**Introduction**

**Data types** are used to classify one particular type of data in programming languages. For instance, a number and a string of characters are different types of data that will be treated differently by JavaScript.

## **Dynamic Typing**

JavaScript has dynamic data types, meaning that type checking is done at runtime rather than compile time. Python’s [data types](https://www.digitalocean.com/community/tutorials/understanding-data-types-in-python-3) are also dynamically typed.

With dynamically typed languages, a variable of the same name can be used to hold different data types.

For example, the variable t, defined as a variable by the let keyword (note that let keeps a given variable limited in scope), can be assigned to hold different data types, or can be initialized but left undefined:

let t = 16; // t is a number

let t = "Teresa"; // t is a string

let t = true; // t is a Boolean

let t; // t is undefined

Each of the variables t above can be set to any data type available in JavaScript; they do not need to be explicitly declared with a data type before they are used.

## **Numbers**

JavaScript has only one number type, there is no separate designation for integers and floating-point numbers. Because of this, numbers can be written in JavaScript with or without decimals:

let num1 = 93;

let num2 = 93.00;

Scientific exponential notation can be used in JavaScript to abbreviate very large or small numbers, as in the following examples:

let num3 = 987e8; // 98700000000

let num4 = 987e-8; // 0.00000987

Numbers in JavaScript are considered to be accurate up to 15 digits. That means that numbers will be rounded after the 16th digit is reached:

let num5 = 999999999999999; // remains as 999999999999999

let num6 = 9999999999999999; // rounded up to 10000000000000000

In addition to representing numbers, the JavaScript number type also has three symbolic values available:

* Infinity — a numeric value that represents a **positive** number that approaches infinity
* -Infinity— a numeric value that represents a **negative** number that approaches infinity
* NaN — a numeric value that represents a non-number, standing for **n**ot **a** **n**umber.

**Infinity or -Infinity will be returned if you calculate a number outside of the largest possible number available in JavaScript**. These will also occur for values that are undefined, as when dividing by zero:

* let num7 = 5 / 0; // will return Infinity
* let num8 = -5 / 0; // will return -Infinity

**For numbers that are not legal numbers, NaN will be displayed. If you attempt to perform a mathematical operation on a number and a non-numeric value, NaN will be returned.** This is the case in the following example:

let x = 20 / "Shark"; // x will be NaN

Since the number 20 cannot be divided by the string "Shark" because it cannot be evaluated as a number, the returned value for the x variable is NaN.

However, if a string can be evaluated as a numeric value, the mathematical expression can be performed in JavaScript:

let y = 20 / "5"; // y will be 4

In the above example, since the string "5" can be evaluated as a numeric value in JavaScript, it is treated as such and will work with the mathematical operator for division, /.

let a = NaN;

let b = 37;

let c = a + b; // c will be NaN

There is only one number data type in JavaScript. When working with numbers, any number you enter will be interpreted as the data type for numbers; you are not required to declare what kind of data type you are entering because JavaScript is dynamically typed.

## **Strings**

A **string** is a sequence of one or more characters (letters, numbers, symbols). Strings are useful in that they represent textual data.

In JavaScript, strings exist within either single quotes ' or double quotes ", so to create a string, enclose a sequence of characters in quotes:

let singleQuotes = 'This is a string in single quotes.';

Copy

let doubleQuotes = "This is a string in double quotes.";

Copy

You can choose to use either single quotes or double quotes, but whichever you decide on you should remain consistent within a program.

## **Booleans**

The **Boolean** data type can be one of two values, either **true** or **false**. Booleans are used to represent the truth values that are associated with the logic branch of mathematics, which informs algorithms in computer science.

Many operations in math give us answers that evaluate to either true or false:

* **greater than**
  + 500 > 100 true
  + 1 > 5 false
* **less than**
  + 200 < 400 true
  + 4 < 2 false
* **equal**
  + 5 = 5 true
  + 500 = 400 false

Like with other data types, we can store a Boolean value in a variable:

let myBool = 5 > 8; // false

## Arrays

An **array** can hold multiple values within a single variable. This means that you can contain a list of values within an array and iterate through them.

Each item or value that is inside of an array is called an **element**. You can refer to the elements of an array by using an index number.

Just as strings are defined as characters between quotes, arrays are defined by having values between square brackets [ ].

An array of strings, for example, looks like this:

let fish = ["shark", "cuttlefish", "clownfish", "eel"];

Copy

If we call the variable fish, we’ll receive the following output:

["shark", "cuttlefish", "clownfish", "eel"]

## **Objects**

The JavaScript **object** data type can contain many values as **name:value** pairs. These pairs provide a useful way to store and access data. The object literal syntax is made up of name:value pairs separated by colons with curly braces on either side { }.

Typically used to hold data that are related, such as the information contained in an ID, a JavaScript object literal looks like this, with whitespaces between properties:

let sammy = {firstName:"Sammy", lastName:"Shark", color:"blue", location:"ocean"};

Copy

Alternatively, and especially for object literals with a high number of name:value pairs, we can write this data type on multiple lines, with a whitespace after each colon:

let sammy = {

firstName: "Sammy",

lastName: "Shark",

color: "blue",

location: "Ocean"

};

The object variable sammy in each of the examples above has 4 properties: firstName, lastName, color, and location. These are each passed values separated by colons.