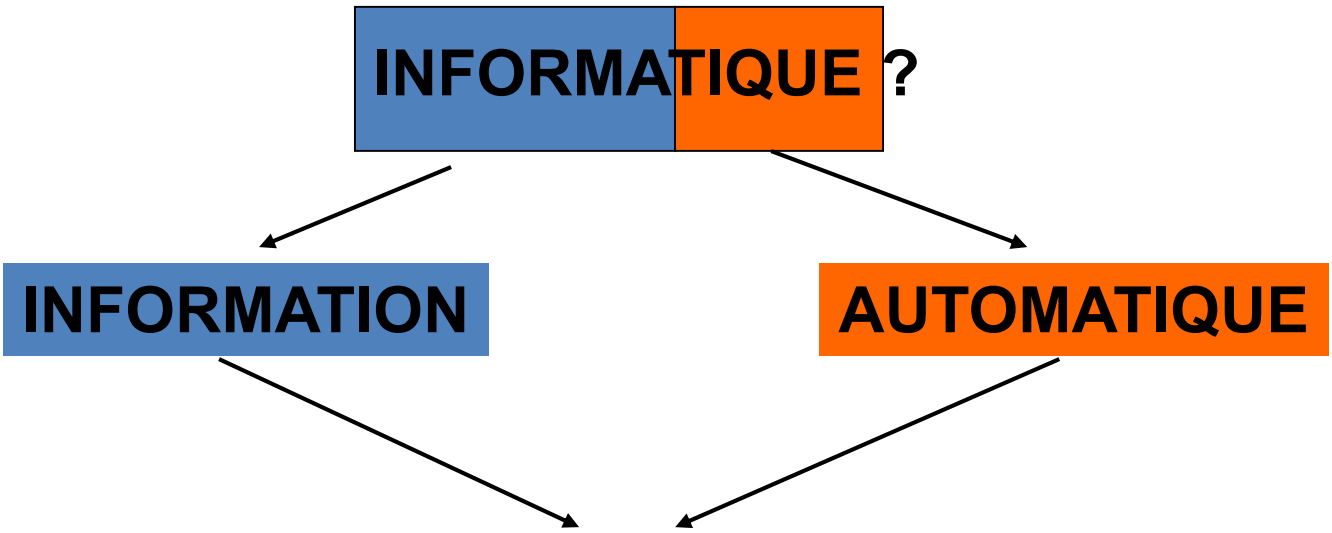
Chapitre 1 : Introduction générale sur l'algorithme

I. Définition de l'informatique

L'informatique est la science qui regroupe l'ensemble des théories et des techniques permettant de traiter de l'information à l'aide d'un ordinateur.



Traitement **automatique** de l'information → Ordinateur

- Un ordinateur est une machine qui permet de traiter de l'information :
  - d'acquérir et de conserver de l'information (acquisition, stockage)
  - d'effectuer des traitements (calcul),
  - de restituer les informations stockées (restitution)

- Un Ordinateur permet de lire «information» ⇔ «données» (0 ou 1)
- Un ordinateur traite différents types d'informations : valeurs numériques, textes, images, sons, …: tout cela avec des 0 ou 1

II. Langage machine

Langage **binaire**: l'information est exprimée et manipulée sous forme d'une suite de bits

Un **bit** (*binary digit*) = 0 ou 1 (2 états électriques)

Une combinaison de 8 bits= 1 **Octet** ➔ possibilités qui permettent de coder tous les caractères alphabétiques, numériques, et symboles tels que ?,*,&, …

Le code **ASCII** (*American Standard Code for Information Interchange*) donne les correspondances entre les caractères alphanumériques et leurs représentations binaires, Ex. A= 01000001, ?=00111111

ASCII	Decimal	Binary
@	64	0100 0000
A	65	0100 0001
B	66	0100 0010
C	67	0100 0011
D	68	0100 0100
E	69	0100 0101
F	70	0100 0110
G	71	0100 0111
H	72	0100 1000
I	73	0100 1001
J	74	0100 1010
K	75	0100 1011
L	76	0100 1100
M	77	0100 1101
N	78	0100 1110
O	79	0100 1111

III. Langage de programmation

- Problème: le langage machine est difficile à comprendre par l'humain
- Idée: trouver un langage compréhensible par l'homme qui sera ensuite converti en langage machine

  ➔ **Les langages de programmation**

IV. L'algorithme

Reflexion :

Un exemple de problème qui nous concerne tous: vous êtes dans une cuisine, vous trouvez du riz, comment le cuire ?

Voici une marche à suivre simple :

1- remplir une casserole d'eau ;

2- y ajouter une pincée de sel ;

3- la mettre sur le feu ;

4- attendre l'ébullition de l'eau ;

5- mettre le riz dans la casserole ;

6- le laisser cuire 10 à 15 minutes ;

7- égoutter le riz.

Quels sont les outils utilisés ? et Comment on les a utilisé ?

- casserole / riz / eau / feu / sel : données
- on les a utilisé en suivant une procédure logique : algorithme

Définition :

Algorithme : Procédure décrivant, étape par étape, une méthode permettant de résoudre un problème.

Mot provenant du nom d'un mathématicien arabe du IX^{éme} siècle El-Khawarizmi

C'est la base de tout programme informatique.

Un algorithme est suite finie d'instructions vérifiant :

- Chaque étape est décrite de façon **précise**.
- Chaque étape est **déterministe**: produit des résultats uniques.
- Reçoit des **données en entrée**.

- Produit des **données en sortie**.

Exemple 1 : Somme de deux nombres :

Données en entrées : Deux nombres.

Données en sorties : Somme

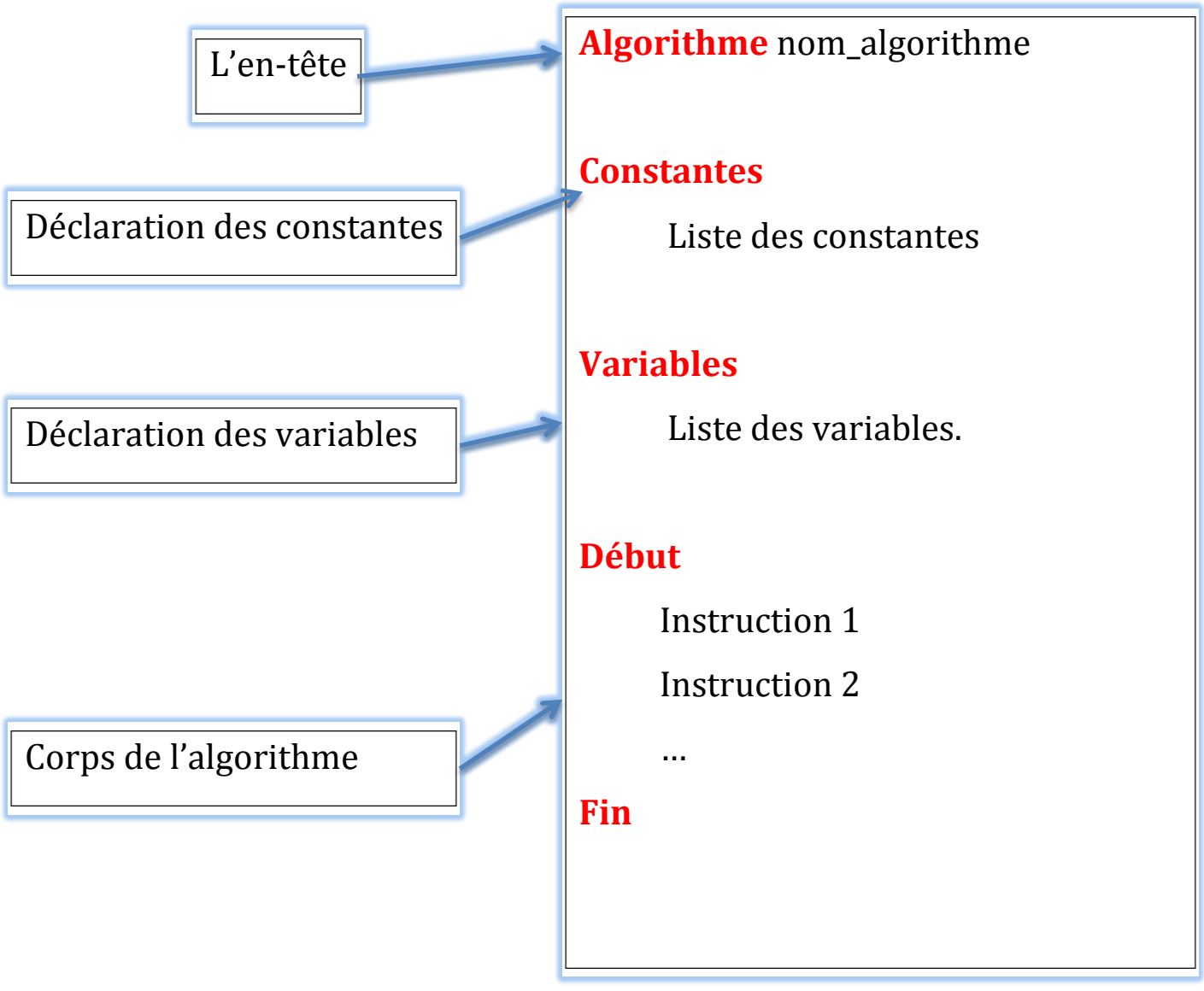
Exemple 2 : Tri d'un ensemble d'éléments :

Données en entrées : Suite de n éléments : a1, a2, a3,……,an

Données en sorties : La suite réordonnée.

V. Structure générale

La structure de base d'un algorithme est la suivante :



L’en-tête : Permet l’identification d’un programme.

Les déclarations : Les zones de stockage qui seront utilisées par le programme.

Le corps de l’algorithme : Liste des instructions à exécuter par l’ordinateur.

Chapitre 2 : Les variables

I. Le rôle des variables

Les variables servent comme un lieu de stockage pour un programme informatique.

Pour employer une image, une variable est une boîte, que le programme (l'ordinateur) va repérer par une **étiquette** (nom). Pour avoir accès au **contenu de la boîte**, il suffit de la désigner par son étiquette.

Une variable désigne en fait un **emplacement mémoire** dont le contenu peut changer au cours d'un programme (d'où le nom variable).

Les variables doivent être **déclarées avant d'être utilisées**, elle doivent être caractérisées par :

- Un nom (Identificateur)
- Un type (entier, réel, caractère, chaîne de caractères, …)

Exemple :

Pour réaliser un algorithme pour faire l'addition entre deux nombre. Nous avons besoin des emplacements pour stocker les valeurs de ces deux pour que l'ordinateur réaliser l'opération souhaitée.

II. Déclaration des variables

La déclaration des variables permet de réserver une espace mémoire (boite) qui va être utilisé par son nom dans le corps de l'algorithme.

La forme générale pour déclarer une variable est la suivante :

Nom_de_la_variable : Type_de_la_variable

1.1. Nom des variables

Le choix des noms de variables est soumis à quelques règles qui varient selon le langage, mais en général:

- Un nom doit commencer par une lettre alphabétique

  Exemple valide : A1

  Exemple invalide : 1A

- Doit être constitué uniquement de lettres, de chiffres et du soulignement _ (Eviter les caractères de ponctuation et les espaces)

  Exemple valides : SMIP2007, SMP_2007

  Exemple invalides : SMP 2005 , SMI-2007, SMP;2007

- Doit être différent des mots réservés du langage (par exemple en Java: **int, float, else, switch, case, default, for, main, return**, …).

- La longueur du nom doit être inférieure à la taille maximale spécifiée par le langage utilisé.

Conseil : pour la lisibilité du code choisir des noms significatifs qui décrivent les données manipulées.

Exemples: TotalVentes2004 , Prix_TTC , Prix_HT.

1.2. Les types des variables

Le type d'une variable détermine l'ensemble des valeurs qu'elle peut prendre, les types offerts par la plus part des langages sont:

- Type numérique (entier ou réel)

  Byte (codé sur 1 octet): de 0 à 255

  Entier court(codé sur 2 octets) : -32 768 à 32 767

  Entier long (codé sur 4 ou 8 octets)

  Réel simple précision (codé sur 4 octets)

  Réel double précision (codé sur 8 octets)

  - Remarque: pour le type numérique on va se limiter aux entiers et réels sans considérer les sous types

- Type logique ou booléen:

  Deux valeurs VRAI ou FAUX

- Type caractère:

  Lettres majuscules, minuscules, chiffres, symboles, …

  Exemples: 'A', 'a', '1', '?', …

- Type chaîne de caractère:

Toute suite de caractères, **Exemples: " Nom, Prénom", "code postale: 1000", ...**

Exemple :

```
Algorithme Exemple_Declaration
Variables
        Var1 : Entier
        Var2, Var3 : booléen
        Var4, var5 : caractére
        Var6 : chaine
  Début


  Fin
```

III. Affectation

L'affectation consiste à attribuer une valeur à une variable (ça consiste en fait à remplir où à modifier le contenu d'une zone mémoire)

En algorithme, l'affectation se note avec le signe ←

Var← e : attribue la valeur de e à la variable Var

- e peut être une valeur, une autre variable ou une expression

- Var et e doivent être de même type ou de types compatibles
- L'affectation ne modifie que ce qui est à gauche de la flèche.
- l'affectation n'est pas commutative : A=B est différente de B=A.

- **Exemple valides :**

```
Algorithme Exemple_Valide_affectation
Variables
   i, j,k : entier
   x, y : réel
   OK: booléen
   ch1, ch2 : chaîne de caractères
Début
   i ←1
   j ←i
   k ←i+j
   x ←10.3
   OK ←FAUX
   ch1 ←"TDI"
   ch2 ←ch1
   x ←4
   x ←j
Fin
```

- **Exemple non valides :**

```
Algorithme Exemple_NONValide_affectation
Variables
    i, j,k : entier
    x, y : réel
    OK: booléen
    ch1, ch2 : chaîne de caractères
Début
    i ←10.3
    OK ←"TDI"
    j ←x
Fin
```

Certains langages de programmation donnent des valeurs par défaut aux variables déclarées. Pour éviter tout problème il est préférable d'initialiser les variables déclarées.

IV. Exercices d'application

1.1. Exercice 1

Donnez les valeurs des variables A, B et C après exécution des instructions suivantes ?

Algorithme Exo1

Variables

A, B, C: **Entier**

Début

A ← 3

B ← 7

A ← B

C ← A

Fin

Solution : A =7 / B=7 / C=7

1.2. Exercice 2

Donnez les valeurs des variables A et B après exécution des instructions suivantes ?

Algorithme Exo2

Variables A, B : **Entier**

Début

A ← 1

B ← 2

A ← B

B ← A

Fin

Solution : A=2 / B=2

Les deux dernières instructions permettent-elles d'échanger les valeurs de A et B ? **Réponse : Non**

1.3. Exercice 3

Ecrire un algorithme qui permet de :

- Déclarer trois variables A, B et C de type entier
- Initialiser les valeurs de A et B par des valeurs de votre choix.
- Echanger les valeurs A et B en utilisant la variable C.

Solution :

Algorithme Echange_valeurs_A_B

Variables A,B,C : entier

Début

A← 5

B←10

C←B

B←A

A←C

Fin

V. Expression et opérateurs

Une expression peut être une valeur, une variable ou une opération constituée de variables reliées par des opérateurs

Exemples: 1, b, a*2, a+ 3*b-c, ...

L'évaluation de l'expression fournit une valeur unique qui est le résultat de l'opération

Les **opérateurs** dépendent du type de l'opération, ils peuvent être :

- **Des opérateurs arithmétiques: +, -, *, /, % (modulo), ^ (puissance)**

  Exemple :

  Algorithme OP_ARTH

  Variables

  A , B : Entier

  Début

  A ← 4 + 3

  B ← A * 2

  Fin

- **Des opérateurs logiques: NON, OU, ET**
- **Des opérateurs relationnels: =,<> , <, >, <=, >=**
- **Des opérateurs sur les chaînes: & (concaténation)**

  Exemple :

  Algorithme OP_CHAINE

  Variables

  A , B, C : chaine

  Début

  A ← **"TDI"**

  B ← **"TRI"**

  C ← A & B

  Fin

Une expression est évaluée de gauche à droite mais en tenant compte de **priorités**

Pour les opérateurs arithmétiques donnés ci-dessus, l'ordre de priorité est le suivant (du plus prioritaire au moins prioritaire):

- ^ : (élévation à la puissance)
- * , / (multiplication, division)
- % (modulo)
- + , - (addition, soustraction)

Exemple: 2 + 3 * 7 vaut 23

En cas de besoin (ou de doute), on utilise les parenthèses pour indiquer les opérations à effectuer en priorité

exemple: (2 + 3) * 7 vaut 35

VI. Les exercices d'application

1.1. Exercice 1

Que produit l'algorithme suivant :

Algorithme EXO_1

Variables

A , B , C : chaine

Début

A ← **"432"**

B ← **"15"**

C ← **A & B**

Fin

Solution : C = "43215"

1.2. Exercice 2

Que produit l'algorithme suivant :

Algorithme EXO_2

Variables

A , B , C : chaine

Début

A ← **"432"**

B ← **"15"**

C ← **A + B**

Fin

Solution : Erreur on peut jamais additionner des chaines de caractères

1.3. Exercice 3

Que produit l'algorithme suivant :

Algorithme EXO_3

Variables

A , B : entier

Début

A ← **4**

B ← **1**

A ← A + B

B ← **A – B**

A ← A - B

Fin

Solution : A = 1 / B = 4 (Une autre façon pour échanger les valeurs car on peut considérer que l'expression est comme une variable)

Chapitre 3 – Lecture Ecriture

Les instructions de lecture et d'écriture permettent à la machine de communiquer avec l'utilisateur

I. Lecture

La **lecture** permet d'entrer des donnés à partir du clavier

- On note, **lire (var)** :

  La machine met la valeur entrée au clavier dans la zone mémoire nommée var

Exemple :

```
Algorithme Carre_de_deux_nombre
Variables
        A, B : entier
Debut
    Lire(A)
    B ← A * A
Fin
```

II. Ecriture

L'écriture permet d'afficher des résultats à l'écran.

- En pseudo-code, on note: **écrire (var)**

La machine affiche le contenu de la zone mémoire var

Conseil: Avant de lire une variable, il est fortement conseillé d'écrire des messages à l'écran, afin de prévenir l'utilisateur de ce qu'il doit frapper.

Exemple :

```
Algorithme Carre_de_deux_nombre
Variables
        A, B : entier
Debut
    Ecrire("Entrer une valeur :  ")
    Lire(A)
    B ← A * A
    Ecrire("Le carré de  ",A , " est : ",B)
Fin
```

III. Exercices

1.1. Exercice 1

Quelles seront les valeurs des variables A, B et C après exécution des instructions suivantes :

```
Algorithme EXO1
Variables
        A, B : entier
```

```
Debut
    A ← 2
    B ← 5
    A ← A + 1
    B ← A * B / A + B - 1
    A ← B / A
Fin
```

1.2. Exercice 2

Ecrire un algorithme qui permet de faire la somme de deux nombres saisis par l'utilisateur.

1.3. Exercice 3

Ecrire un algorithme qui permet de faire l'addition, la soustraction, produit et la division de deux nombres saisis par l'utilisateur.

1.4. Exercice 4

Ecrire un algorithme qui vous demande de saisir votre nom puis votre prénom et qui affiche ensuite votre nom complet

1.5. Exercice 5

Ecrire un algorithme qui permet de calculer la surface d'un carré. (Surface d'un carré = longueur * longueur / longueur^2).

1.6. Exercice 6

Ecrire un algorithme qui permet de calculer la surface d'un rectangle. (Surface d'un rectangle = longueur * largeur)

1.7. Exercice 7

Ecrire un algorithme qui permet de calculer la surface d'un cercle. (Surface d'un cercle = Pi (=3,14) * R^2)

1.8. Exercice 8

Écrire un algorithme qui lit le prix HT d'un article, le nombre d'articles et le taux de TVA, et qui fournit le prix total TTC correspondant. Faire en sorte que des libellés apparaissent clairement. (Prix total TTC = prix HT * Nombre article * (1 + taux TVA).

1.9. Exercice 9

Écrire un algorithme qui permet d'échanger les valeurs de deux entiers A et B (les valeurs de A et B, vont être saisies par l'utilisateur). Il est demandé dans ce programme :

- D'afficher les deux valeurs initiales de A et B.
- Effectuer la permutation de A et B.
- Afficher les deux valeurs de A et B permutées.

1.10. Exercice 10

Un magasin dispose de cinq produits :

Produit A : prix 5.00 DH

Produit B : prix 2.50 DH

Produit C : prix 3.00 DH

Produit D : prix 10.00 DH

Produit E : prix 7.00 DH

Un client achète :

X unités du produit A, Y unités du produit B, Z unités du produit C, T unités du produit D, U unités du produit E.

On désire calculer et afficher :

- Le prix hors taxe (PHT) de cette vente.
- La taxe sur la valeur ajoutée (TVA)
- Le prix toutes taxes comprises (PTTC) de cette vente
- On donne le taux de TVA : TTVA=0.20

1. Déterminer les données en entrées.
2. Déterminer les données en sorties.
3. Déterminer la formule pour calculer Le prix toutes taxes comprises (PTTC) de cette vente

4. Écrire un algorithme qui permet de calculer et afficher : Le prix hors taxe, la taxe sur la valeur ajoutée et le prix toutes taxes comprises ? (L'utilisateur doit entrer le nombre d'articles achetés de chaque produit)

Chapitre 4 – Les structures alternatives

I. Introduction

Contrairement au traitement séquentiel, La structure alternative ou conditionnelle permet d'exécuter ou non une série d'instruction selon la valeur d'une condition.

II. Structure d'un test

Il y'a deux formes possibles pour un test :

- La forme la plus simple :

```
Si Booléen Alors
        Instruction 1
        Instruction 2
        …
Finsi
```

- La forme la plus complexe :

```
Si Booléen Alors
        Instruction 1
        Instruction 2
        …
Sinon
        Instruction 3
        Instruction 4
        …
```

Finsi

Un **booléen** est une **expression** dont la valeur est VRAI ou FAUX. Cela peut donc être (il n'y a que deux possibilités) :

- Une **variable** (ou une expression) de type booléen
- Une **condition**

Exemple :

Allez tout droit jusqu'au prochain magasin

Si la rue est bloqué **Alors**

Tournez à droite

Avancez

Prenez la 2éme rue à gauche

Sinon

Continuez tout droit

Prenez le 3éme rue à droite

Finsi

III. Les conditions

Une condition est une comparaison. Elle est composée de trois éléments :

- Valeur
- **Opérateur de comparaison**
- Une autre valeur

Les valeurs peuvent être de n'importe quel type (numérique, caractères…).

Les opérateurs de comparaison possibles sont :

- Egal à : =
- Différent de : <>
- Strictement plus petit : =<
- Strictement plus grand : >=
- Plus petit ou égal à : <
- Plus grand ou égal à : >

Exemple :

```
Algorithme exemple
Variables
    A, B : entier
Début
    Ecrire("Donner la valeur de A :  ")
    Lire(A)
    Ecrire("Donner la valeur de B :  ")
    Lire(B)
    Si A > B Alors
        Ecrire("La valeur de A est plus grand que B  ")
    Sinon
        Ecrire("La valeur de A est plus petit que B  ")
    Finsi
Fin
```

IV. Exercices

1.1. Exercice 1

Écrire un algorithme qui demande un nombre à l'utilisateur, et l'informe ensuite si ce nombre est positif ou négatif.

1.2. Exercice 2

Ecrire un algorithme qui demande un nombre entier à l'utilisateur, puis qui teste et affiche s'il est divisible par 3

1.3. Exercice 3

Ecrire un algorithme qui permet de calculer la division entre deux nombres saisis par l'utilisateur. Afficher le message « impossible de faire la division » si le dénominateur égale à 0.

1.4. Exercice 4

Écrire un algorithme qui permet la résolution d'une équation du premier degré (une équation sous la forme ax+b=0)

1.5. Exercice 5

Ecrire un algorithme permettant de saisir deux nombres ainsi que la lettre représentant l'opération (s pour la somme, p pour produit).

V. Conditions composés

Certains problèmes exigent de formuler des conditions qui ne peuvent pas être exprimée sous la forme simple.

Dans le cas d'une condition composée, on utilise un opérateur logique. Les opérateurs logiques que nous avons à notre disposition 3 opérateurs logiques :

- **ET** : Pour qu'une **"condition1" ET "condition2"** soit vraie. Il faut que la condition1 soit vraie et même la condition2 soit vraie.

Condition 1	Condition2	Condition1 ET Condition2
Vrai	Vrai	Vrai
Vrai	Faux	Faux
Faux	Vrai	Faux
Faux	Faux	Faux

- **OU** : Pour qu'une **"condition1" OU "condition2"** soit vraie. Il faut que l'un des deux conditions soit vraie.

Condition 1	Condition2	Condition1 OU Condition2
Vrai	Vrai	Vrai
Vrai	Faux	Vrai
Faux	Vrai	Vrai
Faux	Faux	Faux

- **NON** : cet opérateur inverse une condition :
  - NON(condition1) est vraie si la condition1 est faux.
  - NON(condition1) est faux si la condition 1 est vraie.

Exemple :

```
Algorithme exemple
Variables
        A : entier
Début
        Ecrire("Donner la valeur de A :  ")
        Lire(A)
        Si A > 0 ET A < 10 Alors
                Ecrire("La valeur de A est est entre 0 et 10  ")
        Sinon
                Ecrire("La valeur de A n'est pas entre 0 et 10  ")
        Finsi
Fin
```

VI. Exercices

1.1. Exercice 1

Écrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si le produit est négatif ou positif. Attention toutefois, on ne doit pas calculer le produit !

1.2. Exercice 2

Ecrire un algorithme qui demande à l'utilisateur de saisir un caractère et affiche le message « **Entre A et M** » si le caractère saisi est entre A et M sinon il va afficher « **Le caractère saisi n'est pas entre A et M** »

1.3. Exercice 3

Écrire un algorithme qui lit deux valeurs entières (A et B) au clavier et qui affiche le signe de la somme de A et B sans faire l'addition.

VII. Les tests imbriquées

Les tests peuvent avoir un degré quelconque d'imbrications

Si condition1 **alors**
    Si condition2 **alors**
        instructionsA
    Sinon
        instructionsB
    Finsi
Sinon
    Si condition3 **alors**
        instructionsC
    Finsi
Finsi

Exemple :

```
Algorithme Test_signe
Variables n : entier
Début
        Ecrire ("entrez un nombre : ")
        Lire (n)
        Si (n < 0) alors
                Ecrire ("Ce nombre est négatif")
        Sinon
                Si (n = 0) alors
                        Ecrire ("Ce nombre est nul")
                Sinon
                        Ecrire ("Ce nombre est positif")
                Finsi
        Finsi
Fin
```

VIII. Exercices

1.4. Exercice 1

Donner les valeurs des a, b, c et d après chaque instruction de l'algorithme suivant :

```
Algorithme Penible
Variables
        a, b ,c ,d : entier
Début
         Ecrire("Saisir trois valeurs :")
        Lire(a)
        Lire(b)
        Lire(c)
        d ← 2
        Si a = 3 Alors
                a ← 2
```

```
            b ← (a +c) * d
        Sinon
            Si a = 0 Alors
                a ← 2
                c ← d * a
            Sinon
                c ← 2 + b
                d ← b - a
            Finsi
        Finsi
        Ecrire("A = ", a, "B = ",b, "C = ",c)
Fin
```

Cas 1 : a = 3, b = 2 , c = 1

Cas 2 : a = 0 , b = 4, c = 5

Cas 3 : a = 1, b = 5, c = 3

Après instruction	Valeurs des variables			
	a	b	c	d
i1				
i2				
i3				
i4				
i5				
i6				
i7				
i8				
i9				

1.5. Exercice 2 :

Écrire un algorithme qui permet la résolution d'une équation du second degré (une équation sous la forme $ax^2+bx+c=0$)

1.6. Exercice 3 :

Calculer le lendemain d'une journée donnée (jour, mois, année)" On ne tiendra pas compte ici des années bissextiles, le mois de février aura toujours 28 jours.

1.7. Exercice 4 :

Calculer la durée d'un trajet connaissant l'heure de départ et d'arrivée". On se contente des heures et des minutes, la durée totale ne dépassera jamais 24 heures.

1.8. Exercice 5 :

Le prix de photocopies dans une reprographie varie selon le nombre demandé: 0,5 DH la copie pour un nombre de copies inférieur à 10, 0,4DH pour un nombre compris entre 10 et 20 et 0,3DH au-delà.

Ecrivez un algorithme qui demande à l'utilisateur le nombre de photocopies effectuées, qui calcule et affiche le prix à payer

IX. Les structures alternatives Multiples

Pour éviter un lourd algorithme contenant plusieurs briques de **Si..Finsi** pour plus de lisibité on a adopté une structure plus simple pour traiter plusieurs conditions à la fois cette structure est : **Selon .. FinSelon**

Exemple :

Selon (variable_a_tester /condition)
cas cas1 : BlocInstruction1
cas cas2 : BlocInstruction2

.....

Autrement/sinon : BlocInstruction

Finselon

Exemple : Exercice jours de la semaine

Suite Chapitre 5 – les structures répétitives

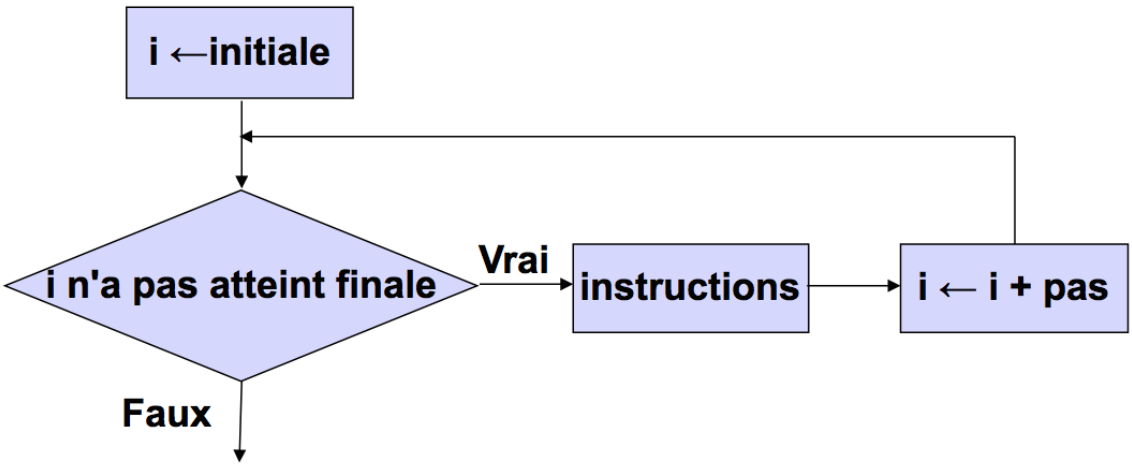
I. Boucle Pour

La syntaxe de la boucle Pour est :

Pour compteur **allant de** initiale **à** finale par **pas** valeur du pas

Liste des instructions

FinPour



Remarque : le nombre d'itérations dans une boucle Pour est connu avant le début de la boucle

- **Compteur** est une variable de type entier (ou caractère). Elle doit être déclarée.
- **Initiale et finale** peuvent être des valeurs, des variables définies avant le début de la boucle ou des expressions de même type que compteur.

- **Pas** est un entier qui peut être positif ou négatif. **Pas** peut ne pas être mentionné, car par défaut sa valeur est égal à 1. Dans ce cas, le nombre d'itérations est égal à finale - initiale+ 1.

Déroulement des boucles Pour :

1. La valeur initiale est affectée à la variable compteur
2. On compare la valeur du compteur et la valeur de finale :
    a. Si la valeur du compteur est > à la valeur finale dans le cas d'un pas positif (ou si compteur est < à finale pour un pas négatif), on sort de la boucle et on continue avec l'instruction qui suit FinPour
    b. Si compteur est <= à finale dans le cas d'un pas positif (ou si compteur est >= à finale pour un pas négatif), instructions seront exécutées
        i. Ensuite, la valeur de compteur est incrémentée de la valeur du pas si pas est positif (ou décrémenté si pas est négatif)
        ii. On recommence l'étape 2 : La comparaison entre compteur et finale est de nouveau effectuée, et ainsi de suite …

Exemple 1 : Pour avec pas positif

Algorithme exemple_1
Variables

```
        n, i : entier
Debut
    Ecrire (" Entrez la valeur de n ")
    Lire (n)
    Pour i allant de 1 à n
        Ecrire ("Bonjour tout le monde")
    FinPour
Fin
```

Exemple 2 : Pour avec pas négatif

```
Algorithme exemple_1
Variables
        n, i : entier
Debut
    Ecrire (" Entrez la valeur de n ")
    Lire (n)
    Pour i allant de n à 1 pas -1
        Ecrire ("Bonjour tout le monde")
    FinPour
Fin
```

II. Les boucles imbriquées

Les instructions d'une boucle peuvent être des instructions itératives. Dans ce cas, on aboutit à des **boucles imbriquées.**

```
Pour i allant de 1 à 5
```

```
        Pour j allant de 1 à i
            écrire("O")
        FinPour
        écrire("X")
FinPour
```

Exécution du programme :

OX

OOX

OOOX

OOOOX

OOOOOX

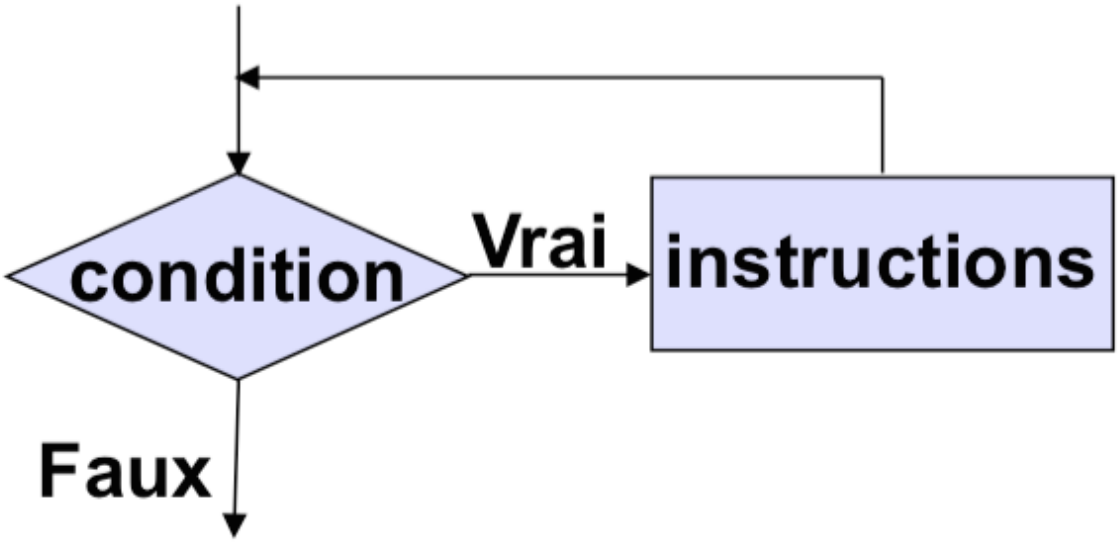
III. Les boucles Tant que

La syntaxe d'une boucle Tant que est :

```
Tant que (condition)
        Liste des instructions
FinTant que
```

- La condition (dite condition de contrôle de la boucle) est évaluée avant chaque itération.
- Si la condition est vraie, on exécute instructions (corps de la boucle), puis, on retourne tester la condition. Si elle est encore vraie, on répète l'exécution, …

- Si la condition est fausse, on sort de la boucle et on exécute l'instruction qui est après FinTantQue



Exemple 1 : Contrôle de saisie d'une lettre majuscule jusqu'à ce que le caractère entré soit valable.

```
Algorithme Contrôle_Saisie
Variable
        C : caractère
Debut
        Ecrire (" Entrez une lettre majuscule ")
        Lire (C)
        TantQue (C < 'A' OU C > 'Z')
                Ecrire ("Saisie erronée. Recommencez")
                Lire (C)
```

```
        FinTantQue
        Ecrire ("Saisie valable")
    Fin
```

Exemple 2 : Un algorithme qui détermine le premier nombre entier N tel que la somme de 1 à N dépasse strictement 100.

```
    Algorithme somme_à_100
    Variables
        som, i : entier
    Début
        i ← 0
        som← 0
        TantQue (som <=100)
            i ← i+1
            som ← som+i
        FinTantQue
        Ecrire (" La valeur cherchée est N=  ", i)
    Fin
```

Remarques :

- Le nombre d'itérations dans une boucle TantQue n'est pas connu au moment d'entrée dans la boucle. Il dépend de l'évolution de la valeur de condition

- Une des instructions du corps de la boucle doit absolument changer la valeur de condition de vrai à faux (après un certain nombre d'itérations), sinon le programme tourne indéfiniment

➔ **Il faut toujours faire attention aux boucles infinies**

- Exemple de boucle infinie :

```
i ← 2
TantQue  (i > 0)
        i ←  i+1
FinTantQue
```

Liens entre Pour et tant que

La boucle Pour est un cas particulier de Tant Que (cas où le nombre d'itérations est connu et fixé) . Tout ce qu'on peut écrire avec Pour peut être remplacé avec TantQue (la réciproque est fausse)

```
Pour compteur allant de initiale à finale par pas valeur du pas
        instructions
FinPour
```

peut être remplacé par : (cas d'un pas positif)

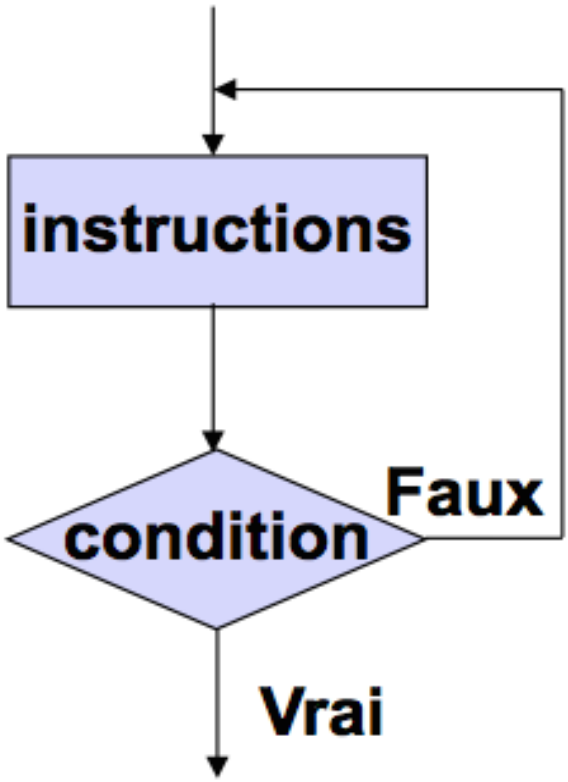
```
        compteur ← initiale
        TantQue  compteur <= finale
                        instructions
```

```
                compteur ← compteur+pas
    FinTantQue
```

IV. Les boucles Répéter … jusqu'à …

La syntaxe de l'instruction jusqu'à est :

```
Répéter
        instructions

Jusqu'à  condition
```



- Condition est évaluée après chaque itération

- Les instructions entre *Répéter* et *jusqu'à* sont exécutées au moins une fois et leur exécution est répétée jusqu'à ce que condition soit vrai (tant qu'elle est fausse.

V. Exercices d'application

Exercice 1 :

Écrire un algorithme qui demande un nombre de départ, et qui ensuite écrit la table de multiplication de ce nombre, présentée comme suit (cas où l'utilisateur entre le nombre 5) :

Table de 5 :

5 x 1 = 5

5 x 2 = 10

5 x 3 = 15

…

5 x 10 = 50

Exercice 2 :

Écrire un algorithme qui demande un nombre de départ, et qui calcule la somme des entiers jusqu'à ce nombre. Par exemple, si l'on entre 5, le programme doit calculer :

1 + 2 + 3 + 4 + 5 = 15

Exercice 3 :

Écrire un algorithme qui demande un nombre de départ, et qui calcule sa factorielle.

NB : la factorielle de 8, notée 8 ! vaut 1 x 2 x 3 x 4 x 5 x 6 x 7 x 8

Exercice 4 :

Écrire un algorithme qui demande successivement 10 nombres à l'utilisateur, et qui lui dise ensuite quel était le plus grand parmi ces 10 nombres et sa position :

Entrez le nombre numéro 1 : 12

Entrez le nombre numéro 2 : 14

…

Entrez le nombre numéro 10 : 6

Le plus grand de ces nombres est : 14, sa position : 2

Exercice 5

Écrire un algorithme qui demande successivement des nombres à l'utilisateur, et qui lui dise ensuite quel était le plus grand parmi ces nombres et quel était sa position. La saisie des nombres s'arrête lorsque l'utilisateur entre un zéro.

Exercice 6 :

Écrire un algorithme qui demande successivement des nombres à l'utilisateur, et qui calcule leur moyenne. La saisie des nombres s'arrête lorsque l'utilisateur entre un zéro.

Chapitre 6 – Les fonctions et les procédures

I. Introduction

Certains problèmes conduisent à des programmes longs, difficiles à écrire et à comprendre. On les découpe en des parties appelées **sous-programmes** ou **modules.**

Les **fonctions** et les **procédures** sont des modules (groupe d'instructions) indépendants désignés par un nom. Elles ont plusieurs **intérêts** :

- Permettent de **"factoriser" les programmes**, c.-à-d. de mettre en commun les parties qui se répètent.
- Permettent une **structuration** et une **meilleure lisibilité** des programmes.
- **Facilitent la maintenance** du code (il suffit de modifier une seule fois).
- Ces procédures et fonctions peuvent éventuellement être **réutilisées** dans d'autres programmes.

II. Les fonctions

Le **rôle** d'une fonction en programmation est similaire à celui d'une fonction en mathématique : elle **retourne un résultat à partir des valeurs des paramètres**

Une fonction est un mini-programme qu'on déclare dans la partie réservée aux variables, ce afin de pouvoir utiliser les variables globales. Étant donné qu'il s'agit d'un bloc à part entière, elle possèdera éventuellement un en-tête, une série de traitements, et une gestion des résultats tout comme l'algorithme qui la contient. En outre, une fonction

peut également recevoir des arguments qui lui seront alors passés en paramètres.

La syntaxe de déclaration d'une fonction :

FONCTION nom_de_la_fonction **[(liste des paramètres : type)]** : **type_fonction**

Variable : liste des variables

DEBUT {corps de la fonction}

{la liste ne doit pas être vide}

retourne….

FINFONCTION

Remarques :

- Pour le choix d'un nom de fonction il faut respecter les mêmes règles que celles pour les noms de variables
- **Type_fonction** est le type du résultat retourné
- L'instruction **retourne** sert à retourner la valeur du résultat
- La liste des paramètres est facultative. Mais quand elle existe, ces paramètres sont déclarés de la même façon qu'on déclare une série de variables de différents types.
- Les variables déclarées à l'intérieur de la fonction sont inutilisables à l'extérieur du bloc.

Exemple :

```
Algorithme Exemple_1
Variables x, y, Res : entier
    FONCTION Somme(a : entier, b : entier) : entier
        Variable s : entier
    Début
        s ← a + b
        retourne s
    FinFonction
Début
    Ecrire("Donner la valeur de x :")
    Lire(x)
    Ecrire("Donner la valeur de y :")
    Lire(y)
    Res = Somme(x , y)
    Ecrire(x,"+",y,"=",res)
    Res = Somme(2, 4)
    Ecrire("2+ 4=",res)
    Res = Somme(6,7)
    Ecrire("6+ 7=",res)
Fin
```

Exercices d'application

Exercice 1 :

Ecrire une fonction qui retourne la somme de deux nombres passé en paramètre.

Exercice 2 :

Ecrire une fonction qui retourne le nombre maximal de deux nombres passés en paramètre.

Exercice 3 :

Ecrire une fonction booléenne qui permet de retourner si un nombre est pair ou non.

Exercice 4 :

Ecrire une fonction booléenne qui permet de retourner si un nombre est premier ou non.

Exercice 5

Ecrire une fonction qui retourne le factoriel d'un nombre passé en paramètre.

Exercice 6 :

Ecrire une fonction qui retourne la moyenne de trois nombre passé en paramètre.

Exercice 7 :

Écrire la fonction NCHIFFRES du type entier qui obtient une valeur entière N (positive) du type long entier comme paramètre et qui fournit le nombre de chiffres de N comme résultat.

Exemple:

Introduire un nombre entier : 6457392

Le nombre 6457392 a 7 chiffres.

Exercice 8 :

Ecrire une fonction qui reçoit en arguments 2 nombres réels et un caractère et qui fournit un résultat correspondant à l'une des 4 opérations appliquées à ses deux premiers arguments, en fonction de la valeur de ce dernier, à savoir : addition pour le caractère +, soustraction pour le caractère -, multiplication pour * et division pour / (tout autre caractère sera interprété comme une addition). On ne tiendra pas compte des risques de la division par 0.

III. Les procédures

Dans certains cas, on peut avoir besoin de répéter une tache dans plusieurs endroits du programme, mais que dans cette tache on ne calcule pas de résultats ou qu'on calcule plusieurs résultats à la fois

Dans ces cas on ne peut pas utiliser une fonction, on utilise une **procédure**

Une **procédure** est un sous-programme semblable à une fonction mais qui **ne retourne rien**

Une procédure s'écrit en dehors du programme principal sous la forme :

```
Procédure nom_procédure (paramètres et leurs types)
    Variable : Liste des variable
    Instructions constituant le corps de la procédure
FinProcédure
```

Remarque : une procédure peut ne pas avoir de paramètres

L'appel d'une procédure

L'appel d'une procédure, se fait dans le programme principale ou dans une autre procédure par une instruction indiquant le nom de la procédure :

```
Procédure exemple_proc (…)
    …
FinProcédure
Algorithme exepmleAppelProcédure
Début
    exemple_proc (…)
    …
Fin
```

Remarque : contrairement à l'appel d'une fonction, on ne peut pas affecter la procédure appelée ou l'utiliser dans une expression. L'appel d'une procédure est une instruction autonome

Les paramètres d'une procédure

Les paramètres servent à échanger des données entre le programme principale (ou la procédure appelante) et la procédure appelée

Les paramètres placés dans la déclaration d'une procédure sont appelés **paramètres formels**. Ces paramètres peuvent prendre toutes les valeurs possibles mais ils sont abstraits (n'existent pas réellement)

Les paramètres placés dans l'appel d'une procédure sont appelés **paramètres effectifs**. Ils contiennent les valeurs pour effectuer le traitement

Le nombre de paramètres effectifs doit être égal au nombre de paramètres formels. L'ordre et le type des paramètres doivent correspondre

Transmission des paramètres

Il existe deux modes de transmission de paramètres dans les langages de programmation :

- **La transmission par valeur** : les valeurs des paramètres effectifs sont affectées aux paramètres formels correspondants au moment de l'appel de la procédure. Dans ce mode le paramètre effectif ne subit aucune modification
- **La transmission par adresse (ou par référence)** : les adresses des paramètres effectifs sont transmises à la procédure appelante. Dans ce mode, le paramètre effectif subit les mêmes modifications que le paramètre formel lors de l'exécution de la procédure

Remarque : le paramètre effectif doit être une variable (et non une valeur) lorsqu'il s'agit d'une transmission par adresse

En algorithme, on va préciser explicitement le mode de transmission dans la déclaration de la procédure

Exemple 1 :

Procédure incrementer1 (**x** : entier **par valeur, y** : entier **par adresse**)

x ← x+1

y ← y+1

FinProcédure

Algorithme Test_incrementer1

variables n, m : entier

Début

n ← 3

m ← 3

incrementer1(n, m) **résultat :**

écrire (" n= ", n, " et m= ", m) **n=3 et m=4**

Fin

Remarque : l'instruction x ← x+1 n'a pas de sens avec un passage par valeur

Exemple 2 :

Procédure qui calcule la somme et le produit de deux entiers :

Procédure SommeProduit (x,y: entier **par valeur,** som, prod : entier **par adresse**)

som ← x+y

prod ← x*y

FinProcédure

Procédure qui échange le contenu de deux variabales :

Procédure Echange (**x** : réel **par adresse, y** : réel **par adresse**)

variables z : réel

z ← x

x ← y

y ← z

FinProcédure

Les variables locales et globales

On peut manipuler 2 types de variables dans un module (procédure ou fonction) : des **variables locales** et des **variables globales**. Elles se distinguent par ce qu'on appelle leur **portée** (leur "champ de définition", leur "durée de vie")

Une **variable locale** n'est connue qu'à l'intérieur du module ou elle a été définie. Elle est créée à l'appel du module et détruite à la fin de son exécution

Une **variable globale** est connue par l'ensemble des modules et le programme principale. Elle est définie durant toute l'application et peut être utilisée et modifiée par les différents modules du programme

Exercices d'application

Exercice 1 :

Écrire :

- Une procédure, nommée f1, se contentant d'afficher bonjour (elle ne possèdera aucun argument).
- Une procédure, nommée f2, qui affiche bonjour un nombre de fois égale à la valeur reçue en argument (entier).
- Une fonction, nommée, f3, qui fait la même chose que f2, mais qui, de plus, renvoie la valeur (entier) 0.

Exercice 2 :

Écrire un programme dans lequel une procédure prend en paramètres un prénom et un âge. La procédure affichera ensuite un message disant 'BONJOUR', PRENOM, puis indiquera s'il s'agit d'un enfant (-20), d'un adulte (-50), ou d'une personne âgée.

Exercice 3 :

Écrire une procédure qui prend en paramètres 2 entiers p et n rentrés par l'utilisateur dans le bloc principal, et qui calcule ensuite la valeur p^n. p devra changer de valeur pour prendre cette nouvelle valeur.

Exercice 4 :

Ecrire une procédure qui affiche le PGCD de deux nombres passés en paramètres.

Exercice 5 :

Ecrire une procédure qui donne la solution d'une équation de 2^{éme} degré.

Chapitre 7 : Les Fonctions et les méthodes

I. Les Fonctions :

En programmation, les fonctions sont très utiles pour réaliser plusieurs fois la même opération au sein d'un programme. Elles rendent également le code plus lisible et plus clair en le fractionnant en blocs logiques.

Il nous est arrivé de travailler avec plusieurs fonctions sans qu'on nous rendons compte exemple : print() , input() , int() , type() , str() , range() , … donc les fonctions sont des pseudo-programmes qu'on définit à l'avance et qui font un traitement spécifique qu'on va utiliser après dans notre programme principale.

En python comme dans d'autres langages de programmation il existe plusieurs fonctions . Donc **d'où vient ces fonctions ?**

En général, les fonctions proviennent d'au moins trois endroits :

• à partir de Python lui-même - de nombreuses fonctions (comme print ()) font **partie intégrante de Python** et sont toujours disponibles sans effort supplémentaire de la part du programmeur ; nous appelons ces fonctions des **fonctions intégrées** ;

• à partir des **modules préinstallés** de Python - de nombreuses fonctions, très utiles, mais utilisées beaucoup moins souvent que celles intégrées, sont disponibles dans un certain nombre de modules installés avec Python ; l'utilisation de ces fonctions nécessite quelques étapes supplémentaires de la part du programmeur afin de les rendre entièrement accessibles (nous vous en parlerons dans un moment);

• **directement à partir de votre code** - vous pouvez écrire vos propres fonctions, les placer dans votre code et les utiliser librement ;

• il existe une autre possibilité, mais elle est liée aux classes, nous allons donc l'omettre pour l'instant.

1- Définition d'une fonction :

Pour définir une fonction, Python utilise le mot-clé **def :**

```
def message():
    print("Hello world")
```

Si on souhaite qu'elle renvoie ou retourne un résultat on ajoute le mot clé **return** :

```
def calcul_somme(x,y):
    return x+y
```

Notez que la syntaxe de **def** utilise les deux points comme les boucles for et while ainsi que les tests if, un bloc d'instructions est donc attendu. De même que pour les boucles et les tests, l'indentation de ce bloc d'instructions (qu'on appelle le corps de la fonction) est obligatoire

2- L'invocation de la Fonction (Appel) :

La fonction se déclare généralement avant le programme principale et pour l'appeler il suffit d'invoquer son nom au sein du programme principale

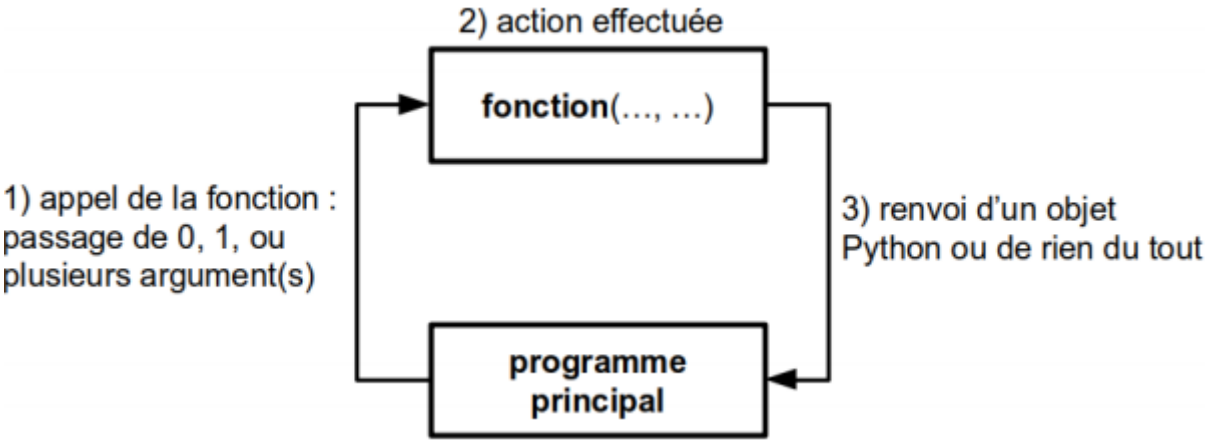
Exemple :

```
def message():
    print("Hello world")

def calcul_somme(x,y):
    return x+y

print("programme principale :")
message()
a=int(input("Entrez a :"))
b=int(input("Entrez b :"))
print("La somme est :",calcul_somme(a,b))
print("Bye")
```

Résultat :

```
= RESTART: C:\Users\TOSHIBA\AppData\Local\Programs\Python\Python38\serie_srepeti
ttives.py
programme principale :
Hello world
Entrez a : 5
Entrez b : 3
La somme est : 8
Bye
```

Comment ça fonctionne ?



Dans l'exemple précédent, nous avons passé un argument à la fonction calcul_somme() qui nous a renvoyé (ou retourné) une valeur que nous avons immédiatement affichée à l'écran avec l'instruction print(). Que veut dire valeur renvoyée ? Et bien cela signifie que cette dernière est récupérable dans une variable

Dans ce cas la fonction, message() se contente d'afficher la chaîne de caractères "Hello world " à l'écran. Elle ne prend aucun argument et ne renvoie rien. Par conséquent, cela n'a pas de sens de vouloir récupérer dans une variable le résultat renvoyé par une telle fonction.

3- Passage d'arguments :

Le nombre d'arguments que l'on peut passer à une fonction est variable. Nous avons vu ci-dessus des fonctions auxquelles on passait 0 ou 1 argument. Dans les chapitres précédents, vous avez rencontré des fonctions internes à Python qui

prenaient au moins 2 arguments. Souvenez-vous par exemple de range(1, 10) ou encore range(1, 10, 2). Le nombre d'argument est donc laissé libre à l'initiative du programmeur qui développe une nouvelle fonction. Une particularité des fonctions en Python est que vous n'êtes pas obligé de préciser le type des arguments que vous lui passez, dès lors que les opérations que vous effectuez avec ces arguments sont valides. Python est en effet connu comme étant un langage au « typage dynamique », c'est-à-dire qu'il reconnaît pour vous le type des variables au moment de l'exécution.

Exemple :

```
>>> def fois(x,y):
        return x*y

>>> fois(2,3)
6
>>>
>>> fois(3.1456,5.892)
18.5338752
>>>
>>> fois('to',5)
'tototototo'
>>>
>>> fois([1,3],2)
[1, 3, 1, 3]
```

L'opérateur * reconnaît plusieurs types (entiers, floats, chaînes de caractères, listes). Notre fonction fois() est donc capable d'effectuer des tâches différentes.

4- Renvoi des resultats :

Un énorme avantage en Python est que les fonctions sont capables de renvoyer plusieurs objets à la fois, comme dans cette fraction de code :

```
>>> def carre_cube(x):
        return x**2,x**3

>>> carre_cube(2)
(4, 8)
```

En réalité Python ne renvoie qu'un seul objet, mais celui-ci peut être séquentiel, c'est-à-dire contenir lui-même d'autres objets. Dans notre exemple Python renvoie un objet de type tuple, type que nous verrons plus tard.

Notre fonction pourrait tout autant renvoyer une liste :

```
>>> def carre_cube(x):
        return [x**2,x**3]

>>> carre_cube(5)
[25, 125]
```

Remarque 1 :

si une fonction n'est pas destinée à produire un résultat, l' **utilisation de l'** return **instruction n'est pas obligatoire** - elle sera exécutée implicitement à la fin de la fonction.

Quoi qu'il en soit, vous pouvez l'utiliser pour **terminer les activités d'une fonction à la demande** , avant que le contrôle n'atteigne la dernière ligne de la fonction.

Remarque 2 :

On peut utiliser le mot clé return pour interrompre l'exécution d'une fonction dans ce cas il suffit d'écrire return ou return 0

Exemple :

```
def happy_new_year(corona):
    print("trois..")
    print("deux..")
    print("Un..")
    if not corona:
        return
    print("Happy New Year")

reponse=input("Est ce qu'il ya toujours corona ?")
if reponse =="OUI":
    corona=False
else:
    corona=True
happy_new_year(corona)
```

5- Arguments positionnels et arguments par mot-clé :

- **Arguments positionnels :**

Jusqu'à maintenant, nous avons systématiquement passé le nombre d'arguments que la fonction attendait. Que se passe-t-il si une fonction attend deux arguments et que nous ne lui en passons qu'un seul ?

```
>>> def fois(x,y):
        return x*y

>>> fois(2,3)
6
>>> fois(5)
Traceback (most recent call last):
  File "<pyshell#23>", line 1, in <module>
    fois(5)
TypeError: fois() missing 1 required positional argument: 'y'
```

On constate que passer un seul argument à une fonction qui en attend deux conduit à une erreur.

Définition : Lorsqu'on définit une fonction def fct(x, y): les arguments x et y sont appelés arguments positionnels (en anglais positional arguments). Il est strictement obligatoire de les préciser lors de l'appel de la fonction. De plus, il est nécessaire de respecter le même ordre lors de l'appel que dans la définition de la fonction. Dans l'exemple ci-dessus, 2 correspondra à x et 3 correspondra à y. Finalement, tout dépendra de leur position, d'où leur qualification de positionnel.

- **Argument avec mot-clé :**

Python propose une autre convention pour le passage d'arguments, où la signification de l'argument est dictée par son nom, et non par sa position - c'est ce qu'on appelle le passage d'argument par mot clé.

Exemple :

```
def Introduction(firstname , lastname):
    print("Bonjour Je suis :",firstname,lastname)


print("Programme Principale")
Introduction("joairia","lafhal")
Introduction(firstname="joairia",lastname="lafhal")
Introduction(lastname="lafhal",firstname="joairia")
```

Resultat :

```
= RESTART: C:\Users\TOSHIBA\AppData\Loc
ttives.py
Programme Principale
Bonjour Je suis : joairia lafhal
Bonjour Je suis : joairia lafhal
Bonjour Je suis : joairia lafhal
```

Le concept est clair - les valeurs transmises aux paramètres sont précédées du nom des paramètres cibles, suivi du signe =.

La position n'a pas d'importance ici - la valeur de chaque argument connaît sa destination sur la base du nom utilisé.

Remarque : je dois respecter l'écriture du mot-clé comme je l'ai écrit dans la définition du la fonction je dois l'écrire dans l'appel

- **Mélanger les arguments positionnels et avec mot clé :**

Vous pouvez mélanger les deux modes si vous le souhaitez - il n'y a qu'une seule règle incassable: **vous devez mettre les arguments positionnels avant les arguments mot-clé**.

Pour vous montrer comment cela fonctionne, nous utiliserons la fonction simple à trois paramètres suivants :

```
def adding(a,b,c):
    print(a,"+",b,"+",c,"=",a+b+c)
```

Son but est d'évaluer et de présenter la somme de tous ses arguments.

La fonction, lorsqu'elle est invoquée de la manière suivante :

adding(1, 2, 3)

affichera :

1 + 2 + 3 = 6

C'était - comme vous pouvez le soupçonner - un pur exemple de **passage d'arguments positionnels**.

Bien sûr, vous pouvez remplacer une telle invocation par une variante purement mot-clé, comme celle-ci :

adding(c = 1, a = 2, b = 3)

Notre programme affichera une ligne comme celle-ci :

2 + 3 + 1 = 6

Notez l'ordre des valeurs.

Essayons maintenant de mélanger les deux styles.

Regardez l'invocation de fonction ci-dessous :

adding(3, c = 1, b = 2) Analysons-le:

• l'argument (3) pour le paramètre a est passé en utilisant la manière positionnelle ;

• les arguments pour c et b sont spécifiés comme mots-clés.

Voici ce que vous verrez dans la console :

3 + 2 + 1 = 6

Soyez prudent et méfiez-vous des erreurs. Si vous essayez de passer plusieurs valeurs à un seul argument, vous n'obtiendrez qu'une erreur d'exécution.

Regardez l'invocation ci-dessous - il semble que nous ayons essayé de définir a deux fois :

```
>>> adding(3,a=2,b=1)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    adding(3,a=2,b=1)
TypeError: adding() got multiple values for argument 'a'
```

6- Les fonctions paramétrées (valeur par défaut) :

Il arrive parfois que les valeurs d'un paramètre particulier soient utilisées plus souvent que d'autres. Ces arguments peuvent avoir leurs **valeurs par défaut (prédéfinies)** prises en considération lorsque leurs arguments correspondants ont été omis.

Exemple :

```
def Introduction(lastname, firstname ="Mohamed"):
    print("Bonjour Je suis :",firstname,lastname)


print("Programme Principale")
Introduction("Lafhal")
Introduction("lafhal","joairia")
Introduction(firstname="joairia",lastname="lafhal")
Introduction(lastname="lafhal")
```

Resultat :

```
= RESTART: C:\Users\TOSHIBA\AppDat
ttives.py
Programme Principale
Bonjour Je suis : Mohamed Lafhal
Bonjour Je suis : joairia lafhal
Bonjour Je suis : joairia lafhal
Bonjour Je suis : Mohamed lafhal
```

On remarque que lorsqu'on appelé la fonction avec un seul paramètre ça n'a pas créé des problèmes car python a déjà une valeur par défaut pour le 2ème argument .

7- Les variables locales et globales :

Lorsqu'on manipule des fonctions, il est essentiel de bien comprendre comment se comportent les variables. Une variable est dite locale lorsqu'elle est créée dans une fonction. Elle n'existera et ne sera visible que lors de l'exécution de ladite fonction. Une variable est dite globale lorsqu'elle est créée dans le programme principal. Elle sera visible partout dans le programme.

Exemple :

```
def carre(x):
    return x**2

print("Programme principale")
z= int(input("Entrez un nombre :" ))
print(x)
y=carre(x)
print(y)
```

Resultat :

```
= RESTART: C:\Users\TOSHIBA\AppData\Local\P
ttives.py
Programme principale
Entrez un nombre :5
Traceback (most recent call last):
  File "C:\Users\TOSHIBA\AppData\Local\Prog
ves.py", line 6, in <module>
    print(x)
NameError: name 'x' is not defined
```

Python ne connait pas la variable **x** car elle est créé dans la fonction et manipulée dans la fonction est détruite une fois le traitement de la fonction est terminé donc les variables locales n'ont aucun rôle en dehors la fonction.

Par contre les variables **z** et **y** sont des variables global elles sont connu partout dans le programme principale et même dans une fonction

Exemple :

```
def carre(x):
    print(a)
    return x**2

a=5
print("Programme principale")
z= int(input("Entrez un nombre :" ))
y=carre(z)
print(y)
```

Resultat :

```
= RESTART: C:\Users\TOSHIBA\AppData\Local\Pr
ttives.py
Programme principale
Entrez un nombre :6
5
36
```

Remarque : Python reconnait les variables globales dans la fonction mais on ne peut pas modifier la valeur de la variable dans le programme principale

Exemple :

```
def carre(x):

    print("Incrementation de a dans la fonction :")
    a+=1
    print(a)
    return x**2

a=5
print("Programme principale")
z= int(input("Entrez un nombre :" ))
y=carre(z)
print("a=",a)
print("y=",y)
```

Resultat :

```
= RESTART: C:\Users\TOSHIBA\AppData\Local\Programs\Python\Python38\
ttives.py
Programme principale
Entrez un nombre :6
Incrementation de a dans la fonction :
Traceback (most recent call last):
  File "C:\Users\TOSHIBA\AppData\Local\Programs\Python\Python38\ser
ves.py", line 11, in <module>
    y=carre(z)
  File "C:\Users\TOSHIBA\AppData\Local\Programs\Python\Python38\ser
ves.py", line 4, in carre
    a+=1
UnboundLocalError: local variable 'a' referenced before assignment
```

- **Le mot clé « global » :**

Il existe une méthode Python spéciale qui peut **étendre la portée d'une variable d'une manière qui inclut les corps des fonctions** (même si vous voulez non seulement lire les valeurs, mais aussi les modifier).

Un tel effet est provoqué par un mot-clé nommé global :

global name

global name1, name2, ...

L'utilisation de ce mot-clé dans une fonction avec le nom (ou les noms séparés par des virgules) d'une ou de plusieurs variables, force Python à s'abstenir de créer une nouvelle variable à l'intérieur de la fonction - celle accessible de l'extérieur sera utilisée à la place.

En d'autres termes, ce nom devient global (il a une **portée globale** et peu importe qu'il fasse l'objet d'une lecture ou d'une affectation).

Exemple :

```
def carre(x):
    global a
    print("Incrementation de a dans la fonction :")
    a+=1
    print(a)
    return x**2

a=5
print ("a avant fonction : ",a)
z= int(input("Entrez un nombre :" ))
y=carre(z)
print("a apres la fonction ",a)
print("y=",y)
```

Resultat :

```
= RESTART: C:\Users\TOSHIBA\AppData\Local\Prog
ttives.py
a avant fonction :  5
Entrez un nombre :6
Incrementation de a dans la fonction :
6
a apres la fonction  6
y= 36
```

8- La valeur None :

Il n'y a que deux types de circonstances qui None peuvent être utilisées en toute sécurité :

• lorsque vous l'**affectez à une variable** (ou le retournez comme résultat d'une **fonction**)

• lorsque vous le **comparez à une variable** pour diagnostiquer son état interne.

Comme ici:

```
value = None
if value is None:
print("Sorry, you don't carry any value")
```

N'oubliez pas ceci : si une fonction ne retourne pas une certaine valeur en utilisant une return clause d'expression, on suppose qu'elle **renvoie implicitement** None. (On peut l'utiliser dans l'exercice 6 pour tester si a et b sont saisis)

9- Les Fonctions récursives :

En mathématiques, une suite (Un) n∈N est récurrente lorsque le terme un+1 est une fonction du terme un. En informatique, une fonction f est récursive lorsque la définition de f utilise des valeurs de f. Chaque fonction récursive est construite sur une relation de récurrence.

Exemple :

Le factoriel : La fonction factorielle est une fonction récursive car elle appelle elle-même dans le traitement

Si on souhaite calculer le factoriel de 5 c'est

5 != 5*4*3*2*1

Et 4*3*2*1 c'est le factoriel de 4

Donc 5 !=5*4 !

Et donc 5 !=5*4*3 ! est ça continue comme ça donc en peut écrire

Exemple :

```
def factoriel(x):
    if x==0:
        return 1
    return x*factoriel(x-1)

print("Programme principale :" )
a=int(input("Entrez un nombre :"))
print("le resultat est :",factoriel(a))
```

Résultat :

```
= RESTART: C:\Users\TOSHIBA\AppData\Loca
ttives.py
Programme principale :
Entrez un nombre :5
le resultat est : 120
```

10- Passage par adresse et passage par valeur :

En python c'est toujours le passage des paramètres est par valeur pour les variables ça se fait automatiquement donc on n'aura pas de problèmes en python par contre il faut faire attention dans les autres langages de programmation

Exemple :

```
def f(x):
    x+=1
    print("dans la fonction:",x)

print("Programme principale :" )
a=2
print("a avant la fonction:",a)
f(a)
print("a apres la fonction:",a)
```

Resultat :

```
= RESTART: C:\Users\TOSHIBA\AppData\Lo
ttives.py
Programme principale :
a avant la fonction: 2
dans la fonction: 3
a apres la fonction: 2
```

On remarque que la modification de la variables ne se propage pas en dehors de la fonction est donc le passage des variables en parametre se fait automatiquement par valeur

11- Exercices

Exercice 1 :

créez une fonction gen_pyramide() à laquelle vous passez un nombre entier N et qui renvoie une pyramide de N lignes sous forme de chaîne de caractères. Le programme principal demandera à l'utilisateur le nombre de lignes souhaitées (utilisez pour cela la fonction input()) et affichera la pyramide à l'écran.

Exercice 2 :

Ecrire une fonction qui accepte comme paramètre le jour, le mois et l'année qui détermine si la date est correcte.

Exercice 3 :

Ecrire une fonction **chiffrePorteBonheur(nb)** qui permet de déterminer si un nombre entier **nb** est chiffre porte Bonheur ou non.

Un nombre **chiffre porte Bonheur** est un nombre entier qui, lorsqu'on ajoute les **carrés** de **chacun** de ses chiffres, puis les carrés des chiffres de ce **résultat** et ainsi de suite jusqu'à l'obtention d'un nombre à un seul chiffre égal à 1 (un).

Le calcule s'arrête lorsque le chiffre devient inférieur à 10

le résultat doit être comme suit :

```
Nombre de départ: 913
9^2=81
1^2=1
3^2=9
Nouveau: 81+1+9=91
9^2=81
1^2=1
Nouveau: 81+1=82
8^2=64
2^2=4
Nouveau: 64+4=68
6^2=36
8^2=64
Nouveau: 36+64=100
1^2=1
0^2=0
0^2=0
Nouveau: 1+0+0=1
Le chiffre: 913 est un porte boneur
```

Exercice 4 :

Définir une fonction qui renvoie les nombres parfaits qui se trouve entre deux nombres entiers a et b. tel que a < b.

Exercice 5 :

Définir une fonction qui calcul le nombre des mots dans une phrase passée en paramètre