What is Java?

Java is a popular programming language, created in 1995.

It is owned by Oracle, and more than **3 billion** devices run Java.

It is used for:

* Mobile applications (specially Android apps)
* Desktop applications
* Web applications
* Web servers and application servers
* Games
* Database connection
* And much, much more!

Why Use Java?

* 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
* It has a huge community support (tens of millions of developers)
* Java is an object oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs
* As Java is close to [C++](https://www.w3schools.com/cpp/default.asp) and [C#](https://www.w3schools.com/cs/default.asp), it makes it easy for programmers to switch to Java or vice versa
* **A class should always start with an uppercase first letter.**

## 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).

## Java Multi-line Comments

Multi-line comments start with /\* and ends with \*/.

Any text between /\* and \*/ will be ignored by Java.

## 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);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_variables)

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);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_variables_int)

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

### **Example**

int myNum;

myNum = 15;

System.out.println(myNum);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_variables_int2)

Note that if you assign a new value to an existing variable, it will overwrite the previous value:

### **Example**

Change the value of myNum from 15 to 20:

int myNum = 15;

myNum = 20; // myNum is now 20

System.out.println(myNum);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_variables_ow)

## Final Variables

However, you can add the final keyword if you don't want others (or yourself) to overwrite existing values (this will declare the variable as "final" or "constant", which means unchangeable and read-only):

### **Example**

final int myNum = 15;

myNum = 20; // will generate an error: cannot assign a value to a final variable

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_variables_final)

## 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";

You will learn more about [data types](https://www.w3schools.com/java/java_data_types.asp) in the next chapter.

## Display Variables

The println() method is often used to display variables.

To combine both text and a variable, use the + character:

### **Example**

String name = "John";

System.out.println("Hello " + name);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_variables2)

You can also use the + character to add a variable to another variable:

### **Example**

String firstName = "John ";

String lastName = "Doe";

String fullName = firstName + lastName;

System.out.println(fullName);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_variables3)

For numeric values, the + character works as a mathematical operator (notice that we use int (integer) variables here):

### **Example**

int x = 5;

int y = 6;

System.out.println(x + y); // Print the value of x + y

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_variables4)

From the example above, you can expect:

* x stores the value 5
* y stores the value 6
* Then we use the println() method to display the value of x + y, which is **11**

## Declare Many Variables

To declare more than one variable of the **same type**, use a comma-separated list:

### **Example**

int x = 5, y = 6, z = 50;

System.out.println(x + y + z);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_variables5)

## Java Identifiers

All Java **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

**Note:** It is recommended to use descriptive names in order to create understandable and maintainable code:

### **Example**

// Good

int minutesPerHour = 60;

// OK, but not so easy to understand what **m** actually is

int m = 60;

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_variables_identifiers)

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 \_ (but we will not use it in this tutorial)
* 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 Data Types

As explained in the previous chapter, a variable in Java must be a specified data type:

### **Example**

int myNum = 5; // Integer (whole number)

float myFloatNum = 5.99f; // Floating point number

char myLetter = 'D'; // Character

boolean myBool = true; // Boolean

String myText = "Hello"; // String

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_data_types)

Data types are divided into two groups:

* Primitive data types - includes byte, short, int, long, float, double, boolean and char
* Non-primitive data types - such as [String](https://www.w3schools.com/java/java_strings.asp), [Arrays](https://www.w3schools.com/java/java_arrays.asp) and [Classes](https://www.w3schools.com/java/java_classes.asp) (you will learn more about these in a later chapter)

## Primitive Data Types

A primitive data type specifies the size and type of variable values, and it has no additional methods.

There are eight primitive data types in Java:

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Size** | **Description** |
| Byte | 1 byte | Stores whole numbers from -128 to 127 |
| Short | 2 bytes | Stores whole numbers from -32,768 to 32,767 |
| Int | 4 bytes | Stores whole numbers from -2,147,483,648 to 2,147,483,647 |
| Long | 8 bytes | Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| Float | 4 bytes | Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits |
| double | 8 bytes | Stores fractional numbers. Sufficient for storing 15 decimal digits |
| boolean | 1 bit | Stores true or false values |
| Char | 2 bytes | Stores a single character/letter or ASCII values |

## Numbers

Primitive number types are divided into two groups:

**Integer types** stores whole numbers, positive or negative (such as 123 or -456), without decimals. Valid types are byte, short, int and long. Which type you should use, depends on the numeric value.

**Floating point types** represents numbers with a fractional part, containing one or more decimals. There are two types: float and double.

Even though there are many numeric types in Java, the most used for numbers are int (for whole numbers) and double (for floating point numbers). However, we will describe them all as you continue to read.

## Integer Types

### **Byte**

The byte data type can store whole numbers from -128 to 127. This can be used instead of int or other integer types to save memory when you are certain that the value will be within -128 and 127:

### **Example**

byte myNum = 100;

System.out.println(myNum);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_type_byte)

### **Short**

The short data type can store whole numbers from -32768 to 32767:

### **Example**

short myNum = 5000;

System.out.println(myNum);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_type_short)

### **Int**

The int data type can store whole numbers from -2147483648 to 2147483647. In general, and in our tutorial, the int data type is the preferred data type when we create variables with a numeric value.

### **Example**

int myNum = 100000;

System.out.println(myNum);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_type_int)

### **Long**

The long data type can store whole numbers from -9223372036854775808 to 9223372036854775807. This is used when int is not large enough to store the value. Note that you should end the value with an "L":

### **Example**

long myNum = 15000000000L;

System.out.println(myNum);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_type_long)

## Floating Point Types

You should use a floating point type whenever you need a number with a decimal, such as 9.99 or 3.14515.

### **Float**

The float data type can store fractional numbers from 3.4e−038 to 3.4e+038. Note that you should end the value with an "f":

### **Example**

float myNum = 5.75f;

System.out.println(myNum);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_type_float)

### **Double**

The double data type can store fractional numbers from 1.7e−308 to 1.7e+308. Note that you should end the value with a "d":

### **Example**

double myNum = 19.99d;

System.out.println(myNum);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_type_double)

Use float or double?

The **precision** of a floating point value indicates how many digits the value can have after the decimal point. The precision of float is only six or seven decimal digits, while double variables have a precision of about 15 digits. Therefore it is safer to use double for most calculations.

### **Scientific Numbers**

A floating point number can also be a scientific number with an "e" to indicate the power of 10:

### **Example**

float f1 = 35e3f;

double d1 = 12E4d;

System.out.println(f1);

System.out.println(d1);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_type_scientific)

## Booleans

A boolean data type is declared with the boolean keyword and can only take the values true or false:

### **Example**

boolean isJavaFun = true;

boolean isFishTasty = false;

System.out.println(isJavaFun); // Outputs true

System.out.println(isFishTasty); // Outputs false

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_booleans)

Boolean values are mostly used for conditional testing, which you will learn more about in a later chapter.

## Characters

The char data type is used to store a **single** character. The character must be surrounded by single quotes, like 'A' or 'c':

### **Example**

char myGrade = 'B';

System.out.println(myGrade);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_type_char)

Alternatively, you can use ASCII values to display certain characters:

### **Example**

char myVar1 = 65, myVar2 = 66, myVar3 = 67;

System.out.println(myVar1);

System.out.println(myVar2);

System.out.println(myVar3);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_type_char_ascii)

**Tip:** A list of all ASCII values can be found in our [ASCII Table Reference](https://www.w3schools.com/charsets/ref_html_ascii.asp).

## Strings

The String data type is used to store a sequence of characters (text). String values must be surrounded by double quotes:

### **Example**

String greeting = "Hello World";

System.out.println(greeting);

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_type_string)

The String type is so much used and integrated in Java, that some call it "the special **ninth** type".

A String in Java is actually a **non-primitive** data type, because it refers to an object. The String object has methods that are used to perform certain operations on strings. **Don't worry if you don't understand the term "object" just yet**. We will learn more about strings and objects in a later chapter.

## Non-Primitive Data Types

Non-primitive data types are called **reference types** because they refer to objects.

The main difference between **primitive** and **non-primitive** data types are:

* Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).
* Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
* A primitive type has always a value, while non-primitive types can be null.
* A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.
* The size of a primitive type depends on the data type, while non-primitive types have all the same size.

## Java Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + **operator** to add together two values:

### **Example**

int x = 100 + 50;

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper)

Although the + operator is often used to add together two values, like in the example above, it can also be used to add together a variable and a value, or a variable and another variable:

### **Example**

int sum1 = 100 + 50; // 150 (100 + 50)

int sum2 = sum1 + 250; // 400 (150 + 250)

int sum3 = sum2 + sum2; // 800 (400 + 400)

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper2)

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** | **Try it** |
| + | Addition | Adds together two values | x + y | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_add) |
| - | Subtraction | Subtracts one value from another | x - y | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_sub) |
| \* | Multiplication | Multiplies two values | x \* y | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_mult) |
| / | Division | Divides one value by another | x / y | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_div) |
| % | Modulus | Returns the division remainder | x % y | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_mod) |
| ++ | Increment | Increases the value of a variable by 1 | ++x | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_inc) |
| -- | Decrement | Decreases the value of a variable by 1 | --x | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_dec) |

ADVERTISEMENT

## Java Assignment Operators

Assignment operators are used to assign values to variables.

In the example below, we use the **assignment** operator (=) to assign the value **10** to a variable called **x**:

### **Example**

int x = 10;

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_assignment1)

The **addition assignment** operator (+=) adds a value to a variable:

### **Example**

int x = 10;

x += 5;

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_assignment2)

A list of all assignment operators:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Example** | **Same As** | **Try it** |
| = | x = 5 | x = 5 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_ass1) |
| += | x += 3 | x = x + 3 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_ass2) |
| -= | x -= 3 | x = x - 3 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_ass3) |
| \*= | x \*= 3 | x = x \* 3 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_ass4) |
| /= | x /= 3 | x = x / 3 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_ass5) |
| %= | x %= 3 | x = x % 3 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_ass6) |
| &= | x &= 3 | x = x & 3 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_ass7) |
| |= | x |= 3 | x = x | 3 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_ass8) |
| ^= | x ^= 3 | x = x ^ 3 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_ass9) |
| >>= | x >>= 3 | x = x >> 3 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_ass10) |
| <<= | x <<= 3 | x = x << 3 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_ass11) |

## Java Comparison Operators

Comparison operators are used to compare two values:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Try it** |
| == | Equal to | x == y | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_compare1) |
| != | Not equal | x != y | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_compare2) |
| > | Greater than | x > y | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_compare3) |
| < | Less than | x < y | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_compare4) |
| >= | Greater than or equal to | x >= y | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_compare5) |
| <= | Less than or equal to | x <= y | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_compare6) |

## Java Logical Operators

Logical operators are used to determine the logic between variables or values:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Name** | **Description** | **Example** | **Try it** |
| && | Logical and | Returns true if both statements are true | x < 5 &&  x < 10 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_logical1) |
| || | Logical or | Returns true if one of the statements is true | x < 5 || x < 4 | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_logical2) |
| ! | Logical not | Reverse the result, returns false if the result is true | !(x < 5 && x < 10) | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_oper_logical3) |

## 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";

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_strings)

## 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:

### **Example**

String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

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

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_strings_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"

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_strings_touppercase)

## 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

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_strings_indexof)

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**:

### **Example**

String firstName = "John";

String lastName = "Doe";

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

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_strings_conc)

Note that we have added an empty text (" ") to create a space between firstName and lastName on print.

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

### **Example**

String firstName = "John ";

String lastName = "Doe";

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

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_strings_conc2)

## 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:

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_strings_double_quote)

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

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_strings_single_quote)

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

### **Example**

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

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_strings_backslash)

Six other escape sequences are valid in Java:

|  |  |  |
| --- | --- | --- |
| **Code** | **Result** | **Try it** |
| \n | New Line | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_strings_newline) |
| \r | Carriage Return | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_strings_r) |
| \t | Tab | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_strings_tab) |
| \b | Backspace | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_strings_b) |
| \f | Form Feed |  |

## Adding Numbers and Strings

WARNING!

Java uses the + operator for both addition and concatenation.

Numbers are added. Strings are concatenated.

If you add two numbers, the result will be a number:

### **Example**

int x = 10;

int y = 20;

int z = x + y; // z will be 30 (an integer/number)

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_strings_num1)

If you add two strings, the result will be a string concatenation:

### **Example**

String x = "10";

String y = "20";

String z = x + y; // z will be 1020 (a String)

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_strings_num2)

If you add a number and a string, the result will be a string concatenation:

### **Example**

String x = "10";

int y = 20;

String z = x + y; // z will be 1020 (a String)

[Try it Yourself »](https://www.w3schools.com/java/tryjava.asp?filename=demo_strings_num3)