Algorithmique

1. **Les instructions**
2. **Les variables**
   - **Entiers, réels, chaines et booléens**
   - **Opérateurs logiques**
3. **Les opérateurs de comparaisons**
4. **Structures de contrôle**
   - **Conditions**
   - **Bloc de code**
   - **boucles pour et tant que**
5. **Création de fonctions**
6. **Les bases de JavaScript**
7. **Les fonctions internes avec JavaScript**

Notion d’algorithmique

- Les premiers algorithmes remontent a l'antiquité. Par exemple l'algorithme de calcul du plus grand commun diviseur de deux nombres, appelé maintenant "algorithme d'Euclide". Il s'agissait en général de méthodes de calcul.
- Notez qu'a l‘époque, on vous demandait juste d'appliquer la méthode sans vous tromper, on ne vous a pas explique pourquoi cette méthode marchait à tous les coups.

Notion d’algorithmique

vous n'aviez pas le niveau en mathématiques

pour comprendre pourquoi la succession d‘étapes qu'on vous donnait était valide, mais vous étiez capable d‘exécuter chaque étape de la méthode.

- Le mot algorithme prend étymologiquement ses racines dans le nom d'un mathématicien arabe du moyen âge : Al-Kawarizmi.

Notion d’algorithmique

- en concevant un algorithme, vous pouvez décomposer un calcul complique en une succession d‘étapes compréhensibles.
- un ordinateur fonctionne de la m^eme facon qu'un monteur de bibliotheque.
- Pour chaque chose que vous lui demanderez, il faudra lui dire comment faire.

Notion d’algorithmique

- Vous aller donc lui donner des successions d'instructions a suivre, et lui les respectera a la lettre et sans jamais se tromper.
- Une suite d'instructions de la sorte est fournie a l'ordinateur sous la forme de programme.
- Pour coder un programme, on utilise un langage de programmation, par exemple C, Java, Pascal, VB...

Notion d’algorithmique

- Nous nous intéresserons uniquement à la façon de combiner des instructions pour former des programmes, indépendamment des langages de programmation.
- Le but de ce cours est donc de vous apprendre a créer des algorithmes, c'est-a-dire a décomposer des calculs compliques en successions d‘étapes simples.

Les instructions

- Une instruction est un "ordre" que l'on donne à l'ordinateur, comme on pourrait taper "format C:" dans sa console.
- Quand on donne une instruction à l'ordinateur, il faut également lui dire où est la fin de cette instruction.
- Utiliser un **point-virgule** (**;**) à la fin de chaque instruction.

Les variables et les constantes

Un programme utilise toutes sortes de valeurs différentes.

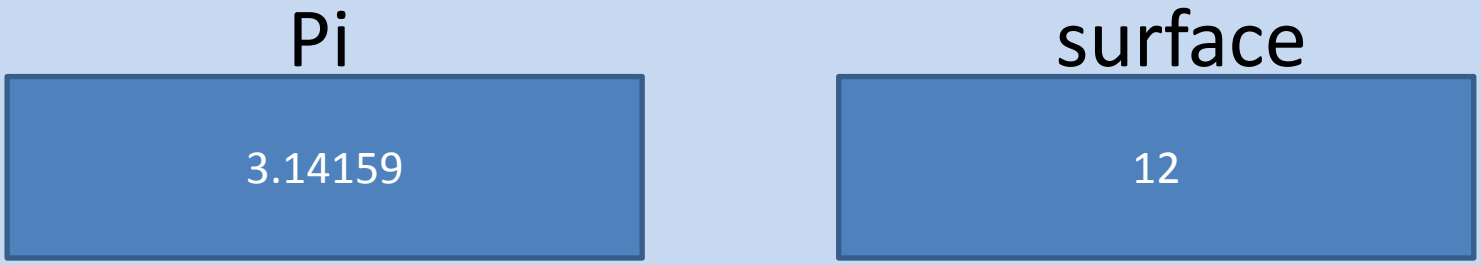
Ces valeurs sont soit variables, soit constantes.

Les variables et les constantes d'un programme désignent en réalité des espaces de stockages de données.

Chaque espace de stockage doit être identifié par un nom distinct.

Les variables et les constantes

Exemple:



Pi: est une valeur constante qu'on utilise dans les calcul des périmètres et de la surface d'un cercle. Cet espace gardera la même valeur tout au long du programme.

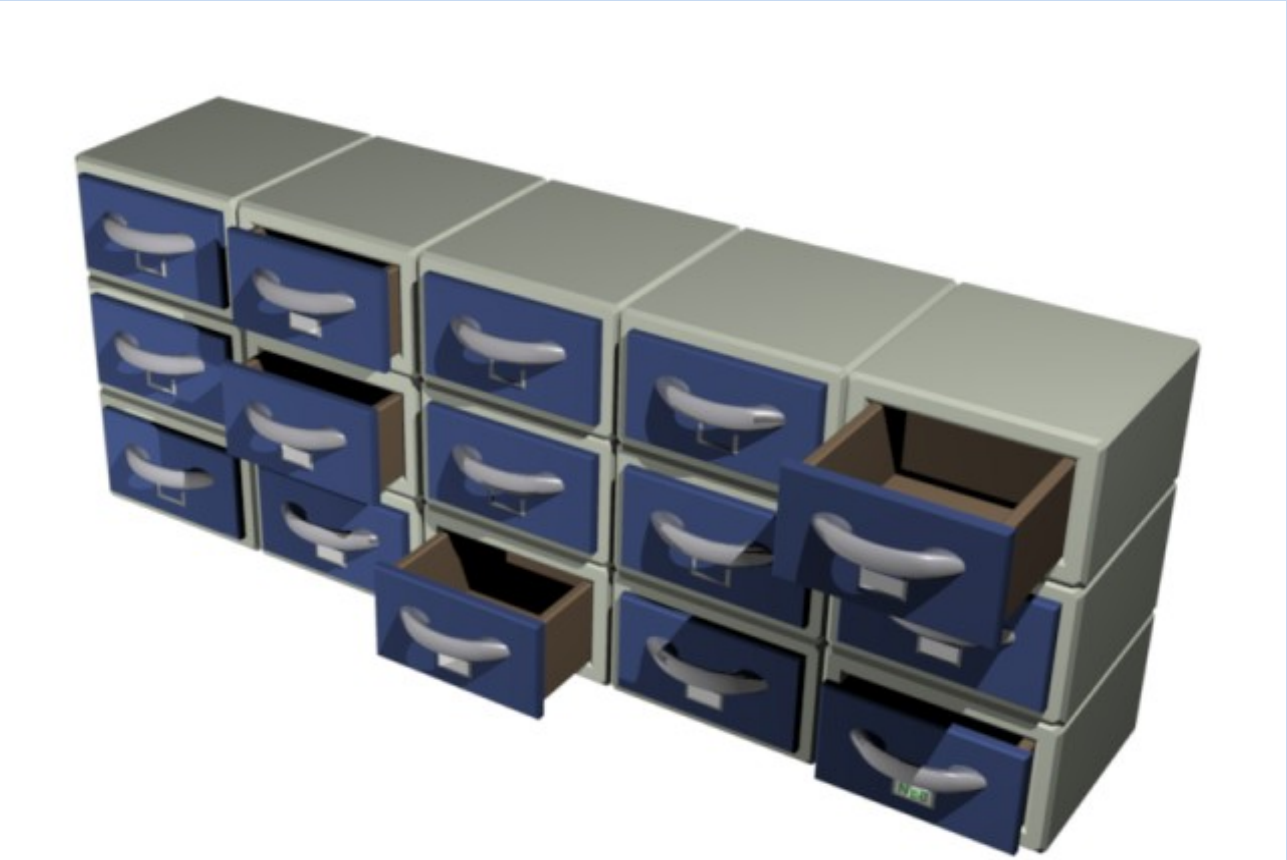
Par contre la surface d'un cercle varie selon la valeur du rayon. Cette valeur peut être stockée dans un espace identifié dans le programme par « surface ». Cet espace recevra tout au long du programme les valeurs des surfaces des différents cercles.

Les variables et les constantes

Dans la mémoire interne de l'ordinateur, les espaces de stockage de « Pi » et « Surface » sont physiquement identiques.

C'est lors de l'écriture du programme qu'on spécifiera au compilateur que l'un va recevoir une valeur constante, et l'autre une valeur variable.

Les variables et les constantes



Les types

Toutes les variables d'un programme sont d'un type donné.

Le type d'une variable est l'ensemble des valeurs que cette variable peut prendre.

Les types sont de deux sortes:

- Les types standards: ils sont définis dans le compilateur. On peut les utiliser directement.
- Les types non standard: ils ne sont pas définis et c'est à l'utilisateur de les définir.

Les types

Il existe quatre types standards pour les variables.

entier: pour les nombres entiers.

- Le type Sur 16 bits, a valeur dans - 32 768 … + 32 767 (2 E15 : : : + 2 E15 -1).
- Sur 32 bits, a valeur dans -2 147 483 648 …+2 147 483 647 (2 E31 : : : +2 E31 1).

Les types

Le type réel: pour les nombres réels.

Exemples de real

- 0.0 ;
- -21.4E3 (= -21, 4 103 = -21400) ;
- 1.234E-2 (= 1, 234 10 -2)

Les types

Le type booléen: pour les variables booléennes, également appelées variables logiques.

Ces variables peuvent prendre soit la valeur prédéfinie « True » soit la valeur prédéfinie « false »

Sur les booléens, on peut effectuer les opérations : AND, OR, XOR et NOT.

Les types

Le type caractère: pour les caractères.

Un caractère du clavier (chaque touche) correspond un chiffre appelé code ASCII.

Exemple:

Le code ASCII de A est 65 et celui de B est 6, etc..

L’affectation

Les instructions les plus élémentaires d’un programme consistent à effectuer des calculs numériques, ou d’une manière générale, à évaluer des expressions contenant des variables et des constantes, des opérations (addition, soustraction, etc…) et des fonctions.

Les résultats des évaluations de ces expressions sont la plupart du temps assignés à des variables.

L’affectation

On dit qu’il s’agit d’une affectation du résultat d’une expression à une variable.

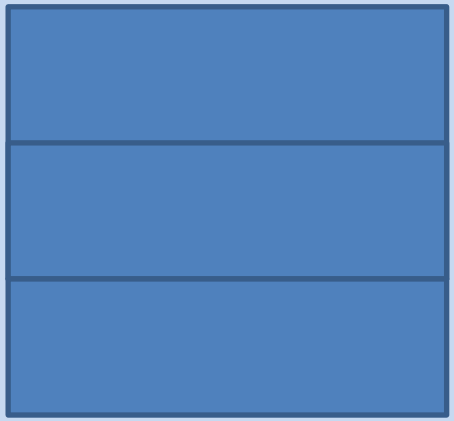
Le symbole d’affectation diffère d’un langage à un autre.

En algorithmique c’est : ←

Exemple:

X ← 15 y ← 3

z ← 2*(x+y)



Les commentaires

Un commentaire est un texte écrit entre deux slashs. // je suis un commentaire

- Un commentaire n'est pas une instruction.
- Il est par conséquent ignoré par l'ordinateur.
- Il est utile au programmeur.
- Il l'aide à se souvenir ou à expliquer aux autres programmeurs le fonctionnement du programme.
- N'apparaissent pas à l'exécution du programme.

Exercice

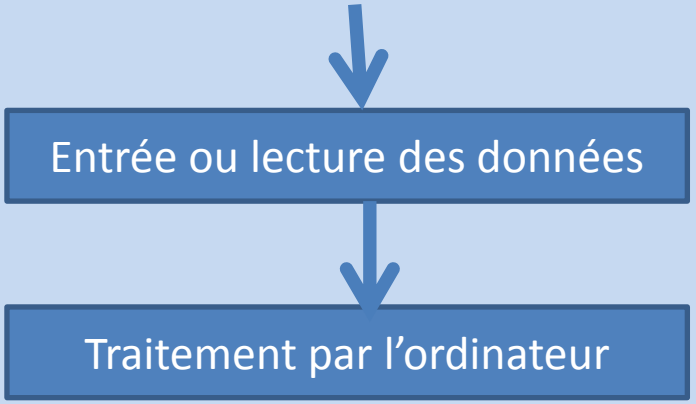
```
Programme programme01

//déclaration des constantes
Constante    x=3;
             y=5;
//déclaration des variables
Variable     z : entier;
             t : réel;
Début
        //début du programme ou corps du programme
        z := 2 * x + 1;
        t := y + 0.5
Fin.
```

Les entrées sorties

Lorsqu'on écrit un programme, c'est pour traiter les données.

Ces données sont généralement fournies par l'utilisateur du programme.

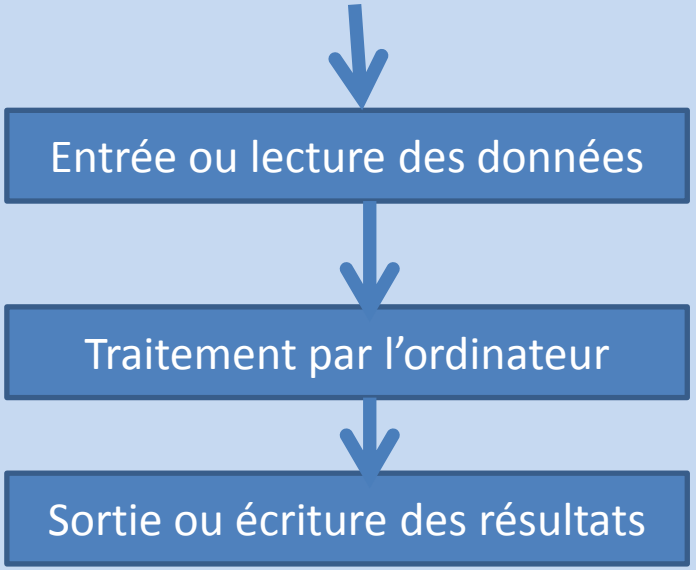
Cette opération s'appelle : entrée des données ou lecture des données.



Les entrées sorties

Après traitement automatique (par l'ordinateur) de ces données, celui-ci doit généralement nous afficher des résultats.

Cette opération s'appelle: sortie des résultats ou écriture des résultats.



Les entrées sorties

Les procédures lire et écrire:

La procédure de lecture se fait par : lire

Exemple: lire (x); lire (x, y, z);

Ici signifie que l'ordinateur attend une ou plusieurs données.

La procédure d'écriture se fait par : écrire

Exemple: écrire (x); écrire (x, y, z);

Ici, signifie que l'ordinateur va afficher le contenu de ou des variables.

Exercice

```
Programme surface
   Constante pi =3.14
Variable      rayon, surface : réel;
Debut
   ecrire('entrez la valeur du rayon');
   lire (rayon);
   surface := pi* (rayon*rayon)
   ecrire (surface)
Fin.
```

Structures de contrôle

Conditions: l'alternative si…alors…sinon est utilisée lorsqu'on a le choix d'exécuter tel bloc d'instruction si une condition donnée est vraie, et un autre bloc, si la condition est fausse.

Le bloc peut être composé d'une ou plusieurs instructions.

Structures de contrôles

Cette alternative s'écrit comme suit:

Si condition alors

{bloc 1}

Sinon

{bloc 2}

Si le bloc est constitué de plus d'une instruction, il doit alors obligatoirement commencer par « « début » et se terminer par « Fin ».

Exercice 01

```
Programme alternative1
Variable x, y : réel;
Debut
        ecrire(‘ entrez 2 nombres x et y différents’);
        lire (x,y);
        si x > y alors
            ecrire(‘x est supérieur à y’);
        sinon
            ecrire(‘y est supérieur à x’);
Fin.
        //expliquez le fonctionnement de cet algorithme.
```

Exercice 02

Ecrire un algorithme qui permet de chercher le plus grand de 3nombres entiers saisis au clavier.

```
Programme alternative2
Variable x, y, z, pg :entier;

Debut
        ecrire('entrez trois entiers');
        lire(x,y,z)

        si (x>=y) et (x>=z) alors
                pg:=x
        sinon
                si (y>=x) et (y<=z) alors
                        pg:=y
        sinon
                pg:=z;

        ecrire('Le plus grand des 3 nombres est : ', pg)
Fin.
```

Cas particulier si…alors

Dans le cas où on n'a pas deux ou plusieurs blocs d'instructions à exécuter sous certaines conditions, mais juste un seul. Ce bloc d'instructions sera exécuté si la condition est vraie. Sinon ne rien faire.

Dans ce cas, l'alternative simple « si…alors » suffit.

Exemple:

écrire un algorithme qui permet d'afficher la valeur absolue d'un nombre réel saisi au clavier.

Cas particulier « si…alors »

```
Programme alternative3
Variable x : réel;
Debut
        ecrire('entrez un nombre : ');
        lire(x)
        si x < 0 alors
                x := -x;
        ecrire ('La valeur absolue est : ', x)
Fin.
```

L’alternative « choix…de »

Dans cette structure, on peut comparer un objet (variable ou expression) à toute une série de valeurs, et d’exécuter, en fonction de la valeur effective de l’objet, différentes séquences d’instructions.

Une autre par défaut peut être prévue dans le cas ou où l’objet n’est égal à aucune des valeurs énumérées.

Syntaxe:

```
Choix <expression> de
    cas1: instruction1;

    casN: instructionN

    sinon
        instruction(N+1)
Fin.
```

Exercice

```
Programme alternative4;
Variable nb1, nb2, nb3, choix : entier;
Debut
        ecrire('entrez trois nombres');
        lire(nb1,nb2,nb3);
        //affichage du menu et saisi du choix
        ecrire('1. pour la multiplication')
        ecrire('2. pour la somme');
        ecrire('3. pour la moyenne');
        ecrire('4. votre choix');
        lire(choix) ;
    Choix choix de
        1: ecrire ('le produit des trois nombres est : ', nb1*nb2*nb3);
        2: ecrire ('la somme des trois nombres est : ', nb1+nb2+nb3);
        3: ecrire ('la moyenne des trois nombre est :', nb1+nb2+nb3 / 3)
        sinon
            ecrire ('saisie de choix incorrecte ')
    fin;
Fin.
```

// expliquez le fonctionnement de cet algorithme.

Les boucles

Lorsqu'un traitement est répété plusieurs fois successivement, il faut utiliser une boucle.

Il existe trois façon d'écrire un traitement répétitif:

- En utilisant une boucle « tant que… faire »
- En utilisant une boucle « répéter…jusqu'à »
- En utilisant une boucle « pour…faire »

La boucle « pour…faire »

La syntaxe de la boucle se présente comme suit:

Pour « variable := valeur initiale » à « valeur finale » faire

debut

{bloc d'instructions}

fin;

Valeur initiale: désigne début de la boucle.

Valeur finale: désigne la valeur d'arrêt de la boucle.

Variable: ne peut pas être de type réel.

Exercice

```
Programme boucle1;
Variable i, chiffre, somme : entier;
Debut
        somme := 0;
        pour i := 1 à 10 faire
                debut
                        ecrire('donnez un chiffre');
                        lire (chiffre);
                        somme := somme+chiffre
                fin;
        ecrire('la somme des 10 chiffres saisis est :', somme)
Fin.
```

Exercice

Imprimer les chiffres de 9 à 0.

```
Programme boucle 2;
Variable i : entier;
Debut
    pour i := 9 jusqu'à 0 faire
        écrire (i);
Fin.
```

La boucle « tantque…faire »

Syntaxe:

Tantque « condition » faire

début

{bloc d'instructions}

fin;

La condition peut être une expression booléenne ou une variable booléenne.

« tant que la condition est vraie, le bloc d'instructions entre début et fin est répété. »

Exercice

Calculer et afficher les 10 chiffres saisis au clavier.

```
Programme boucle3;
Variable : i, chiffre, somme : entier;
Debut
        i := 1; somme := 0;
        tantque  i < 10 faire
                debut
                        ecrire (‘donnez un chiffre’);
                        lire (chiffre);
                        somme := somme + chiffre;
                        i := i + 1;
                fin;
        ecrire (‘la somme des 10 chiffres saisis est :’, somme)
Fin.
```

La boucle « répéter…jusqu'à »

La syntaxe:

Répéter

{bloc d'instructions}

Jusqu'à <condition>

Le bloc d'instruction est répété jusqu'à ce que la condition soit vérifiée.

Si la condition est vérifiée dés le départ, le bloc d'instructions est tout de même exécuté une fois.

Exercice

```
Programme boucle 4
Variable i, chiffre, somme : entier;
Debut
        i := 1; somme := 0;
        répéter
                écrire (‘donnez un chiffre’);
                lire (chiffre);
                somme := somme +chiffre;
                i := i+1;
        jusqu’à i > 10
        écrire (‘la somme des 10 chiffres saisis est :’, somme)
Fin.
```

Les procédures et fonctions

Une procédure est un sous programme destiné à une tâche particulière. Elle permet d'éviter les répétitions de parties de programmes identiques, rendant ainsi les programmes plus lisibles et plus faciles à maintenir.

Une procédure est déclarée et définie dans la partie déclaration du programme selon la syntaxe suivante:

Procédure <nom de la procédure>;
debut
{instructions de la procédure}
fin;

{Une fois déclarée et définie, une procédure peut être utilisée (appelée) en tout point du programme}

Exercice 01

```
Programme procedure1;
{déclaration et définition de la procédure}
Procédure affiche;
        debut
            ecrire (‘ je suis dans la procédure’);
        fin;
{programme principal}
Debut
        ecrire (‘je suis dans le programme principal’);
        affiche; {appel de la procédure affiche}
        ecrire (‘je suis de retour dans le programme principal’)
Fin.
```

Procédure avec paramètres

Une procédure peut utiliser des paramètres (ou arguments). La déclaration et définition d'une telle procédure se fait comme suite:

Procédure <nom procédure> (p1:type1; p2:type2; …);

debut

{instructions de la procédure}

fin;

Où p1, p2, …, désignent les paramètres de la procédure, et type1, type2,…, leurs types (entiers, réels,…)

Procédure avec paramètres

Il existe deux modes de passage de paramètres:

- Passage par valeur.
- Passage par adresse (ou par variables).

Procédure avec paramètres

Passage de paramètres par valeur:

Dans le passage de paramètres par valeur, les changements que subissent ces paramètres dans la procédure n'apparaissent pas dans le programme principal et dans les autres procédures.

Exercice 01

```
Programme procedure2;
Procedure permuter (a,b : entier);
Variable t : entier;
Debut
        ecrire (‘a =‘, a,   ‘b = ‘,   b);
        t := a ;
        a := b ;
        b := t ;
        ecrire (‘a = ‘, a, ‘b = ‘, b)
Fin;
{programme principal}
Variable x, y : entier;
Debut
        ecrire (‘ donnez x et y’);
        lire (x,y);
        permuter (x,y); {appel de la procédure permuter}
        ecrire ( ‘x =‘, x,      ‘ y =‘,   y)
Fin.

//résultat : à l’entrée de la procédure: a= 7 b=9
// après la permutation dans la procédure : a=9 b=7
// au retour dans le programme principal : x=7 y = 9
```

Passage de paramètres par variable

Dans le passage de paramètres par variable, les changements que subissent ces paramètres dans la procédure apparaissent dans le programme principal et dans les autres procédures. Le passage de paramètres par variables se fait avec l'ajout du mot clé variable dans l'entête de la procédure.

Exercice

```
Programme procedure3;
Procedure permuter (var a, b : entier);
Variable t : entier;
Debut
        ecrire (‘a =‘, a,   ‘b = ‘,   b);
        t := a ;
        a := b ;
        b := t ;
        ecrire (‘a = ‘, a, ‘b = ‘, b)
Fin;
{programme principal}
Variable x, y : entier;
Debut
        ecrire (‘ donnez x et y’);
        lire (x,y);
        permuter (x,y); {appel de la procédure permuter}
        ecrire ( ‘x =‘, x,      ‘ y =‘,   y)
Fin.

//résultat : à l’entrée de la procédure: a= 7 b=9
// après la permutation dans la procédure : a=9 b=7
// au retour dans le programme principal : x=9 y = 7
```

Variables locales et variables globales

Une variable est locale à la procédure dans laquelle elle est déclarée. Elle n'existe que dans cette procédure. Cas de la variable t de l'exemple précédent.

Une variable est dite globale si elle peut être utilisée dans le programme principal ainsi que dans toutes les procédures définies.

Pour être globale aussi, une variable doit être déclarée avant les procédures

Exercice

```
Programme procedure4;
Variable  pp : entier; {c'est une variable globale }
Procedure plus_petit (a, b, c : entier);
    debut
        si (a<=b) et (a<=c) alors
                pp:=c
    fin;
Var x, y, z : entier; {variables locales}
Debut
        ecrire ('donnez 3 entiers :');
        lire(x,y,z);
        plus_petit(x,y,z); {appel de la procédure}
        ecrire('le plus petit des 3 entiers est :', pp);
Fin.
//ici on récupère le résultat dans la variable pp.
```

Notion de fonction

Une fonction est une procédure qui renvoie toujours une valeur.

Elle possède par conséquent un type.

Le type de la fonction est le type de la valeur qu'elle renvoie en sortie.

Elle fonctionne exactement comme une procédure.

Syntaxe:

Fonction <nom fonction> (paramètres): <type>;

debut

{instructions de la fonction}

fin;

Exercice

Ecrire un programme qui contient une fonction somme qui reçoit en paramètres 3 entiers, puis renvoie leur somme. La somme de trois entiers étant un entier, la fonction somme est donc de type « entier ».

Solution

```
Programme fonction;
Fonction somme (a,b,c : entier): entier;
        debut
                somme := a + b + c
        fin;
Var x, y, z, som : entier;
Debut
        ecrire(‘ donnez 3 entiers’);
        lire (x,y,z);
        som := somme(x,y,z);
        écrire(‘ la somme des 3 nombres entiers est :’, som)
Fin.
```