb, ".")	nombre ?	nombre ? 10	nombre ? 10 nb vaut 10.

d. Lecture (Saisir par l'utilisateur):

Considérons l'algorithme suivant :

```
VARIABLE A : ENTIER
Début
        A ← 12 ^ 2
        Afficher(A)
Fin
```

Il permet de calculer et d'afficher le carré de 12. Le problème de cet algorithme, c'est que, si l'on veut calculer le carré d'un autre nombre que 12, il faut réécrire l'algorithme.

En effet, il existe une instruction qui permet à l'utilisateur de faire entrer des valeurs au clavier pour qu'elles soient utilisées par l'algorithme. La syntaxe de cette instruction de lecture est :

- Lire une donnée :

  ***Saisir** (nom_de_la_variable)*

- Lire plusieurs données :

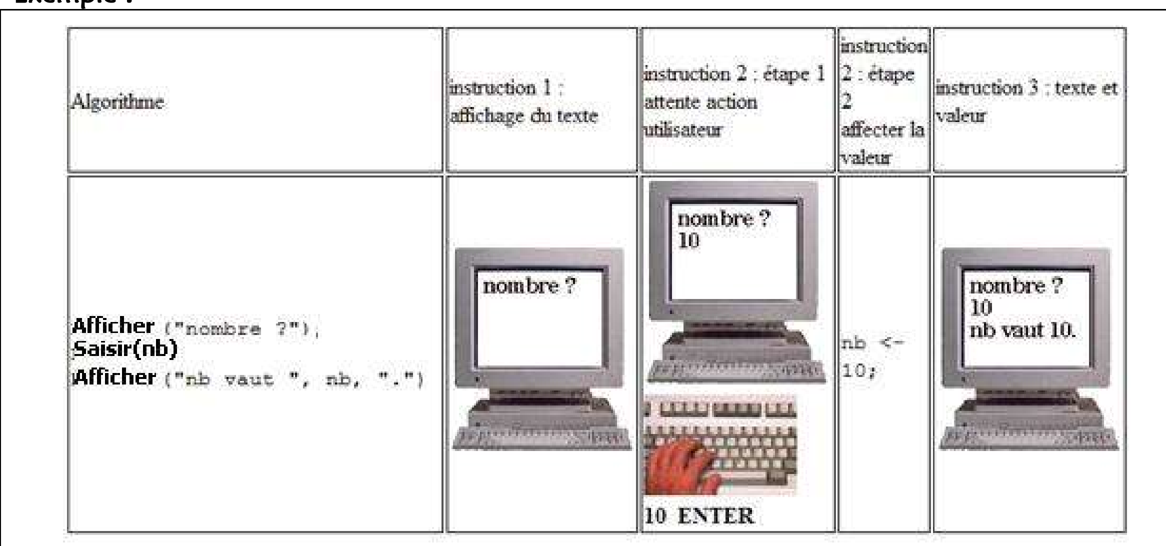
  ***Saisir** (nom_de_la_variable_1, nom_de_la_variable_2, ...)*

Lorsque l'algorithme rencontre une instruction ***Saisir***, l'exécution de l'algorithme s'interrompt, attendant la saisie d'une valeur au clavier. Dès que l'on frappe sur la touche ENTER, l'exécution reprend.

Remarque :

Avant de lire une variable, il est fortement conseillé d'écrire des libellés à l'écran, afin de prévenir l'utilisateur de ce qu'il doit frapper. La même chose pour l'instruction d'écriture.

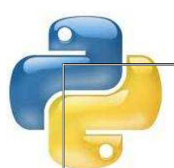
Exemple :



Algorithme	instruction 1 : affichage du texte	instruction 2 : étape 1 attente action utilisateur	instruction 2 : étape 2 affecter la valeur	instruction 3 : texte et valeur
Afficher ("nombre ?"); **Saisir(nb)** **Afficher** ("nb vaut ", nb, ".")	nombre ?	nombre ? 10 10 ENTER	nb <- 10;	nombre ? 10 nb vaut 10.

IV. Expressions et Opérateurs :

Une expression : est un ensemble de valeurs, reliées par des **opérateurs**, et équivalente à une seule valeur.



Exemple :

- 5+3-5 ; l'expression est 5+3-5
- 'Bien'+'venue' ; l'expression est 'Bien'+'venue'

Un opérateur : est un signe qui relie deux valeurs pour produire un résultat, Parmi les opérateurs, on distingue les fonctions et les opérateurs.

IV-1. Les fonctions :

- La fonction **DIV** permet de donner le résultat de la division entière d'un nombre par un autre. 7 DIV 2 = 3
- La fonction **MOD** (se lit Modulo), permet de donner le reste de la division entière d'un entier par un autre. 7 MOD 2 = 1
- La fonction ** ou ^ permet d'élever un nombre à la puissance d'un autre. 2**3=8

IV-2. Opérateurs sur les entiers et les réels :

Opérateurs numériques Arithmétiques:	*Opérateurs numériques de comparaison:*
• + : Addition • - : Soustraction • * : Multiplication • / : Division • ^ : Puissance	• > : Supérieur • < : Inférieur • >= : Supérieur ou égal • <= : Inférieur ou égal • == : Egal • != : Différent

IV-3. Opérateurs alphanumériques (& ou +)

Cet opérateur permet de concaténer ou de joindre deux chaînes de caractères

IV-4. Opérateurs logiques :

Il s'agit du **ET**, du **OU** et du **NON**

IV-5. Priorité des opérateurs :

Priorité à la multiplication et à la division.

V. Les structures algorithmiques fondamentales :

V-1. Structure séquentielle :

La structure la plus simple. Elle traduit une suite d'instructions élémentaires, exécutées l'une après l'autre, dans un ordre logique.

Exemple :

```
Algorithme structure_séq
Variables Qte ,Prix, Montant : entier
Debut
        Saisir(Qté)
        Saisir(Prix)
        Montant ← Qté*Prix
        Afficher( ''Le montant est : '' ,Montant)
Fin
```

V-2. Structure conditionnelle :

e. La forme réduite :

Elle se traduit par **Si** : si condition vraie, exécuter une ou plusieurs instructions

```
Si Condition Alors
        bloc d'instructions
Fin Si
```

Où ***Condition*** est une expression à valeur booléenne lors de son exécution, l'expression ***Condition*** est évaluée. Si Val(***Condition***)=Vraie, l'instruction ***bloc 1d'instructions*** est exécutée, sinon rien n'est exécuté.

Exemple :

```
SI note > 0 ALORS
        Afficher (''Note positive'')
FIN SI
```

f. La forme alternative :

La deuxième forme possible est :

```
Si Condition Alors
        bloc1 d'instructions
Sinon
        bloc2 d'instructions
Fin Si
```

Si la ***condition*** mentionnée après ***SI*** est ***VRAIE***, on exécute le ***bloc1 d'instructions*** (ce qui figure après le mot ***ALORS***); si la ***condition*** est fausse, on exécute le ***bloc2 d'instructions*** (ce qui figure après le mot ***SINON***).

Exemple :

```
SI note > 0 ALORS
        Afficher (''Note positive'')
SINON
        Afficher (''Note négative'')
FIN SI
```

g. La forme imbriquée :

Il peut arriver que l'une des parties d'une structure alternative contienne à son tour une structure alternative. Dans ce cas, on dit qu'on a des structures alternatives imbriquées les unes dans les autres.

```
Si Condition Alors
        bloc1 d'instructions
Sinon
        Si Condition2 Alors
                bloc2 d'instructions
        Sinon
                Bloc3 d'instructions
        Fin Si
Fin Si
```

Exemple :

```
SI note > 0 ALORS
        Afficher (''Note positive'')
SINON
        SI note ==0 ALORS
                Afficher (''Note nulle)
        SINON
                Afficher (''Note négative'')
        FIN SI
FIN SI
```

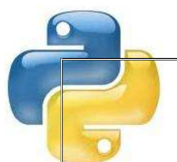
On peut aussi écrire :

```
SI note > 0 ALORS
        ECRIRE (''Note positive'')
FIN SI

SI note ==0 ALORS
        ECRIRE (''Note nulle)
FIN SI

SI note <0 ALORS
        ECRIRE (''Note négative'')
FIN SI
```

La première version est plus simple à écrire et plus lisible. Elle est également plus performante à l'exécution. En effet, les conditions se ressemblent plus ou moins, et surtout on



oblige la machine à examiner trois tests successifs alors que tous portent sur la même chose : la valeur de la variable note.
Mais aussi, et surtout, nous avons fait des économies sur le temps d'exécution de l'ordinateur. Si la note est supérieure à zéro, celui-ci écrit «Note positive' » et passe directement à la fin, sans être ralenti par l'examen des autres possibilités.

h. Les conditions :

Les conditions sont évaluées comme étant vraies ou fausses, si la condition est vraie, alors on exécutera le premier bloc d'instructions, sinon ce sera le deuxième bloc qui sera exécuté. Ces conditions sont souvent le résultat de comparaisons.

Les opérateurs de comparaison sont :

Opérateur	Signification
==	égale
!=	différent
<	inférieur
>	supérieur
<=	inférieur ou égal
>=	supérieur ou égal

Certains problèmes exigent parfois de formuler des conditions qui ne peuvent pas être exprimées sous la forme simple vu en dessus. A cet effet, la plupart des langages autorisent des conditions formées de plusieurs conditions simples reliées entre elles par ce qu'on appelle des opérateurs logiques. Ces opérateurs sont : **ET**, **OU** et **NON**.

- ***Exemple 1:*** Pour que la condition complexe, **condition1 ET condition2** soit **VRAI**, il faut impérativement que la **condition1** soit **VRAI** et que la **condition2** soit **VRAI**.
- ***Exemple 2:*** Pour que la condition **condition1 OU condition2** soit **VRAI**, il suffit que **condition1** soit **VRAI** ou **condition2** soit **VRAI**. Il est à noter que cette condition complexe sera **VRAI** si **condition1** et **condition2** sont **VRAI**.
- ***Exemple 3:*** Le **NON** inverse une condition : **NON**(**condition**) est **VRAI** si **condition** est **FAUX**, et il sera **FAUX** si **condition** est **VRAI**.

D'une manière générale, les opérateurs logiques peuvent porter, non seulement sur des conditions simples, mais aussi sur des conditions complexes. L'usage de parenthèses permet dans de tels cas de régler d'éventuels problèmes de priorité. Par exemple, la condition :
(a < 0 **ET** b > 1) **OU** (a > 0 **ET** b > 3) est **VRAI** si l'une au moins des conditions entre parenthèses est VRAI.

I-2. structures itératives (ou répétitives) :

a. La structure de répétition POUR:

La structure de répétition **Pour** permet de répéter l'exécution d'une suite d'instructions un certain nombre de fois. Sa syntaxe est :

```
Pour compteur← val_initial à val_final
    Instructions à répéter
FinPour
```

- *compteur* : c'est ce qu'on appelle **compteur**. C'est une variable de type entier.
- val_*initial* et *val_final :* sont respectivement la valeur initiale et la valeur finale prise par le compteur. Ce sont des valeurs entières.
- A chaque itération, on ajoute 1 à *compteur*

Exemple :

```
Algorithme boucle
Variable i:entier
Début
        Pour i ← 0 à 10
                Afficher(i*i)
        FinPour
Fin
```

L'algorithme affiche les résultats suivants :

```
0
1
4
9
16
25
36
49
64
81
100
```

b. La structure de répétition Tant Que:

Cette structure sert à répéter un ensemble d'instructions jusqu'à ce qu'une certaine condition soit réalisée.

```
Tant que condition
        Instruction 1
        Instruction 2
        .
        .
        .
Fintantque
```

Exemple :

```
Algorithme boucle2
Variable i:entier
Début

i ← 0
Tant que i <=10
        Afficher(i*i)
        i ← i+1
Fintantque

Fin
```

L'algorithme affiche les résultats suivants :

```
0
1
4
9
16
25
36
49
64
81
100
```


Prof : ZBAKH ABDEL ALI <zbakhabdou@gmail.com>