JAVA

* Java is an object oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs.
* Java works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.).
* It is one of the most popular programming language in the world.
* It is easy to learn and simple to use.
* It is open-source and free.
* It is secure, fast and powerful.

**Java Syntax**

Every line of code that runs in Java must be inside a class. In our example, we named the class Main. A class should always start with an uppercase first letter.  
  
**Note:** Java is case-sensitive: "MyClass" and "myclass" has different meaning.

The name of the java file must match the class name. When saving the file, save it using the class name and add ".java" to the end of the filename.

Example :

public class Main {

public static void main(String[] args) {

System.out.println("Hello World");

}

}

The main Method

The main() method is required and you will see it in every Java program:

public static void main(String[] args)

Any code inside the main() method will be executed. For now, just remember that every Java program has a class name which must match the filename, and that every program must contain the main() method.

System.out.println()

Inside the main() method, we can use the println() method to print a line of text to the screen:

public static void main(String[] args)

{

System.out.println("Hello World");

}  
  
Note: The curly braces {} marks the beginning and the end of a block of code.

System is a built-in Java class that contains useful members, such as out, which is short for "output". The println() method, short for "print line", is used to print a value to the screen (or a file).

Don't worry too much about System, out and println(). Just know that you need them together to print stuff to the screen.

You should also note that each code statement must end with a semicolon (;).

Print Text

When you are working with text, it must be wrapped inside double quotations marks "".If you forget the double quotes, an error occurs:

Example

System.out.println("This sentence will work!");

System.out.println(This sentence will produce an error);

The Print() Method

There is also a print() method, which is similar to println(). The only difference is that it does not insert a new line at the end of the output:

Example

System.out.print("Hello World!");

System.out.print("I will print on the same line.");

Print Numbers

You can also use the println() method to print numbers. However, unlike text, we don't put numbers inside double quotes:

Example

System.out.println(3);

System.out.println(358);

System.out.println(50000);

You can also perform mathematical calculations inside the println() method:

Example

System.out.println(3 + 3);

System.out.println(2 \* 5);

**Single-line Comments**

Single-line comments start with two forward slashes (//).Any text between // and the end of the line is ignored by Java (will not be executed).This example uses a single-line comment before a line of code:

// This is a comment

System.out.println("Hello World");

This example uses a single-line comment at the end of a line of code:

System.out.println("Hello World"); // This is a comment

Java Multi-line Comments

Multi-line comments start with /\* and ends with \*/.Any text between /\* and \*/ will be ignored by Java.

This example uses a multi-line comment (a comment block) to explain the code:

/\* The code below will print the words Hello World

to the screen, and it is amazing \*/

System.out.println("Hello World");

Single or multi-line comments?

It is up to you which you want to use. Normally, we use // for short comments, and /\* \*/ for longer.

Java Variables

Variables are containers for storing data values. In Java, there are different types of variables, for example:

String - stores text, such as "Hello". String values are surrounded by double quotes

int - stores integers (whole numbers), without decimals, such as 123 or -123

float - stores floating point numbers, with decimals, such as 19.99 or -19.99

char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes

boolean - stores values with two states: true or false

Declaring (Creating) Variables

To create a variable, you must specify the type and assign it a value:

Syntax

type variableName = value;

Where type is one of Java's types (such as int or String), and variableName is the name of the variable (such as x or name). The equal sign is used to assign values to the variable.

To create a variable that should store text, look at the following example:

Example

Create a variable called name of type String and assign it the value "John":

String name = "John";

System.out.println(name);

To create a variable that should store a number, look at the following example:

Example

Create a variable called myNum of type int and assign it the value 15:

int myNum = 15;

System.out.println(myNum);

You can also declare a variable without assigning the value, and assign the value later:

Example

int myNum;

myNum = 15;

System.out.println(myNum);

Other Types

A demonstration of how to declare variables of other types:

Example

int myNum = 5;

float myFloatNum = 5.99f;

char myLetter = 'D';

boolean myBool = true;

String myText = "Hello";  
  
**The general rules for naming variables are:**

* Names can contain letters, digits, underscores, and dollar signs
* Names must begin with a letter
* Names should start with a lowercase letter and it cannot contain whitespace
* Names can also begin with $ and \_
* Names are case sensitive ("myVar" and "myvar" are different variables)
* Reserved words (like Java keywords, such as int or boolean) cannot be used as names

**Java Operators**

Operators are used to perform operations on variables and values. Eg: int x = 100 + 50;

Java divides the operators into the following groups:

* Arithmetic operators
* Assignment operators
* Comparison operators
* Logical operators
* Bitwise operators

Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations.

Operator Name Description Example

+ Addition Adds together two values x + y

- Subtraction Subtracts one value from another x - y

\* Multiplication Multiplies two values x \* y

/ Division Divides one value by another x / y

% Modulus Returns the division remainder x % y

++ Increment Increases the value of a variable by 1 ++x

-- Decrement Decreases the value of a variable by 1 --x

A list of all **assignment operators**:

Operator Example Same As

= x = 5 x = 5

+= x += 3 x = x + 3

-= x -= 3 x = x - 3

\*= x \*= 3 x = x \* 3

/= x /= 3 x = x / 3

%= x %= 3 x = x % 3

&= x &= 3 x = x & 3

|= x |= 3 x = x | 3

^= x ^= 3 x = x ^ 3

>>= x >>= 3 x = x >> 3

<<= x <<= 3 x = x << 3

**Java Comparison Operators**

Comparison operators are used to compare two values. This is important in programming, because it helps us to find answers and make decisions. The return value of a comparison is either true or false.

In the following example, we use the greater than operator (>) to find out if 5 is greater than 3:

int x = 5;

int y = 3;

System.out.println(x > y); // returns true, because 5 is higher than 3

Operator Name Example

== Equal to x == y

!= Not equal x != y

> Greater than x > y

< Less than x < y

>= Greater than or equal to x >= y

<= Less than or equal to x <= y

**Java Logical Operators**

You can also test for true or false values with logical operators. Logical operators are used to determine the logic between variables or values:

Operator Name Description Example

&& Logical and Returns true if both statements are true x < 5 && x < 10

|| Logical or Returns true if one of the statements is true x < 5 || x < 4

! Logical not Reverse the result, returns false if the result is true !(x < 5 && x < 10)

**Java Strings**

Strings are used for storing text. A String variable contains a collection of characters surrounded by double quotes:

Example: Create a variable of type String and assign it a value:

String greeting = "Hello";

String Length

A String in Java is actually an object, which contain methods that can perform certain operations on strings. For example, the length of a string can be found with the length() method:

String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

System.out.println("The length of the txt string is: " + txt.length());

More String Methods

There are many string methods available, for example toUpperCase() and toLowerCase():

Example

String txt = "Hello World";

System.out.println(txt.toUpperCase()); // Outputs "HELLO WORLD"

System.out.println(txt.toLowerCase()); // Outputs "hello world"

Finding a Character in a String

The indexOf() method returns the index (the position) of the first occurrence of a specified text in a string (including whitespace):

Example

String txt = "Please locate where 'locate' occurs!";

System.out.println(txt.indexOf("locate")); // Outputs 7

Java counts positions from zero. 0 is the first position in a string, 1 is the second, 2 is the third ...

**String Concatenation**

The + operator can be used between strings to combine them. This is called concatenation:Eg:

String firstName = "John";

String lastName = "Doe";

System.out.println(firstName + " " + lastName);

You can also use the concat() method to concatenate two strings:

Eg:

String firstName = "John ";

String lastName = "Doe";

System.out.println(firstName.concat(lastName));

Strings - Special Characters

Because strings must be written within quotes, Java will misunderstand this string, and generate an error:

String txt = "We are the so-called "Vikings" from the north.";

The solution to avoid this problem, is to use the backslash escape character.The backslash (\) escape character turns special characters into string characters:

Escape Character Result Description

\' ' Single quote

\" " Double quote

\\ \ Backslash

The sequence \" inserts a double quote in a string:

Eg: String txt = "We are the so-called \"Vikings\" from the north.";

The sequence \' inserts a single quote in a string:

Example: String txt = "It\'s alright.";

The sequence \\ inserts a single backslash in a string:

Example: String txt = "The character \\ is called backslash.";

Other common escape sequences that are valid in Java are:

Code Result

\n New Line

\r Carriage Return

\t Tab

\b Backspace

\f Form Feed

**Java Conditions and If Statements**

Java supports the usual logical conditions from mathematics:

* Less than: a < b
* Less than or equal to: a <= b
* Greater than: a > b
* Greater than or equal to: a >= b
* Equal to a == b
* Not Equal to: a != b

You can use these conditions to perform different actions for different decisions.

Java has the following conditional statements:

* Use **if** to specify a block of code to be executed, if a specified condition is true
* Use **else** to specify a block of code to be executed, if the same condition is false
* Use **else if** to specify a new condition to test, if the first condition is false
* Use **switch** to specify many alternative blocks of code to be executed

The if Statement

Use the if statement to specify a block of Java code to be executed if a condition is true.

Syntax

if (condition) {

// block of code to be executed if the condition is true

} Note that if is in lowercase letters. Uppercase letters (If or IF) will generate an error.

Eg : int x = 20;

int y = 18;

if (x > y) {

System.out.println("x is greater than y");

}

The else Statement

Use the else statement to specify a block of code to be executed if the condition is false.

Syntax

if (condition) {

// block of code to be executed if the condition is true

} else {

// block of code to be executed if the condition is false

}

Example

int time = 20;

if (time < 18) {

System.out.println("Good day.");

} else {

System.out.println("Good evening.");

}

// Outputs "Good evening."

**The else if Statement**

Use the else if statement to specify a new condition if the first condition is false.

Syntax

if (condition1) {

// block of code to be executed if condition1 is true

} else if (condition2) {

// block of code to be executed if the condition1 is false and condition2 is true

} else {

// block of code to be executed if the condition1 is false and condition2 is false

}

Example

int time = 22;

if (time < 10) {

System.out.println("Good morning.");

} else if (time < 18) {

System.out.println("Good day.");

} else {

System.out.println("Good evening.");

}

// Outputs "Good evening."

**Short Hand If...Else (Ternary operator)**

There is also a short-hand if else, which is known as the ternary operator because it consists of three operands. It can be used to replace multiple lines of code with a single line, and is most often used to replace simple if else statements:

Syntax

variable = (condition) ? expressionTrue : expressionFalse;

int time = 20;

String result = (time < 18) ? "Good day." : "Good evening.";

System.out.println(result);

**Java Switch Statements**

Instead of writing many if..else statements, you can use the switch statement. The switch statement selects one of many code blocks to be executed:

Syntax

switch(expression) {

case x:

// code block

break;

case y:

// code block

break;

default:

// code block

}

This is how it works:

* The switch expression is evaluated once.
* The value of the expression is compared with the values of each case.
* If there is a match, the associated block of code is executed.

int day = 4;

switch (day) {

case 1:

System.out.println("Monday");

break;

case 2:

System.out.println("Tuesday");

break;

case 3:

System.out.println("Wednesday");

break;

case 4:

System.out.println("Thursday");

break;

case 5:

System.out.println("Friday");

break;

case 6:

System.out.println("Saturday");

break;

case 7:

System.out.println("Sunday");

break;

}

// Outputs "Thursday" (day 4)

Loops

Loops can execute a block of code as long as a specified condition is reached. Loops are handy because they save time, reduce errors, and they make code more readable.

Java While Loop

The while loop loops through a block of code as long as a specified condition is true:

Syntax

while (condition) {

// code block to be executed

}

In the example below, the code in the loop will run, over and over again, as long as a variable (i) is less than 5:

Example

int i = 0;

while (i < 5) {

System.out.println(i);

i++;}

The Do/While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

do {

// code block to be executed

}

while (condition);

The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

Example

int i = 0;

do {

System.out.println(i);

i++;

}

while (i < 5);

Java For Loop

When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop:

Syntax

for (statement 1; statement 2; statement 3) {

// code block to be executed

}

Statement 1 is executed (one time) before the execution of the code block.

Statement 2 defines the condition for executing the code block.

Statement 3 is executed (every time) after the code block has been executed.

The example below will print the numbers 0 to 4:

Example

for (int i = 0; i < 5; i++) {

System.out.println(i);

}

Nested Loops

It is also possible to place a loop inside another loop. This is called a nested loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

Example

// Outer loop

for (int i = 1; i <= 2; i++) {

System.out.println("Outer: " + i); // Executes 2 times

// Inner loop

for (int j = 1; j <= 3; j++) {

System.out.println(" Inner: " + j); // Executes 6 times (2 \* 3)

}

}

Java Break

The break statement can also be used to jump out of a loop.

Java Continue

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

Java Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with square brackets:

String[] cars;

We have now declared a variable that holds an array of strings. To insert values to it, you can place the values in a comma-separated list, inside curly braces:

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

To create an array of integers, you could write:

int[] myNum = {10, 20, 30, 40};

Access the Elements of an Array

You can access an array element by referring to the index number. This statement accesses the value of the first element in cars:

Example

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

System.out.println(cars[0]);

// Outputs Volvo

Change an Array Element

To change the value of a specific element, refer to the index number:

Example

cars[0] = "Opel";

Example

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

cars[0] = "Opel";

System.out.println(cars[0]);

// Now outputs Opel instead of Volvo

Multidimensional Arrays

A multidimensional array is an array of arrays. Multidimensional arrays are useful when you want to store data as a tabular form, like a table with rows and columns. To create a two-dimensional array, add each array within its own set of curly braces:

Example

int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

myNumbers is now an array with two arrays as its elements.

Access Elements

To access the elements of the myNumbers array, specify two indexes: one for the array, and one for the element inside that array. This example accesses the third element (2) in the second array (1) of myNumbers:

Example

int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

System.out.println(myNumbers[1][2]); // Outputs 7

Remember that: Array indexes start with 0: [0] is the first element. [1] is the second element, etc.

Change Element Values

You can also change the value of an element:

Example

int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

myNumbers[1][2] = 9;

System.out.println(myNumbers[1][2]); // Outputs 9 instead of 7

Loop Through a Multi-Dimensional Array

We can also use a for loop inside another for loop to get the elements of a two-dimensional array (we still have to point to the two indexes):

Example

public class Main {

public static void main(String[] args) {

int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

for (int i = 0; i < myNumbers.length; ++i) {

for(int j = 0; j < myNumbers[i].length; ++j) {

System.out.println(myNumbers[i][j]);

}

}

}

}

Java Methods

A method is a block of code which only runs when it is called. You can pass data, known as parameters, into a method. Methods are used to perform certain actions, and they are also known as functions.

Why use methods? To reuse code: define the code once, and use it many times.

Create a Method

A method must be declared within a class. It is defined with the name of the method, followed by parentheses (). Java provides some pre-defined methods, such as System.out.println(), but you can also create your own methods to perform certain actions:

Example

public class Main {

static void myMethod() {

// code to be executed

}

}

Call a Method

To call a method in Java, write the method's name followed by two parentheses () and a semicolon;In the following example, myMethod() is used to print a text (the action), when it is called:

Example

Inside main, call the myMethod() method:

public class Main {

static void myMethod() {

System.out.println("I just got executed!");

}

public static void main(String[] args) {

myMethod();

}

}

// Outputs "I just got executed!"