# Java Variables

## 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 variable = value;*

Where type is one of Java's types (such as int or String), and variable 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);

[Run example »](https://www.w3schools.com/java/showjava.asp?filename=demo_variables" \t "https://www.w3schools.com/java/_blank)

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);

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);

## 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

## 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";

## 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);

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);

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

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);

## 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 isint m = 60;

The general rules for constructing names for variables (unique identifiers) 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

窗体顶端

## Exercise:

Create a variable named carName and assign the value Volvo to it.

窗体底端

# Java Data Types

## 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 numberchar myLetter = 'D'; // Characterboolean myBool = true; // BooleanString myText = "Hello"; // String

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.

## 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);

### Short

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

### Example

short myNum = 5000;

System.out.println(myNum);

### 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);

### 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);

## 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);

### 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);

### 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);

## 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

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);

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

### Example

char a = 65, b = 66, c = 67;System.out.println(a);System.out.println(b);System.out.println(c);

## 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);

## 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.

Examples of non-primitive types are [Strings](https://www.w3schools.com/java/java_strings.asp), [Arrays](https://www.w3schools.com/java/java_arrays.asp), [Classes,](https://www.w3schools.com/java/java_classes.asp)[Interface](https://www.w3schools.com/java/java_interface.asp), etc.