JavaScript Syntax

**Summary**: in this tutorial, you will learn about JavaScript syntax, including whitespace, statements, identifiers, comments, expressions, and keywords.

Whitespace

Whitespace refers to characters that provide the space between other characters. JavaScript has the following whitespace:

* Carriage return
* Space
* New Line
* tab

JavaScript engine ignores whitespace. However, you can use whitespace to format the code to make it easy to read and maintain.

The following JavaScript code doesn’t use whitespace:

let formatted = true; if (formatted) {console.log('The code is easy to read');}

Code language: JavaScript (javascript)

It’s is equivalent to the following code that uses whitespace. Hence, this code is much easy to read:

let formatted = true;

if (formatted) {

console.log('The code is easy to read');

}

Code language: JavaScript (javascript)

Note that JavaScript bundlers remove all whitespace from JavaScript files and put them into a single file for deployment. By doing this, JavaScript bundlers make the JavaScript code lighter and faster to load in the web browsers.

Statements

A statement is a code that declares a variable or instructs the JavaScript engine to do a task. A simple statement is terminated by a semicolon (;).

Although the semicolon (;) is optional; you should always use it to terminate a statement. For example, the following [declares a variable](https://www.javascripttutorial.net/javascript-variables/) and shows it to the console:

let message = "Welcome to JavaScript";

console.log(message);

Code language: JavaScript (javascript)

Blocks

A block is a sequence of zero or more simple statements. A block is delimited by a pair of curly brackets {}. For example:

if (window.localStorage) {

console.log('The local storage is supported');

}

Code language: JavaScript (javascript)

Identifiers

An identifier is a name you choose for variables, parameters, [functions](https://www.javascripttutorial.net/javascript-function/), classes, etc. An identifier name starts with a letter (a-z, or A-Z), an underscore(\_), or a dollar sign ($) and is followed by a sequence of characters including (a-z, A-Z), numbers (0-9), underscores (\_), and dollar signs ($).

Note that the letter is not limited to the ASCII character and may include extended ASCII or Unicode though not recommended.

Identifiers are case-sensitive. For example, the message is different from the Message.

Comments

Comments allow you to add notes or hints to JavaScript code. When executing the code, the JavaScript engine ignores the comments.

JavaScript supports single-line and block comments.

Single-line comments

A single-line comment starts with two forward-slashes characters (//). A single-line comment makes all the text following the // on the same line into a comment. For example:

*// this is a single-line comment*

Code language: JSON / JSON with Comments (json)

Block comments

A delimited comment begins with a forward slash and asterisk /\* and ends with the opposite \*/ as in the following example:

*/\* This is a block comment*

*that can span multiple lines \*/*

Code language: JSON / JSON with Comments (json)

Expressions

An expression is a piece of code that evaluates to a value. For example:

2 + 1

The above expression returns three.

Keywords & Reserved words

JavaScript defines a list of reserved keywords that have specific uses. Therefore, you cannot use the reserved keywords as identifiers or property names by rules.

The following table shows the JavaScript reserved words defined in ECMA-262:

|  |  |  |
| --- | --- | --- |
| [break](https://www.javascripttutorial.net/javascript-break/) | case | catch |
| [continue](https://www.javascripttutorial.net/javascript-continue/) | debugger | default |
| [else](https://www.javascripttutorial.net/javascript-if-else/) | export | [extends](https://www.javascripttutorial.net/es6/javascript-inheritance/) |
| [function](https://www.javascripttutorial.net/javascript-function/) | [if](https://www.javascripttutorial.net/javascript-if/) | [import](https://www.javascripttutorial.net/nodejs-tutorial/nodejs-es-module/) |
| new | return | super |
| throw | try | null |
| void | [while](https://www.javascripttutorial.net/javascript-while-loop/) | with |
| [class](https://www.javascripttutorial.net/es6/javascript-class/) | delete | finally |
| [in](https://www.javascripttutorial.net/javascript-for-loop/) | [switch](https://www.javascripttutorial.net/javascript-switch-case/) | typeof |
| yield | [const](https://www.javascripttutorial.net/es6/javascript-const/) | [do](https://www.javascripttutorial.net/javascript-do-while/) |
| [for](https://www.javascripttutorial.net/javascript-for-loop/) | instanceof | [this](https://www.javascripttutorial.net/javascript-this/) |
| var |  |  |

In addition to the reserved keywords, ECMA-252 also define a list of future reserved words that cannot be used as identifiers or property names:

|  |  |  |
| --- | --- | --- |
| enum | implements | let |
| protected | private | public |
| await | interface | package |
| implements | public |  |

Summary

* Use whitespace including cariage return, space, newline, and tab to format the code. The JavaScript engine ignores the whiespace.
* Use a semicolon (;) to terminate a simple statement.
* Use the curly braces ({}) to form a block that groups one or more simple statments.
* A single-line comment start with // followed by a text. A block comment begins with /\* and ends with \*/. JavaScript engine also ignores the comments.
* Identifers are names that you choose for variables, functions, classes, etc.
* Do not use the reserved keywords and reserved words for identifers.

JavaScript Variables

**Summary**: in this tutorial, you’ll learn about JavaScript variables and how to use variables to store values in the application.

A variable is a label that references a value like a number or string. Before using a variable, you need to declare it.

Declare a variable

To declare a variable, you use the var keyword followed by the variable name as follows:

var message;

Code language: JavaScript (javascript)

A variable name can be any valid identifier. By default, the message variable has a special value [undefined](https://www.javascripttutorial.net/javascript-data-types/#undefined) if you have not assigned a value to it.

Variable names follow these rules:

* Variable names are case-sensitive. This means that the message and Message are different variables.
* Variable names can only contain letters, numbers, underscores, or dollar signs and cannot contain spaces. Also, variable names must begin with a letter, an underscore (\_) or a dollar sign ($).
* Variable names cannot use the reserved words.

By convention, variable names use camelcase like message, yourAge, and myName.

JavaScript is a dynamically typed language. This means that you don’t need to specify the variable’s [type](https://www.javascripttutorial.net/javascript-data-types/) in the declaration like other static typed languages such as Java or [C#](https://www.csharptutorial.net/csharp-tutorial/csharp-variables/).

Starting in ES6, you can use the let keyword to declare a variable like this:

let message;

Code language: JavaScript (javascript)

It’s a good practice to use the let keyword to declare a variable. Later, you’ll learn the differences [between var and let keywords](https://www.javascripttutorial.net/es6/difference-between-var-and-let/). And you should not worry about it for now.

Initialize a variable

Once you have declared a variable, you can initialize it with a value. To initialize a variable, you specify the variable name, followed by an equals sign (=) and a value.

For example, The following declares the message variable and initializes it with a literal string "Hello":

let message;

message = "Hello";

Code language: JavaScript (javascript)

To declare and initialize a variable at the same time, you use the following syntax:

let variableName = value;

Code language: JavaScript (javascript)

For example, the following statement declares the message variable and initializes it with the literal string "Hello":

let message = "Hello";

Code language: JavaScript (javascript)

JavaScript allows you to declare two or more variables using a single statement. To separate two variable declarations, you use a comma (,) like this:

let message = "Hello",

counter = 100;

Code language: JavaScript (javascript)

Since JavaScript is a dynamically typed language, you can assign a value of a different type to a variable. Although, it is not recommended. For example:

let message = 'Hello';

message = 100;

Code language: JavaScript (javascript)

Change a variable

Once you initialize a variable, you can change its value by assigning a different value. For example:

let message = "Hello";

message = 'Bye';

Code language: JavaScript (javascript)

Undefined vs. undeclared variables

It’s important to distinguish between undefined and undeclared variables.

An undefined variable is a variable that has been declared but has not been initialized with a value. For example:

let message;

console.log(message); *// undefined*

Code language: JavaScript (javascript)

In this example, the message variable is declared but not initialized. Therefore, the message variable is undefined.

In contrast, an undeclared variable is a variable that has not been declared. For example:

console.log(counter);

Code language: JavaScript (javascript)

Output:

console.log(counter);

^

ReferenceError: counter is not defined

Code language: JavaScript (javascript)

In this example, the counter variable has not been declared. Hence, accessing it causes a ReferenceError.

Constants

A constant holds a value that doesn’t change. To declare a constant, you use the const keyword. When defining a constant, you need to initialize it with a value. For example:

const workday = 5;

Code language: JavaScript (javascript)

Once defining a constant, you cannot change its value.

The following example attempts to change the value of the workday constant to 4 and causes an error:

workday = 2;

Error:

Uncaught TypeError: Assignment to constant variable.

Code language: JavaScript (javascript)

Later, you’ll learn that the const keyword actually defines a read-only reference to a value in the [constants](https://www.javascripttutorial.net/es6/javascript-const/) tutorial.

Summary

* A variable is a label that references a value.
* Use the let keyword to declare a variable.
* An undefined variable is a variable that has been declared but not initialized while an undeclared variable is variable that has not been declared.
* Use the const keyword to define a readonly reference to a value.

JavaScript Data Types

**Summary**: in this tutorial, you will learn about the JavaScript data types and their unique characteristics.

JavaScript has the primitive data types:

1. [null](https://www.javascripttutorial.net/javascript-data-types/#null)
2. [undefined](https://www.javascripttutorial.net/javascript-data-types/#undefined)
3. [boolean](https://www.javascripttutorial.net/javascript-data-types/#boolean)
4. [number](https://www.javascripttutorial.net/javascript-data-types/#number)
5. [string](https://www.javascripttutorial.net/javascript-data-types/#string)
6. [symbol](https://www.javascripttutorial.net/javascript-data-types/#symbol) – available from ES2015
7. [bigint](https://www.javascripttutorial.net/es-next/javascript-bigint/) – available from ES2020

and a complex data type [object](https://www.javascripttutorial.net/javascript-data-types/#object).

JavaScript is a dynamically typed language. It means that a [variable](https://www.javascripttutorial.net/javascript-variables/) doesn’t associate with a type. In other words, a variable can hold a value of different types. For example:

let counter = 120; *// counter is a number*

counter = false; *// counter is now a boolean*

counter = "foo"; *// counter is now a string*

Code language: JavaScript (javascript)

To get the current type of the value that the variable stores, you use the [typeof](https://www.javascripttutorial.net/javascript-typeof/) operator:

let counter = 120;

console.log(typeof(counter)); *// "number"*

counter = false;

console.log(typeof(counter)); *// "boolean"*

counter = "Hi";

console.log(typeof(counter)); *// "string"*

Code language: JavaScript (javascript)

Output:

"number"

"boolean"

"string"

Code language: JSON / JSON with Comments (json)

The undefined type

The undefined type is a primitive type that has only one value undefined. By default, when a variable is declared but not initialized, it is assigned the value of undefined.

Consider the following example:

let counter;

console.log(counter); *// undefined*

console.log(typeof counter); *// undefined*

Code language: JavaScript (javascript)

In this example, the counter is a variable. Since counter hasn’t been initialized, it is assigned the value undefined. The type of counter is also undefined.

It’s important to note that the typeof operator also returns undefined when you call it on a variable that hasn’t been declared:

console.log(typeof undeclaredVar); *// undefined*

Code language: JavaScript (javascript)

The null type

The null type is the second primitive data type that also has only one value null. For example:

let obj = null;

console.log(typeof obj); *// object*

Code language: JavaScript (javascript)

The typeof null returns object is a known bug in JavaScript. A proposal to fix this was proposed but rejected. The reason was the that fix would break a lot of existing sites.

JavaScript defines that null is equal to undefined as follows:

console.log(null == undefined); *// true*

Code language: JavaScript (javascript)

The number type

JavaScript uses the number type to represent both integer and floating-point numbers.

The following statement declares a variable and initializes its value with an integer:

let num = 100;

Code language: JavaScript (javascript)

To represent a floating-point number, you include a decimal point followed by at least one number. For example:

let price= 12.5;

let discount = 0.05;

Code language: JavaScript (javascript)

Note that JavaScript automatically converts a floating-point number into an integer number if the number appears to be a whole number.

The reason is that Javascript always wants to use less memory since a floating-point value uses twice as much memory as an integer value. For example:

let price = 200.00; *// interpreted as an integer 200*

Code language: JavaScript (javascript)

To get the range of the number type, you use Number.MIN\_VALUE and Number.MAX\_VALUE. For example:

console.log(Number.MAX\_VALUE); *// 1.7976931348623157e+308*

console.log(Number.MIN\_VALUE); *// 5e-324*

Code language: JavaScript (javascript)

Also, you can use Infinity and -Infinity to represent the infinite number. For example:

console.log(Number.MAX\_VALUE + Number.MAX\_VALUE); *// Infinity*

console.log(-Number.MAX\_VALUE - Number.MAX\_VALUE); *// -Infinity*

Code language: JavaScript (javascript)

NaN

NaN stands for Not a Number. It is a special numeric value that indicates an invalid number. For example, the division of a string by a number returns NaN:.

console.log('a'/2); *// NaN;*

Code language: JavaScript (javascript)

The NaN has two special characteristics:

* Any operation with NaN returns NaN.
* The NaN does not equal any value, including itself.

Here are some examples:

console.log(NaN/2); *// NaN*

console.log(NaN == NaN); *// false*

Code language: JavaScript (javascript)

The string type

In JavaScript, a string is a sequence of zero or more characters. A string literal begins and ends with either a single quote(') or a double quote (").

A string that begins with a double quote must end with a double quote. Likewise, a string that begins with a single quote must also end with a single quote:

let greeting = 'Hi';

let message = "Bye";

Code language: JavaScript (javascript)

If you want to single quote or double quotes in a literal string, you need to use the backslash to escape it. For example:

let message = 'I\'m also a valid string'; *// use \ to escape the single quote (')*

Code language: JavaScript (javascript)

JavaScript strings are immutable. This means that it cannot be modified once created. However, you can create a new string from an existing string. For example:

let str = 'JavaScript';

str = str + ' String';

Code language: JavaScript (javascript)

In this example:

* First, declare the str variable and initialize it to a string of 'JavaScript'.
* Second, use the + operator to combine 'JavaScript' with ' String' to make its value as 'Javascript String'.

Behind the scene, the JavaScript engine creates a new string that holds the new string 'JavaScript String' and destroys the original strings 'JavaScript' and ' String'.

The following example attempts to change the first character of the string JavaScript:

let s = 'JavaScript';

s[0] = 'j';

console.log(s)

Code language: JavaScript (javascript)

The output is:

'JavaScript'

Code language: JavaScript (javascript)

But not:

'javaScript'

Code language: JavaScript (javascript)

The boolean type

The boolean type has two literal values: true and false in lowercase. The following example declares two variables that hold the boolean values.

let inProgress = true;

let completed = false;

console.log(typeof completed); *// boolean*

Code language: JavaScript (javascript)

JavaScript allows values of other types to be converted into boolean values of true or false.

To convert a value of another data type into a boolean value, you use the [Boolean()](https://www.javascripttutorial.net/javascript-boolean/) function. The following table shows the conversion rules:

| **Type** | **true** | **false** |
| --- | --- | --- |
| string | non-empty string | empty string |
| number | non-zero number and Infinity | 0, NaN |
| object | non-null object | null |
| undefined |  | undefined |

For example:

console.log(Boolean('Hi'));*// true*

console.log(Boolean('')); *// false*

console.log(Boolean(20)); *// true*

console.log(Boolean(Infinity)); *// true*

console.log(Boolean(0)); *// false*

console.log(Boolean({foo: 100})); *// true on non-empty object*

console.log(Boolean(null));*// false*

Code language: JavaScript (javascript)

The symbol type

JavaScript added a primitive type in ES6: the [symbol](https://www.javascripttutorial.net/es6/symbol/). Different from other primitive types, the symbol type does not have a literal form.

To create a symbol, you call the Symbol function as follows:

let s1 = Symbol();

Code language: JavaScript (javascript)

The Symbol function creates a new unique value every time you call it.

console.log(Symbol() == Symbol()); *// false*

Code language: JavaScript (javascript)

Note that you’ll learn more about symbols in the [symbol tutorial](https://www.javascripttutorial.net/es6/symbol/).

The bigint type

The bigint type represents the whole numbers that are larger than 253 – 1. To form a bigint literal number, you append the letter n at the end of the number:

let pageView = 9007199254740991n;

console.log(typeof(pageView)); *// 'bigint'*

Code language: JavaScript (javascript)

And you’ll learn more about the [bigint type here](https://www.javascripttutorial.net/es-next/javascript-bigint/" \t "_blank).

The object type

In JavaScript, an [object](https://www.javascripttutorial.net/home/javascript-objects/) is a collection of [properties](https://www.javascripttutorial.net/home/javascript-object-properties/), where each property is defined as a key-value pair.

The following example defines an empty object using the object literal syntax:

let emptyObject = {};

Code language: JavaScript (javascript)

The following example defines the person object with two properties: firstName and lastName.

let person = {

firstName: 'John',

lastName: 'Doe'

};

Code language: JavaScript (javascript)

A property name of an object can be any string. You can use quotes around the property name if it is not a valid identifier.

For example, if the person object has a property first-name, you must place it in the quotes such as "first-name".

A property of an object can hold an object. For example:

let contact = {

firstName: 'John',

lastName: 'Doe',

email: 'john.doe@example.com',

phone: '(408)-555-9999',

address: {

building: '4000',

street: 'North 1st street',

city: 'San Jose',

state: 'CA',

country: 'USA'

}

}

Code language: JavaScript (javascript)

The contact object has the firstName, lastName, email, phone, and address properties.

The address property itself holds an object that has building,  street, city, state, and country properties.

To access a object’s property, you can use

* The dot notation (.)
* The array-like notation ([]).

The following example uses the dot notation (.) to access the firstName and lastName properties of the contact object.

console.log(contact.firstName);

console.log(contact.lastName);

Code language: CSS (css)

If you reference a property that does not exist, you’ll get an undefined value. For example:

console.log(contact.age); *// undefined*

Code language: JavaScript (javascript)

The following example uses the array-like notation to access the email and phone properties of the contact object.

console.log(contact['phone']); *// '(408)-555-9999'*

console.log(contact['email']); *// 'john.doe@example.com'*

Code language: JavaScript (javascript)

Summary

* JavaScript has the primitive types: number, string, boolean, null, undefined, symbol and bigint and a complex type: object.

JavaScript Numbers

**Summary**: in this tutorial, you’ll learn about the JavaScript number types and how to use them effectively.

Introduction to the JavaScript Number

JavaScript uses the number type to represent both integers and floating-point values. Technically, the JavaScript number type uses the IEEE-754 format.

ES2020 introduced a new primitive type [bigint](https://www.javascripttutorial.net/es-next/javascript-bigint/" \t "_blank) representing big integer numbers with values larger than 253 – 1.

To support various types of numbers, JavaScript uses different number literal formats.

Integer numbers

The following shows how to declare a variable that holds a decimal integer:

let counter = 100;

Code language: JavaScript (javascript)

Integers can be represented in the following formats:

* Octal (base 8)
* Hexadecimal (based 16)

When you use the octal and hexadecimal numbers in arithmetic operations, JavaScript treats them as decimal numbers.

Octal numbers

An octal literal number starts with the digit zero (0) followed by a sequence of octal digits (numbers from 0 through 7). For example:

let num = 071;

console.log(num);

Code language: JavaScript (javascript)

Output:

57

If an octal number contains a number not in the range from 0 to 7, the JavaScript engine ignores the 0 and treats the number as a decimal. For example:

let num = 080;

console.log(num);

Code language: JavaScript (javascript)

Output:

80

This implicit behavior might cause issues. Therefore, ES6 introduced a new [octal literal](https://www.javascripttutorial.net/es6/octal-and-binary-literals/) that starts with the 0o followed by a sequence of octal digits (from 0 to 7). For example:

let num = 0o71;

console.log(num);

Code language: JavaScript (javascript)

Output:

57

If you an invalid number after 0o, JavaScript will issue a syntax error like this:

let num = 0o80;

console.log(num);

Code language: JavaScript (javascript)

Output:

let num = 0o80;

^^

SyntaxError: Invalid or unexpected token

Code language: JavaScript (javascript)

Hexadecimal numbers

Hexadecimal numbers start with 0x or 0X followed by any number of hexadecimal digits (0 through 9, and a through f). For example:

let num = 0x1a;

console.log(num);

Code language: JavaScript (javascript)

Output:

26

Floating-point numbers

To define a floating-point literal number, you include a decimal point and at least one number after that. For example:

let price = 9.99;

let tax = 0.08;

let discount = .05; *// valid but not recommeded*

Code language: JavaScript (javascript)

When you have a very big number, you can use e-notation. E-notation indicates a number should be multiplied by 10 raised to a given power. For example:

let amount = 3.14e7;

console.log(amount);

Code language: JavaScript (javascript)

Output:

31400000

The notation 3.14e7 means that take 3.14 and multiply it by 107.

Likewise, you can use the E-notation to represent a very small number. For example:

let amount = 5e-7;

console.log(amount);

Code language: JavaScript (javascript)

Output:

0.0000005

Code language: CSS (css)

The 5e-7 notation means that take 5 and divide it by 10,000,000.

Also, JavaScript automatically converts any floating-point number with at least six zeros after the decimal point into e-notation. For example:

let amount = 0.0000005;

console.log(amount);

Code language: JavaScript (javascript)

Output:

5e-7

Floating-point numbers are accurate up to 17 decimal places. When you perform arithmetic operations on floating-point numbers, you often get the approximate result. For example:

let amount = 0.2 + 0.1;

console.log(amount);

Code language: JavaScript (javascript)

Output:

0.30000000000000004

Code language: CSS (css)

Big Integers

JavaScript introduced the [bigint](https://www.javascripttutorial.net/es-next/javascript-bigint/) type starting in ES2022. The bigint type stores whole numbers whose values are greater than 253 – 1.

A big integer literal has the n character at the end of an integer literal like this:

let pageView = 9007199254740991n;

Code language: JavaScript (javascript)

Summary

* JavaScript Number type represents both integer and floating-point numbers.

JavaScript Numeric Separator

**Summary**: in this tutorial, you’ll how to use the JavaScript numeric separator to make the numeric literals more readable.

Introduction to the JavaScript numeric separator

The numeric separator allows you to create a visual separation between groups of digits by using underscores (\_) as separators.

For example, the following number is very difficult to read especially when it contains long digit repetitions:

const budget = 1000000000;

Code language: JavaScript (javascript)

Is this a billion or a hundred million?

The numeric separator fixes this readability issue as follows:

const budget = 1\_000\_000\_000;

Code language: JavaScript (javascript)

As you can see, the number is now very easy to interpret.

JavaScript allows you to use numeric separators for both integer and floating-point [numbers](https://www.javascripttutorial.net/javascript-number/). For example:

let amount = 120\_201\_123.05; *// 120201123.05*

let expense = 123\_450; *// 123450*

let fee = 12345\_00; *// 1234500*

Code language: JavaScript (javascript)

It’s important to note that all numbers in JavaScript are floating-point numbers.

Also, you can use the numeric separators for factional and exponent parts. For example:

let amount = 0.000\_001; *// 1 millionth*

Code language: JavaScript (javascript)

It’s important to notice that you can use the numeric separator for [bigint](https://www.javascripttutorial.net/es-next/javascript-bigint/) literal, binary literal, octal literal, and hex literal. For example:

*// BigInt*

const max = 9\_223\_372\_036\_854\_775\_807n;

*// binary*

let nibbles = 0b1011\_0101\_0101;

*// octal*

let val = 0o1234\_5670;

*// hex*

let message = 0xD0\_E0\_F0;

Code language: JavaScript (javascript)

Summary

* Use underscores (\_) as the numeric separators to create a visual separation between groups of digits.

A Quick Look at Octal and Binary Literals in ES6

**Summary**: in this tutorial, you will learn how to represent the octal and binary literals in ES6.

ES5 provided numeric literals in octal (prefix 0), decimal (no prefix), and hexadecimal (0x). ES6 added support for binary literals and changed how it represents octal literals.

Octal literals

To represent an octal literal in ES5, you use the zero prefix (0) followed by a sequence of octal digits (from 0 to 7). For example:

let a = 051;

console.log(a); *// 41*

Code language: JavaScript (javascript)

If the octal literal contains a number that is out of range, JavaScript ignores the leading 0 and treats the octal literal as a decimal, as shown in the following example:

let b = 058; *// invalid octal*

console.log(b); *// 58*

Code language: JavaScript (javascript)

In this example, since 8 is an invalid digit for representing the octal number, JavaScript ignores the 0 and treats the whole number as a decimal with a value of 58.

Note you can use the octal literals in non-strict mode. If you use them in strict mode, JavaScript will throw an error.

"use strict"

let b = 058; *// invalid octal*

console.log(b);

Code language: JavaScript (javascript)

Here is the error message:

SyntaxError: Decimals with leading zeros are not allowed in strict mode.

Code language: JavaScript (javascript)

ES6 allows you to specify the octal literal by using the prefix 0o followed by a sequence of octal digits from 0 through 7. Here is an example:

let c = 0o51;

console.log(c); *// 41*

Code language: JavaScript (javascript)

If you use an invalid number in the octal literal, JavaScript will throw a SyntaxError as shown in the following example:

let d = 0o58;

console.log(d); *// SyntaxError*

Code language: JavaScript (javascript)

Binary literals

In ES5, JavaScript didn’t provide any literal form for binary numbers. To parse a binary string, you use the parseInt() function as follows:

let e = parseInt('111',2);

console.log(e); *// 7*

Code language: JavaScript (javascript)

ES6 added support for binary literals by using the 0b prefix followed by a sequence of binary numbers (0 and 1). Here is an example:

let f = 0b111;

console.log(f); *// 7*

Code language: JavaScript (javascript)

Summary

* Octal literals start with 0o followed by a sequence of numbers between 0 and 7.
* Binary literals start with 0b followed by a sequence of number 0 and 1.

JavaScript Boolean

**Summary**: in this tutorial, you will learn about the JavaScript Boolean object and the differences between the Boolean object and the boolean primitive type.

JavaScript boolean primitive type

JavaScript provides a [boolean](https://www.javascripttutorial.net/javascript-data-types/" \l "boolean) primitive type that has two values of true and false. The following example declares two [variables](https://www.javascripttutorial.net/javascript-variables/) that hold boolean values of false and true:

let isPending = false;

let isDone = true;

Code language: JavaScript (javascript)

When you apply the  typeof operator to a variable that holds a primitive boolean value, you get the boolean as the following example:

console.log(typeof(isPending)); *// boolean*

console.log(typeof(isDone)); *// boolean*

Code language: JavaScript (javascript)

JavaScript Boolean object

In addition to the boolean primitive type, JavaScript also provides you with the global Boolean() function, with the letter B in uppercase, to cast a value of another type to boolean.

The following example shows how to use the Boolean() function to convert a string into a boolean value. Because the string is not empty, it returns true.

let a = Boolean('Hi');

console.log(a); *// true*

console.log(typeof(a)); *// boolean*

Code language: JavaScript (javascript)

The Boolean is also a wrapper object of the boolean primitive type. It means that when you pass either true or false to the Boolean constructor, it’ll create a Boolean object. For example:

let b = new Boolean(false);

Code language: JavaScript (javascript)

To get the primitive value back, you call the valueOf() method of the Boolean object as follows:

console.log(b.valueOf()); *// false*

Code language: JavaScript (javascript)

However, if you call the toString() method of a Boolean object, you get a string value "true" or "false". For example:

console.log(b.toString()); *// "false"*

Code language: JavaScript (javascript)

JavaScript boolean vs. Boolean

Consider this example:

let completed = true;

let active = new Boolean(false);

Code language: JavaScript (javascript)

First, active is an object so you can add a property to it:

active.primitiveValue = active.valueOf();

console.log(active.primitiveValue); *// false*

Code language: JavaScript (javascript)

However, you cannot do it with the primitive boolean variable like the completed variable:

completed.name = 'primitive';

console.log(completed.name); *// undefined*

Code language: JavaScript (javascript)

Second, the typeof of Boolean object returns object, whereas the typeof of a primitive boolean value returns boolean.

console.log(typeof completed); *// boolean*

console.log(typeof active); *// object*

Code language: JavaScript (javascript)

Third, when applying the  instanceof operator to a Boolean object, it returns true. However, it returns false if you apply the  instanceof operator to a boolean value.

console.log(completed instanceof Boolean); *// false*

console.log(active instanceof Boolean); *// true*

Code language: JavaScript (javascript)

It is a good practice **never** to use the Boolean object because it will create many confusions, especially when using in an expression. For example:

let falseObj = new Boolean(false);

if (falseObj) {

console.log('weird part of the Boolean object');

}

Code language: JavaScript (javascript)

How the script works.

* First, create falseObj as a Boolean object wrapper for the false value.
* Second, use falseObj in the  [if](https://www.javascripttutorial.net/javascript-if-else/) statement. Because falseObj is an object, and JavaScript engine *coerces* it to a boolean value of true. As a result, the statement inside the if block is executed.

The following table summarizes the differences between the JavaScript Boolean and boolean:

| **Operator** | **boolean** | **Boolean** |
| --- | --- | --- |
| typeof | boolean | object |
| instanceof Boolean | false | true |

It is recommended that you use the Boolean() function to convert a value of a different type to a Boolean type, but you should never use the Boolean as a wrapper object of a primitive boolean value.

In this tutorial, you have learned about the JavaScript Boolean object and the differences between the Boolean object and boolean primitive type.

JavaScript string

**Summary**: in this tutorial, you’ll learn about the JavaScript string primitive type and how to use it to define strings.

Introduction to the JavaScript strings

JavaScript strings are primitive values. Also, strings are immutable. It means that if you modify a string, you will always get a new string. The original string doesn’t change.

To create literal strings, you use either single quotes (') or double quotes (") like this:

let str = 'Hi';

let greeting = "Hello";

Code language: JavaScript (javascript)

ES6 introduced [template literals](https://www.javascripttutorial.net/es6/javascript-template-literals/) that allow you to define a string backtick (`) characters:

let name = `John`';

Code language: JavaScript (javascript)

The template literals allow you to use the single quotes and double quotes inside a string without the need of escaping them. For example:

let mesage = `"I'm good". She said";

Code language: JavaScript (javascript)

Also, you can place the [variables](https://www.javascripttutorial.net/javascript-variables/) and expressions inside a template literal. JavaScript will replace the variables with their value in the string. This is called string interpolation. For example:

let name = 'John';

let message = `Hi, I'm ${name}.`;

console.log(message);

Code language: JavaScript (javascript)

Output:

Hi, I'm John.

In this example, JavaScript replaces the name variable with its value inside the template literal.

Escaping special characters

To escape special characters, you use the backslash \ character. For example:

* Windows line break: '\r\n'
* Unix line break: '\n'
* Tab: '\t'
* Backslash '\'

The following example uses the backslash character to escape the single quote character in a string:

let str = 'I\'m a string!';

Code language: JavaScript (javascript)

Getting the length of the string

The length property returns the length of a string:

let str = "Good Morning!";

console.log(str.length); *// 13*

Code language: JavaScript (javascript)

Note that JavaScript has the [String](https://www.javascripttutorial.net/javascript-string-type/) type (with the letter S in uppercase), which is the [primitive wrapper type](https://www.javascripttutorial.net/javascript-primitive-wrapper-types/) of the primitive string type. Therefore, you can access all properties and methods of the String type from a primitive string.

Accessing characters

To access the characters in a string, you use the array-like [] notation with the zero-based index. The following example returns the first character of a string with the index zero:

let str = "Hello";

console.log(str[0]); *// "H"*

Code language: JavaScript (javascript)

To access the last character of the string, you use the length - 1 index:

let str = "Hello";

console.log(str[str.length -1]); *// "o"*

Code language: JavaScript (javascript)

Concatenating strings via + operator

To [concatenate two or more strings](https://www.javascripttutorial.net/string/javascript-string-concatenation/), you use the + operator:

let name = 'John';

let str = 'Hello ' + name;

console.log(str); *// "Hello John"*

Code language: JavaScript (javascript)

If you want to assemble a string piece by piece, you can use the += operator:

let className = 'btn';

className += ' btn-primary'

className += ' none';

console.log(className);

Code language: JavaScript (javascript)

Output:

btn btn-primary none

Code language: JavaScript (javascript)

Converting values to string

To convert a non-string value to a string, you use one of the following:

* String(n);
* ” + n
* n.toString()

Note that the toString() method doesn’t work for undefined and null.

When you convert a string to a boolean, you cannot convert it back. For example:

let status = false;

let str = status.toString(); *// "false"*

let back = Boolean(str); *// true*

Code language: JavaScript (javascript)

In this example:

* First, declare the status variable and initialize its with the value of false.
* Second, convert the status variable to a string using the toString() method.
* Third, convert the string back to a boolean value using the Boolean() function. The Boolean() function converts the string "false" to a boolean value. The result is true because "false" is a non-empty string.

Note that only string for which the Boolean() returns false, is the empty string ('');

Comparing strings

To compare two strings, you use [comparison operators](https://www.javascripttutorial.net/javascript-comparison-operators/) such as >, >=, <, <=, and == operators.

The comparison operators compare strings based on the numeric values of the characters. And it may return the string order that is different from the one used in dictionaries. For example:

let result = 'a' < 'b';

console.log(result); *// true*

Code language: JavaScript (javascript)

However:

let result = 'a' < 'B';

console.log(result); *// false*

Code language: JavaScript (javascript)

Summary

* JavaScript strings are primitive values and immutable.
* Literal strings are delimited by single quotes ('), double quotes ("), or backticks (`).
* The length property returns the length of the string.
* Use the comparison operators `>, >=, <, <=, == to compare strings.

JavaScript Objects

**Summary**: in this tutorial, you will learn about JavaScript objects and how to manipulate object properties effectively.

Introduction to the JavaScript objects

In JavaScript, an object is an unordered collection of key-value pairs. Each key-value pair is called a property.

The key of a property can be a string. And the value of a property can be any value, e.g., a [string](https://www.javascripttutorial.net/javascript-string-type/), a [number](https://www.javascripttutorial.net/javascript-number/), an [array](https://www.javascripttutorial.net/javascript-array/), and even a [function](https://www.javascripttutorial.net/javascript-function/).

JavaScript provides you with many ways to create an object. The most commonly used one is to use the object literal notation.

The following example creates an empty object using the object literal notation:

let empty = {};

Code language: JavaScript (javascript)

To create an object with properties, you use the key:value within the curly braces. For example, the following creates a new person object:

let person = {

firstName: 'John',

lastName: 'Doe'

};

Code language: JavaScript (javascript)

The person object has two properties firstName and lastName with the corresponding values 'John' and 'Doe'.

When an object has multiple properties, you use a comma (,) to separate them like the above example.

Accessing properties

To access a property of an object, you use one of two notations: the dot notation and array-like notation.

1) The dot notation (.)

The following illustrates how to use the dot notation to access a property of an object:

objectName.propertyName

Code language: CSS (css)

For example, to access the firstName property of the person object, you use the following expression:

person.firstName

Code language: CSS (css)

This example creates a person object and shows the first name and last name to the console:

let person = {

firstName: 'John',

lastName: 'Doe'

};

console.log(person.firstName);

console.log(person.lastName);

Code language: JavaScript (javascript)

2) Array-like notation ( [])

The following illustrates how to access the value of an object’s property via the array-like notation:

objectName['propertyName']

Code language: CSS (css)

For example:

let person = {

firstName: 'John',

lastName: 'Doe'

};

console.log(person['firstName']);

console.log(person['lastName']);

Code language: JavaScript (javascript)

When a property name contains spaces, you need to place it inside quotes. For example, the following address object has the 'building no' as a property:

let address = {

'building no': 3960,

street: 'North 1st street',

state: 'CA',

country: 'USA'

};

Code language: JavaScript (javascript)

To access the 'building no' property, you need to use the array-like notation:

address['building no'];

Code language: CSS (css)

If you use the dot notation, you’ll get an error:

address.'building no';

Code language: JavaScript (javascript)

Error:

SyntaxError: Unexpected string

Code language: JavaScript (javascript)

Note that it is not a good practice to use spaces in the property names of an object.

Reading from a property that does not exist will result in an [undefined](https://www.javascripttutorial.net/javascript-undefined/). For example:

console.log(address.district);

Code language: CSS (css)

Output:

undefined

Code language: JavaScript (javascript)

Modifying the value of a property

To change the value of a property, you use the [assignment operator](https://www.javascripttutorial.net/javascript-assignment-operators/) (=). For example:

let person = {

firstName: 'John',

lastName: 'Doe'

};

person.firstName = 'Jane';

console.log(person);

Code language: JavaScript (javascript)

Output:

{ firstName: 'Jane', lastName: 'Doe' }

Code language: CSS (css)

In this example, we changed the value of the firstName property of the person object from 'John' to 'Jane'.

Adding a new property to an object

Unlike objects in other programming languages such as Java and [C#](https://www.csharptutorial.net/csharp-tutorial/csharp-class/), you can add a property to an object after object creation.

The following statement adds the age property to the person object and assigns 25 to it:

person.age = 25;

Deleting a property of an object

To delete a property of an object, you use the delete operator:

delete objectName.propertyName;

Code language: JavaScript (javascript)

The following example removes the age property from the person object:

delete person.age;

Code language: JavaScript (javascript)

If you attempt to reaccess the age property, you’ll get an undefined value.

Checking if a property exists

To check if a property exists in an object, you use the in operator:

propertyName in objectName

The in operator returns true if the propertyName exists in the objectName.

The following example creates an employee object and uses the in operator to check if the ssn and employeeId properties exist in the object:

let employee = {

firstName: 'Peter',

lastName: 'Doe',

employeeId: 1

};

console.log('ssn' in employee);

console.log('employeeId' in employee);

Code language: JavaScript (javascript)

Output:

false

true

Code language: JavaScript (javascript)

Summary

* An object is a collection of key-value pairs.
* Use the dot notation ( .) or array-like notation ([]) to access a property of an object.
* The delete operator removes a property from an object.
* The in operator check if a property exists in an object.

JavaScript Primitive vs. Reference Values

**Summary**: in this tutorial, you’ll learn about two different types of values in JavaScript including primitive and reference values.

JavaScript has two different types of values:

* Primitive values
* Reference values

Primitive values are atomic pieces of data while reference values are objects that might consist of multiple values.

Stack and heap memory

When you declare [variables](https://www.javascripttutorial.net/javascript-variables/), the JavaScript engine allocates the memory for them on two memory locations: stack and heap.

Static data is the data whose size is fixed at compile time. Static data includes:

* Primitive values ([null](https://www.javascripttutorial.net/object/javascript-null/), [undefined](https://www.javascripttutorial.net/javascript-undefined/), [boolean](https://www.javascripttutorial.net/javascript-boolean-type/), [number](https://www.javascripttutorial.net/javascript-number/), [string](https://www.javascripttutorial.net/string/), [symbol](https://www.javascripttutorial.net/es6/symbol/), and [BigInt](https://www.javascripttutorial.net/es-next/javascript-bigint/))
* Reference values that refer to objects.

Because static data has a size that does not change, the JavaScript engine allocates a fixed amount of memory space to the static data and store it on the stack.

For example, the following declares two variables and initializes their values to a literal string and a number:

let name = 'John';

let age = 25;

Code language: JavaScript (javascript)

Because name and age are primitive values, the JavaScript engine stores these variables on the stack as shown in the following picture:

Note that strings are objects in many programming languages, including Java and C#. However, strings are primitive values in JavaScript.

Unlike the stack, JavaScript stores objects (and functions) on the heap. The JavaScript engine doesn’t allocate a fixed amount of memory for these objects. Instead, it’ll allocate more space as needed.

The following example defines the name, age, and person variables:

let name = 'John';

let age = 25;

let person = {

name: 'John',

age: 25,

};

Code language: JavaScript (javascript)

Internally, the JavaScript engine allocates the memory as shown in the following picture:

In this picture, JavaScript allocates memory on the stack for the three variables name, age, and person.

The JavaScript engine creates a new object on the heap memory. Also, it links the person variable on the stack memory to the object on the heap memory.

Because of this, we say that the person variable is a reference that refers to an object.

Dynamic properties

A reference value allows you to add, change, or delete properties at any time. For example:

let person = {

name: 'John',

age: 25,

};

*// add the ssn property*

person.ssn = '123-45-6789';

*// change the name*

person.name = 'John Doe';

*// delete the age property*

delete person.age;

console.log(person);

Code language: JavaScript (javascript)

Output:

{ name: 'John Doe', ssn: '123-45-6789' }

Code language: CSS (css)

Unlike a reference value, a primitive value cannot have properties. This means that you cannot add a property to a primitive value.

JavaScript allows you to add a property to a primitive value. However, it won’t take any effect. For example:

let name = 'John';

name.alias = 'Knight';

console.log(name.alias); *// undefined*

Code language: JavaScript (javascript)

Output:

undefined

Code language: JavaScript (javascript)

In this example, we add the alias property to the name primitive value. But when we access the alias property via the name primitive value, it returns undefined.

Copying values

When you assign a primitive value from one variable to another, the JavaScript engine creates a copy of that value and assigns it to the variable. For example:

let age = 25;

let newAge = age;

Code language: JavaScript (javascript)

In this example:

* First, declare a new variable age and initialize its value with 25.
* Second, declare another variable newAge and assign the age to the newAge variable.

Behind the scene, the JavaScript engine creates a copy of the primitive value 25 and assign it to the newAge variable.

The following picture illustrates the stack memory after the assignment:

On the stack memory, the newAge and age are separate variables. If you change the value of one variable, it won’t affect the other.

For example:

let age = 25;

let newAge = age;

newAge = newAge + 1;

console.log(age, newAge);

Code language: JavaScript (javascript)

When you assign a reference value from one variable to another, the JavaScript engine creates a reference so that both variables refer to the same object on the heap memory. This means that if you change one variable, it’ll affect the other.

For example:

let person = {

name: 'John',

age: 25,

};

let member = person;

member.age = 26;

console.log(person);

console.log(member);

Code language: JavaScript (javascript)

How it works.

First, declare a person variable and initialize its value with an object with two properties name and age.

Second, assign the person variable to the member variable. In the memory, both variables reference the same object, as shown in the following picture:

Third, change the age property of the object via the member variable:

Since both person and member variables reference the same object, changing the object via the member variable is also reflected in the person variable.

Summary

* Javascript has two types of values: primitive values and reference values.
* You can add, change, or delete properties to a reference value, whereas you cannot do it with a primitive value.
* Copying a primitive value from one variable to another creates a separate value copy. It means that changing the value in one variable does not affect the other.
* Copying a reference from one variable to another creates a reference so that two variables refer to the same object. This means that changing the object via one variable reflects in another variable.

JavaScript Arrays

**Summary**: in this tutorial, you’ll learn about JavaScript arrays and their basic operations.

Introduction to JavaScript arrays

In JavaScript, an array is an ordered list of values. Each value is called an *element* specified by an *index*:

A JavaScript array has the following characteristics:

1. First, an array can hold values of mixed types. For example, you can have an array that stores elements with the types number, string, boolean, and null.
2. Second, the size of an array is dynamic and auto-growing. In other words, you don’t need to specify the array size up front.

Creating JavaScript arrays

JavaScript provides you with two ways to create an array. The first one is to use the Array constructor as follows:

let scores = new Array();

Code language: JavaScript (javascript)

The scores array is empty, which does hold any elements.

If you know the number of elements that the array will hold, you can create an array with an initial size as shown in the following example:

let scores = Array(10);

Code language: JavaScript (javascript)

To create an array and initialize it with some elements, you pass the elements as a comma-separated list into the Array() constructor.

For example, the following creates the scores array that has five elements (or numbers):

let scores = new Array(9,10,8,7,6);

Code language: JavaScript (javascript)

Note that if you use the Array() constructor to create an array and pass a number into it, you are creating an array with an initial size.

However, when you pass a value of another type like string into the Array() constructor, you create an array with an element of that value. For example:

let athletes = new Array(3); *// creates an array with initial size 3*

let scores = new Array(1, 2, 3); *// create an array with three numbers 1,2 3*

let signs = new Array('Red'); *// creates an array with one element 'Red'*

Code language: JavaScript (javascript)

JavaScript allows you to omit the new operator when you use the Array() constructor. For example, the following statement creates the artists array.

let artists = Array();

Code language: JavaScript (javascript)

In practice, you’ll rarely use the Array() constructor to create an array.

The more preferred way to create an array is to use the array literal notation:

let arrayName = [element1, element2, element3, ...];

Code language: JavaScript (javascript)

The array literal form uses the square brackets [] to wrap a comma-separated list of elements.

The following example creates the colors array that holds string elements:

let colors = ['red', 'green', 'blue'];

Code language: JavaScript (javascript)

To create an empty array, you use square brackets without specifying any element like this:

let emptyArray = [];

Code language: JavaScript (javascript)

Accessing JavaScript array elements

JavaScript arrays are zero-based indexed. In other words, the first element of an array starts at index 0, the second element starts at index 1, and so on.

To access an element in an array, you specify an index in the square brackets []:

arrayName[index]

Code language: CSS (css)

The following shows how to access the elements of the mountains array:

let mountains = ['Everest', 'Fuji', 'Nanga Parbat'];

console.log(mountains[0]); *// 'Everest'*

console.log(mountains[1]); *// 'Fuji'*

console.log(mountains[2]); *// 'Nanga Parbat'*

Code language: JavaScript (javascript)

To change the value of an element, you assign that value to the element like this:

let mountains = ['Everest', 'Fuji', 'Nanga Parbat'];

mountains[2] = 'K2';

console.log(mountains);

Code language: JavaScript (javascript)

Output:

[ 'Everest', 'Fuji', 'K2' ]

Code language: JSON / JSON with Comments (json)

Getting the array size

Typically, the [length](https://www.javascripttutorial.net/javascript-array-length/) property of an array returns the number of elements. The following example shows how to use the length property:

let mountains = ['Everest', 'Fuji', 'Nanga Parbat'];

console.log(mountains.length); *// 3*

Code language: JavaScript (javascript)

Basic operations on arrays

The following explains some basic operations on arrays. And you’ll learn advanced operations such as [map()](https://www.javascripttutorial.net/javascript-array-map/), [filter()](https://www.javascripttutorial.net/javascript-array-filter/), and [reduce()](https://www.javascripttutorial.net/javascript-array-reduce/) in the next tutorials.

1) Adding an element to the end of an array

To add an element to the end of an array, you use the push() method:

let seas = ['Black Sea', 'Caribbean Sea', 'North Sea', 'Baltic Sea'];

seas.push('Red Sea');

console.log(seas);

Code language: JavaScript (javascript)

Output:

[ 'Black Sea', 'Caribbean Sea', 'North Sea', 'Baltic Sea', 'Red Sea' ]

Code language: JSON / JSON with Comments (json)

2) Adding an element to the beginning of an array

To add an element to the beginning of an array, you use the unshift() method:

let seas = ['Black Sea', 'Caribbean Sea', 'North Sea', 'Baltic Sea'];

seas.unshift('Red Sea');

console.log(seas);

Code language: JavaScript (javascript)

Output:

[ 'Red Sea', 'Black Sea', 'Caribbean Sea', 'North Sea', 'Baltic Sea' ]

Code language: JSON / JSON with Comments (json)

3) Removing an element from the end of an array

To remove an element from the end of an array, you use the pop() method:

let seas = ['Black Sea', 'Caribbean Sea', 'North Sea', 'Baltic Sea'];

const lastElement = seas.pop();

console.log(lastElement);

Code language: JavaScript (javascript)

Output:

Baltic Sea

4) Removing an element from the beginning of an array

To remove an element from the beginning of an array, you use the shift() method:

let seas = ['Black Sea', 'Caribbean Sea', 'North Sea', 'Baltic Sea'];

const firstElement = seas.shift();

console.log(firstElement);

Code language: JavaScript (javascript)

Output:

Black Sea

5) Finding an index of an element in the array

To find the index of an element, you use the [indexOf()](https://www.javascripttutorial.net/javascript-string-indexof/) method:

let seas = ['Black Sea', 'Caribbean Sea', 'North Sea', 'Baltic Sea'];

let index = seas.indexOf('North Sea');

console.log(index); *// 2*

Code language: JavaScript (javascript)

6) Check if a value is an array

To check if a value is an array, you use [Array.isArray()](https://www.javascripttutorial.net/array/how-to-check-if-a-variable-is-an-array-in-javascript/) method:

console.log(Array.isArray(seas)); *// true*

Code language: JavaScript (javascript)

Summary

* In JavaScript, an array is an order list of values. Each value is called an element specified by an index.
* An array can hold values of mixed types.
* JavaScript arrays are dynamic, which means that they grow or shrink as needed.