Les fondements de la programmation

Slide 2

Les variables

Qu'est-ce qu'une variable ?

Une **variable** est un triplet composé :

- d'un type,
- d'un nom,
- d'une valeur selon le type.

Qu'est-ce qu'une variable ?

Règle :

1° - Déclaration

2° - Initialisation

3° - Utilisation

1° - Déclaration

On déclare une variable en :

- lui donnant un **nom**
- et en précisant son **type**.

```
maVariable EST_DU_TYPE ENTIER
```

maVariable c’est le nom et ENTIER c’est le type

2° - Initialisation

Un fois déclarée une variable doit être initialisée.
On lui affecte une valeur.

```
maVariable PREND_LA_VALEUR 12
```

Maintenant maVariable a pour valeur 12.
L'affectation se fait en fonction de son type, 12 c'est bien un entier.

3° - Utilisation

Un fois déclarée et initialisée, une variable peut être utilisée.

On l’utilise en lisant sa valeur, pour un calcul ou un affichage.

```
uneAutreVariable PREND_LA_VALEUR maVariable * 2
```

```
AFFICHER maVariable
```

Les Types de variables

Les types de variables

On distingue 4 types principaux de variables en programmation :

- Une variable entière,
- Une variable décimale,
- Une variable boolean,
- Une variable chaîne de caractères.

Les variables entières

```
maVariable EST_DU_TYPE ENTIER
```

maVariable est un entier.

```
maVariable PREND_LA_VALEUR 12
```

maVariable est initialisée par la valeur entière 12

Les variables décimale

```
maVariable EST_DU_TYPE DECIMALE
```

maVariable est un décimale.

```
maVariable PREND_LA_VALEUR 12.42
```

maVariable est initialisée par la valeur décimale 12,42

Les variables boolean

```
maVariable EST_DU_TYPE BOOLEAN
```

maVariable est un boolean.

```
maVariable PREND_LA_VALEUR true
```

maVariable est initialisée par la valeur décimale **true** (**false** est l'autre valeur possible)

Les variables CHAINE

```
maVariable EST_DU_TYPE CHAINE_DE_CARATERES
```

maVariable est une chaîne de caractères.

```
maVariable PREND_LA_VALEUR “Je suis une chaîne de caractères”
```

maVariable est initialisée avec la valeur entre double quotes.

En java

Déclaration et initialisation :

```
// Un entier, type int
int varInt = 12;

// Un décimal, type double
double varDouble = 13.56792;

// Un boolean, type boolean
boolean varBoolean = true;

// Une chaîne de caractère, type String
String varString = “Une chaîne”;
```

Premières expressions

L’instruction AFFICHER

```
AFFICHER maVariable
```

Permet d’afficher la valeur de la variable.

Attention à l’initialiser avant de l’afficher.

```
AFFICHER “Mon résultat est : “
AFFICHER resultat
```

L’instruction LIRE

```
LIRE maVariable
```

Permet d’initialiser une variable par la saisie de l’utilisateur.

De cette manière la variable sera initialisée à l’exécution du programme et pas dans le programme.

En java

Affichage et saisie :

```
// Un entier, type int
int varInt = 12;

// Affichage simple de la valeur
System.out.println(varInt);

// Saisie d'une nouvelle valeur pour varInt
varInt = scanner.nexInt();

// Affichage avec une concaténation (le plus entre une String et un int)
System.out.println("La nouvelle valeur est : " + varInt);
```

Opérateurs de calcul

Opérateurs de calcul

Nom	Type	Taille
Négation	-a	Opposé de a
Addition	a + b	Somme de a et b
Soustraction	a - b	Différence de a et b
Multiplication	a * b	Produit de a et b
Division	a / b	Division de a par b
Modulo	a % b	Reste de a divisé par b

Opérateurs de calcul

Initialisation et calcul :

```
maVariable PREND_LA_VALEUR 12
maDeuxièmeVariable PREND_LA_VALEUR -maVariable

AFFICHER maDeuxièmeVariable
```

Opérateurs de calcul

Initialisation et calcul :

```
maVariable PREND_LA_VALEUR 12
maDeuxièmeVariable PREND_LA_VALEUR -maVariable

AFFICHER maDeuxièmeVariable
```

-12

Opérateurs de calcul

Initialisation et calcul :

```
maVariable PREND_LA_VALEUR 12
maDeuxièmeVariable PREND_LA_VALEUR 5

AFFICHERCALCUL maVariable + maDeuxièmeVariable
```

Opérateurs de calcul

Initialisation et calcul :

```
maVariable PREND_LA_VALEUR 12
maDeuxièmeVariable PREND_LA_VALEUR 5

AFFICHERCALCUL maVariable + maDeuxièmeVariable
```

17

Opérateurs de calcul

Initialisation et calcul :

```
maVariable PREND_LA_VALEUR 12
maDeuxièmeVariable PREND_LA_VALEUR 5

AFFICHERCALCUL maVariable / maDeuxièmeVariable
```

Opérateurs de calcul

Initialisation et calcul :

```
maVariable PREND_LA_VALEUR 12
maDeuxièmeVariable PREND_LA_VALEUR 5

AFFICHERCALCUL maVariable / maDeuxièmeVariable
```

2.4

Opérateurs de calcul

Initialisation et calcul :

```
maVariable PREND_LA_VALEUR 12
maVariable PREND_LA_VALEUR maVariable + 1

AFFICHER maVariable
```

Opérateurs de calcul

Initialisation et calcul :

```
maVariable PREND_LA_VALEUR 12
maVariable PREND_LA_VALEUR maVariable + 1

AFFICHER maVariable
```

13

Opérateurs logiques

Opérateurs logiques

Ici a et b sont des variables de type boolean.

Logique	Opérateur	Résultat
non	!a	vrai si a est égal à faux
et	a && b	vrai si a **et** b sont égaux vrai
ou	a || b	vrai si a **ou** b est égal vrai

Algèbre de Boole

Conjonction
Elle est définie de la manière suivante :
a ET b est VRAI si et seulement si a est VRAI et b est VRAI.

Table de la loi **ET**

b\a	0	1
0	0	0
1	0	1

b\a	false	true
false	false	false
true	false	true

Algèbre de Boole

Disjonction
Elle est définie de la manière suivante :
a OU b est VRAI si et seulement si a est VRAI ou b est VRAI.

Table de la loi **OU**

b\a	0	1
0	0	1
1	1	1

b\a	false	true
false	false	true
true	true	true

Comparaisons

Comparaisons

Le résultat lui est boolean : Vrai ou Faux

Comparaison	Opérateur	Résultat
égal	a == b	vrai si a est égal à b
différent	a != b	vrai si a est différent de b
plus petit	a < b	vrai si a est strictement plus petit que b
plus grand	a > b	vrai si a est strictement plus grand que b
inférieur ou égal	a <= b	vrai si a est plus petit ou égal à b
supérieur ou égal	a >= b	vrai si a est plus grand ou égal à b

Les instructions conditionnelles

Condition SI ALORS

Une instruction conditionnelle permet d'exécuter un bloc d'instructions seulement dans le cas où la condition est vraie.

Si la condition est vraie **alors** j'exécute le bloc d'instructions.

Le même programme peut alors avoir un comportement différent d'une exécution à un autre, selon l'état dans lequel il est mis.

Condition SI ALORS

La syntaxe :

```
SI (condition) ALORS
  DEBUT_SI

  FIN_SI
```

Condition SI ALORS

Exemple :

```
maVariable PREND_LA_VALEUR 12
SI (maVariable > 10) ALORS
  DEBUT_SI
  AFFICHER "Plus grand que 10 !!"
  FIN_SI
```

Résultat :

Plus grand que 10 !!

Condition SI ALORS

Exemple :

```
maVariable PREND_LA_VALEUR 7
SI (maVariable > 10) ALORS
  DEBUT_SI
  AFFICHER "Plus grand que 10 !!"
  FIN_SI
```

Résultat :

Condition SI ALORS

Exemple :

```
LIRE maVariable
SI (maVariable >= 0 && maVariable <= 100) ALORS
   DEBUT_SI
   AFFICHER "Valeur comprise entre 0 et 100."
   FIN_SI
```

Condition SI ALORS SINON

Possibilité de mettre un bloc SINON qui sera exécuté dans cas où la condition est fausse.

Si la condition est fausse **alors** j'exécute le bloc d'instructions du **SINON**.

Condition SI ALORS SINON

La syntaxe :

```
SI (condition) ALORS
  DEBUT_SI

  FIN_SI
SINON
  DEBUT_SINON

  FIN_SINON
```

Condition SI ALORS SINON

Exemple :

```
LIRE maVariable
SI (maVariable > 10) ALORS
   DEBUT_SI
   AFFICHER "Plus grand que 10 !!"
   FIN_SI
SINON
   DEBUT_SINON
   AFFICHER "Plus petit ou égal à 10 !!"
   FIN_SINON
```

En java

Une condition if :

```
int varInt = 12;

if (varInt > 10) {
   System.out.println(varInt + " est plus grand que 10" );
}
```

En java

Une condition if / else :

```
int varInt = 12;

if (varInt > 10) {
   System.out.println(varInt + " est plus grand que 10" );
} else {
   System.out.println(varInt + " est plus petit ou égal à 10" );
}
```

En java

Une condition if / else et une imbrication :

```
int varInt = 12;

if (varInt > 10) {
   System.out.println(varInt + " est plus grand que 10" );
} else {
   if (varInt < 10) {
      System.out.println(varInt + " est plus petit que 10" );
   } else {
      System.out.println(varInt + " est égal à 10" );
   }
}
```

En java

Une condition if / else if / else :

```
int varInt = 12;

if (varInt > 10) {
   System.out.println(varInt + " est plus grand que 10" );
} else if (varInt < 10) {
   System.out.println(varInt + " est plus petit que 10" );
} else {
   System.out.println(varInt + " est égal à 10" );
}
```

Les instructions de répétition

Une boucle

Une boucle permet de **répéter** un bloc d'instruction.

L'objectif est de **ne pas répéter séquentiellement** le même bout de code mais de le mettre dans une boucle de répétition.

Boucle TANT QUE

Une boucle de répétition de type **“tant que”** permet de répéter un bloc d’instruction tant que la condition est **vraie**.

On ne connaît pas à l’avance le nombre de répétitions.

Boucle TANT QUE

La syntaxe :

```
TANT_QUE (condition) FAIRE
  DEBUT_TANT_QUE

  FIN_TANT_QUE
```

On boucle tant que la condition est vraie.

Boucle TANT QUE

Exemple :

```
maVariable PREND_LA_VALEUR -1
TANT_QUE (maVariable < 0 || maVariable > 100) FAIRE
  DEBUT_TANT_QUE
  LIRE maVariable
  FIN_TANT_QUE
AFFICHER maVariable
```

On a la garantie que la saisie de maVariable est comprise entre 0 et 100.

En java

Une boucle while :

```
int varInt = scanner.nextInt();

while (varInt < 10) {
   System.out.println(varInt);
   varInt = varInt + 1;
}
```

On répète tant que varInt est toujours inférieur à 10.

Boucle POUR

Une boucle de répétition de type **“pour”** permet de répéter un bloc d’instruction un nombre fini de fois.

Le nombre de répétitions est connu, on utilise une variable de type NOMBRE qui est notre **indice de boucle**.

Boucle POUR

La syntaxe :

```
POUR indice ALLANT_DE valeur_de_début A  valeur_de_fin
  DEBUT_POUR

  FIN_POUR
```

indice est une variable de type entier déclarée dans la boucle.

Boucle POUR

La syntaxe :

```
POUR indice ALLANT_DE valeur_de_début A  valeur_de_fin
  DEBUT_POUR

  FIN_POUR
```

indice prend la **valeur_de_début** pour le premier tour.

Boucle POUR

La syntaxe :

```
POUR indice ALLANT_DE valeur_de_début A  valeur_de_fin
  DEBUT_POUR

  FIN_POUR
```

indice est **incrémenté** d'une unité à chaque fin de tour.

Boucle POUR

La syntaxe :

```
POUR indice ALLANT_DE valeur_de_début A  valeur_de_fin
  DEBUT_POUR

  FIN_POUR
```

indice prend la **valeur_de_fin** pour son dernier tour.
Et la boucle de répétition s'arrête.

Boucle POUR

Exemple :

```
POUR indice ALLANT_DE 1 A 10
  DEBUT_POUR
  AFFICHER indice
  FIN_POUR
```

Boucle POUR

Exemple :

```
POUR indice ALLANT_DE 1 A 10
  DEBUT_POUR
  AFFICHER indice
  FIN_POUR
```

1
2
3
4
5
6
7
8
9
10

En java

Une boucle for :

```
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

Pour un i commençant à 0, on répète tant que i est inférieur à 10.