Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in the memory.

Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals, or characters in these variables.

There are two data types available in Java −

* Primitive Data Types
* Reference/Object Data Types

The eight primitive data types supported by the Java programming language are:

* **byte**: The byte data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive). The byte data type can be useful for saving memory in large [arrays](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html), where the memory savings actually matters. They can also be used in place of int where their limits help to clarify your code; the fact that a variable's range is limited can serve as a form of documentation.
* **short**: The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive). As with byte, the same guidelines apply: you can use a short to save memory in large arrays, in situations where the memory savings actually matters.
* **int**: By default, the int data type is a 32-bit signed two's complement integer, which has a minimum value of -231 and a maximum value of 231-1. In Java SE 8 and later, you can use the int data type to represent an unsigned 32-bit integer, which has a minimum value of 0 and a maximum value of 232-1. Use the Integer class to use int data type as an unsigned integer. See the section The Number Classes for more information. Static methods like compareUnsigned, divideUnsigned etc have been added to the [Integer](https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html) class to support the arithmetic operations for unsigned integers.
* **long**: The long data type is a 64-bit two's complement integer. The signed long has a minimum value of -263 and a maximum value of 263-1. In Java SE 8 and later, you can use the long data type to represent an unsigned 64-bit long, which has a minimum value of 0 and a maximum value of 264-1. Use this data type when you need a range of values wider than those provided by int. The [Long](https://docs.oracle.com/javase/8/docs/api/java/lang/Long.html) class also contains methods like compareUnsigned, divideUnsigned etc to support arithmetic operations for unsigned long.
* **float**: The float data type is a single-precision 32-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion, but is specified in the [Floating-Point Types, Formats, and Values](https://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html#jls-4.2.3) section of the Java Language Specification. As with the recommendations for byte and short, use a float (instead of double) if you need to save memory in large arrays of floating point numbers. This data type should never be used for precise values, such as currency. For that, you will need to use the [java.math.BigDecimal](https://docs.oracle.com/javase/8/docs/api/java/math/BigDecimal.html) class instead. [Numbers and Strings](https://docs.oracle.com/javase/tutorial/java/data/index.html) covers BigDecimal and other useful classes provided by the Java platform.
* **double**: The double data type is a double-precision 64-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion, but is specified in the [Floating-Point Types, Formats, and Values](https://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html#jls-4.2.3) section of the Java Language Specification. For decimal values, this data type is generally the default choice. As mentioned above, this data type should never be used for precise values, such as currency.
* **boolean**: The boolean data type has only two possible values: true and false. Use this data type for simple flags that track true/false conditions. This data type represents one bit of information, but its "size" isn't something that's precisely defined.
* **char**: The char data type is a single 16-bit Unicode character. It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).

In addition to the eight primitive data types listed above, the Java programming language also provides special support for character strings via the [java.lang.String](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html) class. Enclosing your character string within double quotes will automatically create a new String object; for example, String s = "this is a string";. String objects are *immutable*, which means that once created, their values cannot be changed. The String class is not technically a primitive data type, but considering the special support given to it by the language, you'll probably tend to think of it as such. You'll learn more about the String class in [Simple Data Objects](https://docs.oracle.com/javase/tutorial/java/data/index.html)

**Default Values**

It's not always necessary to assign a value when a field is declared. Fields that are declared but not initialized will be set to a reasonable default by the compiler. Generally speaking, this default will be zero or null, depending on the data type. Relying on such default values, however, is generally considered bad programming style.

The following chart summarizes the default values for the above data types.

|  |  |
| --- | --- |
| **Data Type** | **Default Value (for fields)** |
| byte | 0 |
| short | 0 |
| int | 0 |
| long | 0L |
| float | 0.0f |
| double | 0.0d |
| char | '\u0000' |
| String (or any object) | null |
| boolean | false |

## Reference Datatypes

* Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example, Employee, Puppy, etc.
* Class objects and various type of array variables come under reference datatype.
* Default value of any reference variable is null.
* A reference variable can be used to refer any object of the declared type or any compatible type.
* Example: Animal animal = new Animal("giraffe");

## Java Literals

A literal is a source code representation of a fixed value. They are represented directly in the code without any computation.

Literals can be assigned to any primitive type variable. For example −

byte a = 68;

char a = 'A';

byte, int, long, and short can be expressed in decimal(base 10), hexadecimal(base 16) or octal(base 8) number systems as well.

Prefix 0 is used to indicate octal, and prefix 0x indicates hexadecimal when using these number systems for literals. For example −

int decimal = 100;

int octal = 0144;

int hexa = 0x64;

String literals in Java are specified like they are in most other languages by enclosing a sequence of characters between a pair of double quotes. Examples of string literals are −

### **Example**

"Hello World"

"two\nlines"

"\"This is in quotes\""

String and char types of literals can contain any Unicode characters. For example −

char a = '\u0001';

String a = "\u0001";

Java language supports few special escape sequences for String and char literals as well. They are −

|  |  |
| --- | --- |
| **Notation** | **Character represented** |
| \n | Newline (0x0a) |
| \r | Carriage return (0x0d) |
| \f | Formfeed (0x0c) |
| \b | Backspace (0x08) |
| \s | Space (0x20) |
| \t | tab |
| \" | Double quote |
| \' | Single quote |
| \\ | backslash |
| \ddd | Octal character (ddd) |
| \uxxxx | Hexadecimal UNICODE character (xxxx) |