JAVA Conditionals, loops & Arrays

Arrays

Un tableau est un ensemble de variables du même type. Lorsque vous devez stocker une liste de valeurs, telles que des nombres, vous pouvez les stocker dans un tableau, au lieu de déclarer des variables distinctes pour chaque nombre.

To declare an array, you need to define the type of the elements with **square brackets**.
For example, to declare an array of integers:

```
int[] arr;
```

The name of the array is **arr**. The type of elements it will hold is int.

Now, you need to define the array's capacity, or the number of elements it will hold. To accomplish this, use the keyword **new**.

```
int[] arr = new int[4];
```

The code above declares an array of 4 integers.
In an array, the elements are ordered and each has a specific and constant position, which is called an **index**.

To reference elements in an array, type the name of the array followed by the index position within a pair of square brackets.
Example:

```
arr[2] = 42;
```

This assigns a value of 42 to the element with 2 as its index.

> Note that elements in the array are identified with **zero-based** index numbers, meaning that the first element's index is 0 rather than one. So, the maximum index of the array int[4] is 4.

Initializing Arrays

Initializing Arrays

Java provides a shortcut for instantiating arrays of primitive types and strings.
If you already know what values to insert into the array, you can use an **array literal**.
Example of an array literal:

```
String[] myNames = {"A", "B", "C", "D"};
System.out.println(myNames[2]);

// Outputs "C"
```

Try It Yourself

Place the values in a **comma-separated** list, enclosed in curly braces.
The code above automatically initializes an array containing 4 elements, and stores the provided values.

> Sometimes you might see the square brackets placed after the array name, which also works, but the preferred way is to place the brackets after the array's data type.

Summing elements in Arrays

Array length

```
int[ ] intArr = new int[5];
System.out.println(intArr.length);

//Outputs 5
```

Vous pouvez accéder à la longueur d'un tableau (le nombre d'éléments qu'il stocke) via sa propriété length.

Sum of elements

```
int [ ] myArr = {6, 42, 3, 7};
int sum=0;
for(int x=0; x<myArr.length; x++) {
  sum += myArr[x];
}
System.out.println(sum);

// 58
```

Nous pouvons calculer la somme de tous les éléments d'un tableau en utilisant des boucles. La boucle for est la boucle la plus utilisée lors du travail avec des tableaux, car nous pouvons utiliser la longueur du tableau pour déterminer combien de fois exécuter la boucle.

Enhanced for Loop

```
int[ ] primes = {2, 3, 5, 7};

for (int t: primes) {
  System.out.println(t);
}

/*
2
3
5
7
*/
```

La boucle for for augmentée (parfois appelée "enhanced for" loop) est utilisée pour traverser les éléments dans les arrays. Les avantages sont qu'il élimine la possibilité de bugs et rend le code plus facile à lire.

Multidimensionnal Arrays

Multidimensional arrays are array that contain other a
most basic multidimensional array.
To create multidimensional arrays, place each array wi
Example of a two-dimensional array:

```
int[ ][ ] sample = { {1, 2, 3}, {4, 5, 6} };
```

This declares an array with two arrays as its elements.
To access an element in the two-dimensional array, pr
another for the element inside that array.
The following example accesses the first element in th

```
int x = sample[1][0];
System.out.println(x);

// Outputs 4
```

Les deux index du tableau sont appelés indice de ligne et index de colonne.

```
int[ ][ ] myArr = { {1, 2, 3}, {4}, {5, 6, 7} };
myArr[0][2] = 42;
int x = myArr[1][0]; // 4
```

Vous pouvez obtenir et définir les éléments d'un tableau multidimensionnel en utilisant la même paire de crochets.

Le tableau à deux dimensions ci-dessus contient trois tableaux. Le premier tableau a trois éléments, le second a un seul élément et le dernier d'entre eux a trois éléments.

En Java, vous n'êtes pas limité aux tableaux bidimensionnels. Les tableaux peuvent être imbriqués dans des tableaux à autant de niveaux que votre programme a besoin. Tout ce que vous avez besoin pour déclarer un tableau avec plus de deux dimensions, est d'ajouter autant de jeux de crochets vides que vous avez besoin. Cependant, ceux-ci sont plus difficiles à maintenir. Rappelez-vous que tous les membres du tableau doivent être du même type.

Conditional statements

```
Syntax:
  if (condition) {
    //Executes when the condition is true
  }

Any of the following comparison operators can be used in the condition:
< less than
> greater than
!= not equal to
== equal to
<= less than or equal to
>= greater than or equal to

For example:

int x = 7;
if(x < 42) {
  System.out.println("Hi!");
}
```

Si l'expression est fausse, le premier ensemble de code après la fin de l'instruction if (après l'accolade fermante) est exécuté.

Vous devez utiliser deux signes égaux (==) pour tester l'égalité, car un seul signe égal est l'opérateur d'affectation

If...else (déclarations)

```
int age = 30;

if (age < 18) {
  System.out.println("Too Young");
} else {
  System.out.println("Welcome");
}

//Outputs "Welcome"
```

Nested if statements

```
int age = 25;
if(age > 0) {
  if(age > 18) {
    System.out.println("Welcome");
  } else {
    System.out.println("Too Young");
  }
} else {
  System.out.println("Error");
}
//Outputs "Welcome"
```

Vous pouvez utiliser une instruction if-else dans une autre instruction if ou else. Vous pouvez imbriquer autant d'instructions if-else que vous le souhaitez.

elseif statements

```
int age = 25;

if(age <= 0) {
  System.out.println("Error");
} else if(age > 18) {
  System.out.println("Too Young");
} else if(age < 100) {
  System.out.println("Welcome");
} else {
  System.out.println("Really?");
}
//Outputs "Welcome"
```

Le code vérifiera la condition à évaluer à vrai et exécutez les instructions à l'intérieur de ce bloc.

Logical statements

One way to accomplish this is to use nested if statements:

```
if (age > 18) {
  if (money > 500) {
    System.out.println("Welcome");
  }
}
```

However, using the **AND** logical operator (**&&**) is a better way:

```
if (age > 18 && money > 500) {
  System.out.println("Welcome");
}
```

Opérateurs logiques
Si les deux opérandes de l'opérateur ET sont vrais, la condition devient vraie.

Let's say you wanted your program to output "Welcome!" only than 18 and the variable **money** is greater than 500.
One way to accomplish this is to use nested if statements:

```
if (age > 18) {
  if (money > 500) {
    System.out.println("Welcome");
  }
}
```

The OR operateur

However, using the **AND** logical operator (**&&**) is a better way:

```
if (age > 18 && money > 500) {
  System.out.println("Welcome");
}
```

The switch statement

```
int day = 3;

switch(day) {
  case 1:
    System.out.println("Monday");
    break;
  case 2:
    System.out.println("Tuesday");
    break;
  case 3:
    System.out.println("Wednesday");
    break;
}
// Outputs "Wednesday"
```

- When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.- When a break statement is reached, the switch terminates

default statement

```
int day = 3;

switch(day) {
  case 6:
    System.out.println("Saturday");
    break;
  case 7:
    System.out.println("Sunday");
    break;
  default:
    System.out.println("Weekday");
}
// Outputs "Weekday"
```

A switch statement can have an optional default case. The default case can be used for performing a task when none of the cases is matched.

While loops

```
int x = 3;

while( x > 0) {
  System.out.println(x);
  x--;
}
/*
Outputs
3
2
1
*/
```

- Lorsque la variable est allumée est égale à un cas, les instructions suivant ce cas s'exécuteront jusqu'à ce qu'une instruction break soit atteinte.- Lorsqu'une instruction break est atteinte, le commutateur se termine

while Loops

```
int x = 6;

while( x < 10 ) {
  System.out.println(x);
  x++;
}
System.out.println("Loop ended");

/*
6
7
8
9
Loop ended
*/
```

Lorsque l'expression est testée et que le résultat est faux, le corps de boucle est ignoré et la première instruction après la boucle while est exécutée.

For Loops

```
Syntax:

for (initialization; condition; increment/decrement) {
  statement(s)
}
```

Initialisation: Expression s'exécute une seule fois au début de la boucle
Condition: Évalue chaque fois que la boucle tourne. La boucle exécute l'instruction à plusieurs reprises, jusqu'à ce que cette condition renvoie false.
Increment / Decrement: Exécute après chaque itération de la boucle.

```
for(int x = 1; x <=5; x++) {
  System.out.println(x);
}

/* Outputs
1
2
3
4
5
*/
```

Ceci initialise x à la valeur 1, et imprime à plusieurs reprises la valeur de x, jusqu'à ce que la condition x <= 5 devienne fausse. Sur chaque itération, l'instruction x ++ est exécutée, incrémentant x par un.

Increment avec "pas de 2"

```
for(int x=0; x<=10; x+=2) {
  System.out.println(x);
}
/*
0
2
4
6
8
10
*/
```

Vous pouvez avoir n'importe quel type d'état et tout type d'instructions d'incrément dans la boucle for.

do while Loops

```
int x = 1;
do {
  System.out.println(x);
  x++;
} while(x < 5);

/*
1
2
3
4
*/
```

```
int x = 1;
do {
  System.out.println(x);
  x++;
} while(x < 0);

//Outputs 1
```

Notez que la condition apparaît à la fin de la boucle, donc les instructions dans la boucle s'exécutent une fois avant qu'il soit testé. Même avec une condition fausse, le code s'exécutera une fois.

```
int x = 1;
while(x < 5) {
  System.out.println(x);
  if(x == 3) {
    break;
  }
  x++;
}

/* Outputs
1
2
3
*/
```

L'instruction break arrête la boucle et transfère l'exécution à l'instruction immédiatement après la boucle.

Continue statement

```
for(int x=10; x<=40; x+=10) {
  if(x == 30) {
    continue;
  }
  System.out.println(x);
}
/* Outputs
10
20
40
*/
```

L'instruction continue provoque la boucle pour ignorer le reste de son corps et puis retest immédiatement sa condition avant de réitération.