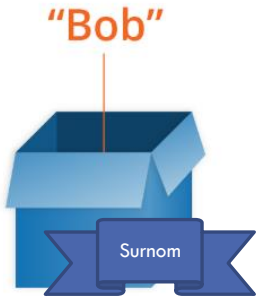
LES INSTRUCTIONS DE BASES : LES VARIABLES, AFFICHER, SAISIR

LES VARIABLES

Base de la programmation, les variables sont des conteneurs permettant de stocker une valeur, celle-ci pouvant évoluer au cours du programme (d'où le nom de variable). Dans l'ordinateur ces variables sont stockées en mémoire vive (RAM)



Dit autrement : Une variable est l'association d'une étiquette (**nom**) et d'une boîte pouvant contenir une **valeur**.

Le type d'une variable sert à préciser la nature des valeurs acceptables par le conteneur. C'est ce que l'on peut mettre dans la "boite".

Les types de bases en algorithmique sont : **Entier, Réel (ou famille des nombres), caractère, chaîne de caractère, booléen.**

En résumé : **variable = NOM + TYPE + VALEUR**

Dans notre exemple **Nom**= « Surnom », **Type** = « chaine de caractère », **Valeur** = « Bob ».

D'autres types de données plus complexe seront utilisés : **objet** personnalisée (par exemple un type Personne comprenant son nom, prénom...) ,une date, ou bien encore un **tableau**.

CHOIX DU NOM

Je veux mémoriser l'âge d'une personne dans une variable, j'ai le choix de l'appeler :

- a ➔ ici c'est trop court, pas assez précis
- âge ➔OK en algo, mais on évitera tout caractère accentué car rarement accepté dans un langage de programmation, notamment pour des problèmes d'encodage des caractères.
- **age ➔ OK en algo, OK dans un langage**
- ageDeLaPersonneDontJeSuisEntrainDeParler ➔ très (trop) précis, long à taper (risque d'erreur de saisie par la suite)

Le nom d'une variable est **unique** dans un même programme

CHOIX DU TYPE ET DE LA VALEUR

Les 2 vont de pairs. Une fois le type de notre variable choisie, il faut affecter une valeur "possible" et inversement. Voir le tableau ci-dessous :

Type	Utilité	Exemple
Booléen	Représente un état binaire, vrai ou faux	Vrai,True ou Faux, False
Entier	Représente un nombre entier	12345
Réel	Représente un nombre réel	3.14
Nombre	Représente un nombre entier ou réel	
Caractère	Représente un caractère unique	"c" ou 'c'
Chaîne de caractères	Représente un texte	"contenu de la chaîne"

SYNTAXE ALGORITHMIQUE

On utilise le mot clef **var**

```
var nom_de_la_variable ,autre_variable : type
```

Exemple :

```
Activité exemple
var age : Nombre
Début
        age <- 12
        Afficher "J'ai", age, "ans"
Fin
```

LES CONSTANTES

Les constantes sont des **variables (nom + type +valeur)** dont la valeur ne peut être changé tout au long de l'algorithme.

Syntaxe : On note ici que la valeur est fixée une bonne fois pour toute lors de la déclaration. On utilise le mot clef **const**

```
const nom_de_la_variable : type = valeur
```

LES OPERATIONS

Selon le type des variables, les opérations possibles seront différentes.

Opération	Symboles	S'applique sur
Opérateur arithmétique	+, -, *, /, ^	Entier et Réel
Division d'entier et Modulo	DIV, MOD	Entier
Comparaisons	<, >, <=, >=, =, <>	Entier, Réel, Booléen, caractère, chaine de caractère
Opérateur Logique	ET,OU, NON	Booléen
Concaténation	+ ou &	Caractère et Chaine de caractère

L'AFFECTATION

Il n'y a que deux utilisations possible d'une variable :

- On peut la lire (c.a.d récupérer la valeur qu'elle contient); par exemple pour l'afficher...
- On peut écrire dedans (lui donner/modifier une/sa valeur). On parle alors d'affectation ou de saisie.
  - Le symbole utilisé pour faire une affectation est : ← .
  - La direction de la flèche donne le sens de l'affectation : de la droite vers la gauche. La variable se trouve donc à gauche de la flèche.
  - C'est l'affectation qui permet de faire varier les valeurs de notre variable.

Il s'utilise de la manière suivante :

```
Variable ← valeur
Variable ← expression
```

Dans le cas d'une affectation avec expression (par ex: a ← b + 10) les choses se font suivants les 2 étapes:

1. évaluation de l'expression (b+10 est d'abord calculé)
2. le résultat est affecté à la variable (ici a).

Attention	
	• **L'affectation est une instruction qui est dite "destructrice".** • **L'ancienne valeur de la variable est détruite, écrasée, effacée par la nouvelle valeur !** • **A gauche du symbole d'affectation ne doit figurer uniquement une variable** • **La variable et la valeur ou expression évaluée doivent être de type compatible.** • **Ne pas confondre l'affectation avec l'égalité mathématique. On évite donc d'utiliser le symbole =**

COMPARATIF ALGO ET MATH

Contrairement aux mathématiques où x = y est équivalent à y = x. L'instruction x ← y signifie que x va prendre la valeur de y. Au contraire y ← x signifie que y va prendre la valeur de x.

En algorithmique le symbole d'affectation c'est le signe ← Mais en pratique, la quasi-totalité des langages de programmation emploient le signe égal (=). La confusion avec les maths est facile. En maths, A = B et B = A sont deux propositions strictement équivalentes. En informatique, absolument pas, puisque cela revient à écrire A ← B et B ← A

- Ex1: a + 5 ← b n'a pas de sens en algo, alors que a + 5 = b a une signification en mathématiques
- Ex2: a ← a + 3 a un sens en programmation, écrire a=a+3 n'a pas de sens en mathématiques (équation sans solution)
- Ex3: y = 3 x + 2 En mathématique les « variables » x et y satisfaisant à l'équation existent en nombre infini (graphiquement, l'ensemble des solutions à cette équation dessine une droite). En informatique, une variable possède à un moment donné une valeur et une seule. A la rigueur, elle peut ne pas avoir de valeur du tout (une fois qu'elle a été déclarée, et tant qu'on ne l'a pas affectée. Mais ce qui est important, c'est que cette valeur justement, ne « varie » pas (Du moins pas tant qu'elle n'a pas fait l'objet d'une nouvelle affectation).

AUTRE EXEMPLE:

```
maVar ← 24     // Attribue la valeur 24 à la variable maVar.
               // Erreur si maVar n'est pas du type Entier ou Réel

res ← maVar    // res et maVar même valeur.
               // maVar garde la même valeur. Il n'y a pas eu de transfert de valeur mais une affectation!
               // Erreur possible si res n'est pas du même type que maVar


res ← res + 1  // calcul de (res + 1), affectation du résultat à res : incrémentation
               // ou opération de cumul

S ← "coucou"           // ici S est une chaine de caractère
C ← 'a'                // ici a est un caractère
S ← S + C              // on concatène la chaine S avec le caractère c
S ← coucou             // Attention, l'absence de guillemet signifie que coucou est une variable

A ← Vrai       // A est un booléen
B ← Faux       // B est un booléen
C ← A OU B     // C est un booléen. Sa valeur est le résultat de l'opération (A OU B)
```

LA SAISIE D'INFORMATIONS

L'opération **SAISIR** permet à l'utilisateur du programme d'interagir avec ce dernier. La saisie peut être au clavier, tactile ou tout autres moyens. C'est une opération bloquante. Tant que l'utilisateur n'entre rien au clavier, le déroulement du programme est stoppé.

Syntaxe :

```
Saisir (<liste de noms de variables>)
```

Exemples :

```
Saisir (A)             // ici A est un Entier
Saisir (nom, prenom)   // ici nom et prenom sont des Chaine
```

On pourrait penser que l'instruction de saisie est inutile car on dispose déjà d'un moyen d'attribuer une valeur aux variables, par l'instruction **d'affectation**. En fait, l'instruction de saisie est indispensable pour permettre d'utiliser le même programme avec des données différentes sans avoir à changer les valeurs du programme à chaque fois.

Par exemple, l'instruction Saisir(x) à laquelle on fournirait au clavier la valeur 5, pourrait être remplacée par x ← 5 (**statique**). Mais alors si on veut utiliser le programme avec une autre valeur, il faudra modifier le programme. En revanche, si on utilise une instruction de saisie, le choix de la valeur se fait en cours d'exécution (**dynamique**) du programme. On peut donc utiliser le programme autant de fois que l'on veut avec des données différentes sans avoir à le modifier.

L'AFFICHAGE D'INFORMATIONS

L'instruction **AFFICHER** permet de fournir des résultats sous forme directement compréhensible pour l'utilisateur à travers l'écran, ou tout autre périphérique de sortie.

Syntaxe:

```
Afficher (<liste de noms de variables, de constantes ou d'expressions>)
```

Exemples:

```
Afficher(A)    // affiche la valeur de la variable A
Afficher(A,B)  // affiche la valeur de la variable A, puis celle de B

Afficher (unNombre, "est différent de 0") //affiche la valeur de unNombre puis la chaine "est différent.."

Afficher ("La somme de", unNombre, "et", val , "est", unNombre+ val)
        // ici on mélange l'affichage de valeurs littérales "La somme de"
        // et de variables, et d'expression arithmétique.
```

EXERCICE

Exercice 1: Quelles seront les valeurs des variables a et b après exécution des instructions suivantes ?

```
ACTIVITE Exo1
VAR a, b : Entier
DEBUT
        a ← 2
        b ← a + 3
        a ← 3
FIN
```

Exercice 2: Quelles seront les valeurs des variables a, b et c après exécution des instructions suivantes ?

```
ACTIVITE Exo2
VAR a, b, c : Entier
DEBUT
        a ← 5
        b ← 3
        c ← a + b
        a ← 7
        c ← b – a
FIN
```

Exercice 3: Quelles seront les valeurs des variables a et b après exécution des instructions suivantes ?

```
ACTIVITE Exo3
VAR a,b : Entier
DEBUT
        a ← 6
        b ← a + 4
        a ← a + 1
        b ← a – 4
FIN
```

Exercice 4: Quelles seront les valeurs des variables a, b et c après exécution des instructions suivantes ?

```
ACTIVITE Exo4
VAR a, b, c : Entier
DEBUT
        a ← 3
        b ← 10
        c ← a + b
        b ← a + b
        a ← c
FIN
```

Exercice5: Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?

```
ACTIVITE Exo5
VAR a,b : Entier
DEBUT
        A ← 5
        B ← 17
        A ← B
        B ← A
FIN
```

Exercice 6 : Permutation. Écrire un algorithme permettant d'échanger les valeurs de deux variables A et B, et ce quel que soit leur contenu préalable.

Exercice 7: Permutation circulaire. On dispose de trois variables a, b et c. Ecrire un algorithme transférant à B la valeur de A, à C la valeur de B et à A la valeur de C.

Exercice 8: Quelles sont les valeurs de a, b, c à la fin de l'algorithme suivant ?

```
ACTIVITE Exo8
VAR a, b, c : Chaînes de caractères
DEBUT
        a ← "423"
        b ← "12"
        c ← a + b
FIN
```

Exercice 9 : on souhaite écrire un programme permettant d'élever au cube un nombre.

Exercice 10:

- Ecrire un algorithme permettant de calculer la surface d'un rectangle.
- Ecrire un algorithme permettant de calculer le périmètre d'un cercle.
- Ecrire un algorithme permettant de calculer la somme, le produit et la moyenne de 3 nombres.

Exercice 11: Ecrire un algorithme qui calcule l'âge de l'utilisateur en fonction de son année de naissance.