



BOOTCAMP

WOMEN WHO
CODE®

Becoming Full Stack Engineer

An event for women by women

Modern Javascript



Sravani Kandarpa

Software developer

Becoming a Full Stack
Engineer Bootcamp

What is JavaScript

JavaScript (JS) is a lightweight, interpreted, object-oriented language with first-class functions, and is best known as the scripting language for Web pages, but it's used in many non-browser environments as well.

It is a prototype-based, multi-paradigm scripting language that is dynamic, and supports object-oriented, imperative, and functional programming styles.

JavaScript runs on the client side of the web, which can be used to design / program how the web pages behave on the occurrence of an event. JavaScript is an easy to learn and also powerful scripting language, widely used for controlling web page behavior.



Data Types

Data Type	Definition	Example
String	Sequence of characters written inside quotes.	"Hello from the Fullstack bootcamp"
Number	represents both integer & floating-point numbers.	2022
Boolean	represents one of two values: true or false.	true
null	represents the intentional absence of any object value.	null
undefined	A variable without a value has the value undefined.	undefined
Symbol	It is used to create unique identifiers for objects.	let Sym1 = Symbol("Sym")
BigInt	values larger or less than $(2^{53}-1)$ (that's 9007199254740991	const bigInt = 12345678901234567890123456789012n;
Object	store collections of data and more complex entities	Let person= {firstname: 'Jane', surname: 'Doe'};

Variables

Statement	Description	Code example
const	Variables defined with const: <ul style="list-style-type: none"> • Cannot be Redeclared. • Cannot be Reassigned. • Have Block Scope. 	<i>const pi=3.14</i>
let	Variables defined with let: <ul style="list-style-type: none"> • Must be Declared before use. • Cannot be Redeclared. • Have Block Scope. 	<i>let bootCamp = "Module3 JS.";</i> <i>bootCamp ="Still in Module3";</i> <i>let bootCamp="Can't redeclare me"</i>
var	Variables defined with var:(before ES6) <ul style="list-style-type: none"> • Can be Redeclared • Can be Reassigned • Global or local scope 	<i>var bootCamp ="Again!! Still in Module3" ;</i> <i>Console.log("finished " + Module2);</i> <i>var Module2 ="css";</i>

Logical Operators

Operator	Description
==	Equal to
===	Equal value and equal type
!=	Not equal
!==	Not equal value or not equal type
> , < , <= , >=	Greater than, Less than, Greater than or equal to , Less than or equal to
, &&	Logical OR, Logical AND

“Falsy” Values

Value	Description
false	The keyword false
0	The number zero
-0	The number negative zero
“ ”, ‘ ’, ` `	Empty string value
null	Null - the absence of a value
undefined	Undefined - the primitive value
NaN	NaN - not a number.

All values are considered “truthy” unless they are defined as “falsy”.

Type Coercion

Type coercion is the automatic or implicit conversion of values from one data type to another (such as strings to numbers)

Here is a table with some examples:

Original Value	Converted to Number	Converted to String	Converted to Boolean
"1"	1	"1"	true
"0"	0	"0"	true
0	0	"0"	false
" "	0	" "	false
[]	0	" "	true

Nullish coalescing operator

The nullish coalescing operator (??) is a logical operator that returns its right-hand side operand when its left-hand side operand is null or undefined, and otherwise returns its left-hand side operand.

Code Example 1

```
const name = null ?? 'Isaac';
const lastName = undefined ?? 'Isaac';
```

Code Example 2

```
const number = 0 ?? 26;
const animal = "" ?? "Crocodile";
```

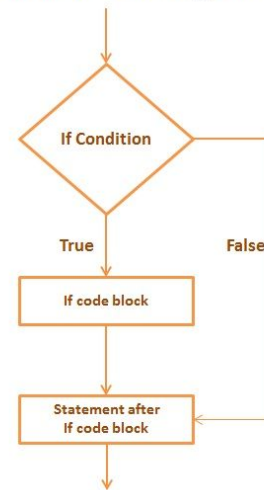
Conditional Statements

Conditional statements are used to perform different actions based on different conditions.

In JavaScript we have the following conditional statements:

- IF to specify a block of code to be executed, if a specified condition is true
- ELSE to specify a block of code to be executed, if the same condition is false
- ELSE IF to specify a new condition to test, if the first condition is false
- SWITCH to specify many alternative blocks of code to be executed

If Statement Flow Diagram



techcrashcourse.com

Conditional Statements

Use the if statement to specify a block of JavaScript code to be executed if a condition is true.

Code Example

```
if (condition1) {  
  // block of code to be executed if condition1 is true  
} else if (condition2) {  
  // block of code to be executed if the condition1 is false and condition2 is true  
} else {  
  // block of code to be executed if the condition1 is false and condition2 is false  
}
```

Conditional ternary operator

The conditional (ternary) operator is the only JavaScript operator that takes three operands: a condition, an expression to execute if truthy and an expression to execute if falsy.

The syntax is as following:

condition ? <expression if true> : <expression if false>

Code Example

```
let accessAllowed = (age > 18) ? true : false;
```

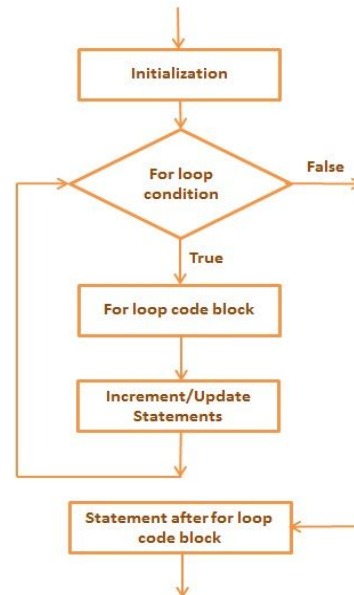
Loops

Loops offer a quick and easy way to do something repeatedly. There are many different kinds of loops, but they all essentially do the same thing: they repeat an action some number of times.

The statements for loops provided in JavaScript are:

- for
- while
- do...while
- for..in
- for..of

For Loop Flow Diagram



techcrashcourse.com

Loops for

A for loop repeats until a specified condition evaluates to false. The JavaScript for loop is similar to the Java and C for loop.

Code Example

```
for (let i = 0; i < 5; i++) {  
  let text = "The number is " + i;  
}
```

Functions

A JavaScript function is a block of code designed to perform a particular task and is executed when "something" invokes it (calls it).

Function parameters are listed inside the parentheses () in the function definition, arguments are the values received by the function when it is invoked. We can also declare default parameters.

Code Example

```
function functionName (parameters) {
  // function body
}

functionName(arg1, arg2, arg3); // function invocation

**function hoisting
```


Arrow function expressions

An arrow function expression is a compact alternative to a traditional function expression but is limited and can't be used in all situations.

```
const square = function(number) { return number * number }  
var x = square(4)
```

For example, it does not have its own bindings to this or super and should not be used as methods.

Code Example

```
const functionExample = (parameter1, parameter2) => {  
  // Code to be executed  
}  
  
functionExample(arg1, arg2);
```

Objects

The Object class represents one of JavaScript's data types. It is used to store various keyed collections and more complex entities.

Code to create a new object:

Code Example 1

```
const bootCamp = { name: "fullStack", day:2 ,conductedBy: "WomenWhoCode" }
```

Code Example 2

```
const bootCamp = {  
  day:1,  
  name: "fullStack",  
  module: function () {  
    console.log("Please!! not JS again ");  
  }  
}
```

Objects

You can access object properties in two ways:

Code Example

```
objectName.propertyName  
objectName["propertyName"]
```

You can assign and delete variables with the following syntax:

Code Example

```
bootCamp.day = 3; // assign or reassign a value  
  
delete bootCamp.day; // delete an existing value
```

Arrays

The Array object, as with arrays in other programming languages, enables storing a collection of multiple items under a single variable name, and has members for performing common array operations.

The following code example shows how to create an array using square bracket notation:

Code Example

```
const pets = ["Dog", "Cat", "Snake"];
```

You access an array element by referring to the index number:

Code Example

```
const myPet = pets[0];
```

Arrays

Few array methods

Methods	Definition
<i>push()</i>	<i>adds an element to the end.</i>
<i>pop()</i>	<i>Takes an element from the end</i>
<i>shift()</i>	<i>Extracts the first element of the array</i>
<i>unshift()</i>	<i>Add the element to the beginning of the array</i>
<i>toString()</i>	<i>Converts an array into a string of comma-separated array values</i>
<i>sort()</i>	<i>It is used to create unique identifiers for objects.</i>
<i>splice()</i>	<i>Changes the contents of an array by removing or replacing existing elements and/or adding new elements in place</i>

**slice ,indexOf, map , filter, find , reduce ,forEach*

Becoming a Full Stack
Engineer Bootcamp

Spread Syntax -Arrays

Spread syntax (...) allows an iterable such as an array expression or string to be expanded.

Code Example 1

```
const sum = (x, y, z) => {
  return x + y + z;
}
const numbers = [1, 2, 3];
console.log(sum(...numbers));
```

We can also merge two arrays using the spread syntax.

Code Example 1

```
const arr1 = [1,2,3];
const arr2 = [4,5,6];
const mergedArray = [...arr1, ...arr2];
```

Spread syntax - Object

With Rest/Spread syntax, you can spread properties of objects. It copies own enumerable properties from a provided object onto a new object.

Code Example 2

```
let obj1 = { product: "Chair", cost: 299 };  
let obj2 = { product: "Chair", cost: 349, type: "Furniture" };  
  
let clonedObj = { ...obj1 };  
  
let mergedObj = { ...obj1, ...obj2 };
```

Rest Syntax

The rest parameter syntax allows a function to accept an indefinite number of arguments as an array.

Code Example

```
const sum = (firstNumber, ...otherNumbers) => {
  console.log(firstNumber);
  console.log(otherNumbers);
};

console.log(sum(1, 2, 3));
console.log(sum(1, 2, 3, 4));
```


Object & Arrays manipulation

The `forEach()` method calls a function for each element in an array, and it is not executed for empty elements.

Code Example

```
const array = ["firstElement", "secondElement", "thirdElement"];

array.forEach((element, i) => {
  console.log(element);
  console.log(i);
});
```

Object & Arrays manipulation

The `map()` method creates a new array from calling a function for every array element, calls a function once for each element in an array, does not execute the function for empty elements, does not change the original array.

Code Example

```
// adds dollar sign to numbers

const numbers = [10, 3, 4, 6];
const dollars = numbers.map( number => '$' + number);
```

Object & Arrays manipulation

The `filter()` method creates a new array with all elements that pass the test implemented by the provided function.

Code Example

```
const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];  
  
const result = words.filter(word => word.length > 6);
```

Object & Arrays manipulation

The `reduce()` method executes a reducer function for array element, returns a single value: the function's accumulated result, does not execute the function for empty, does not change the original array.

Code Example

```
const numbers = [100, 300, 500, 70];

const sum = numbers.reduce((accumulator, value) =>
  accumulator + value
, 0);
```

Object & Arrays manipulation

The Javascript Object values() method retrieves an array of direct enumerable property values.

Code Example

```
const user = {
  age: 26,
  mobile: 8801967402131,
  name: "Jose"
}
const user = Object.values(user);
```

The entries() method returns a new Array Iterator object that contains the key/value pairs for each index in the array.

Code Example

```
const user = Object.entries(user);
```

Object & Arrays Manipulation

The `Object.keys()` method returns an array of a given object's own enumerable property names, iterated in the same order that a normal loop would.

Code Example

```
const user = Object.keys(user1);
```

Optional Chaining Operator

The optional chaining operator (?.) enables you to read the value of a property located deep within a chain of connected objects without having to check that each reference in the chain is valid.

Code Example

```
const person = {
  name: "Jane Doe",
  socialMedia: {
    linkedIn: "@myLinkedIn",
    youtube: "@myYoutubeChannel",
  },
  createInstagram: function () {
    console.log("Instagram created.");
  },
};

// non existing method, wont give an error because of optional chaining.
person.createFacebook?.();

//non existing property, wont give an error because of optional chaining.
console.log(person.socialMedia?.TikTok);
```

Destructuring assignment

The destructuring assignment syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

The following example shows array and object destructuring:

Code Example 1

```
const user = {
  id: 42,
  isVerified: true
};

const {id, isVerified} = user;
```

Code Example 2

```
const numbers = ['one', 'two', 'three'];
// name does not matter when destructuring arrays
const [red, yellow, green] = numbers;
```


Async Await

The `async` function declaration specifies an asynchronous function, which can return an `AsyncFunction` object. Async functions perform in a separate order than the rest of the code through the event loop and return a `Promise` as its result.

Code Example 1

```
const getData = async function () {  
  const response = await fetch("https://jsonplaceholder.typicode.com/users");  
  const data = await response.json();  
  console.log(data);  
};  
  
getData();
```

Imports

The static import statement is used to import read only live bindings which are exported by another module.

Here are some import examples:

Code example 1 - Importing default exports from another module

```
import defaultExport from "../module1";
```

Code example 2 - Import named exports from another module:

```
import { export1 } from "../modules/module2";
```

Code example 3 - Import named exports from another module and give them an alias:

```
import { export1 as alias1 } from "../module/module3";
```

Exports

The export statement is used when creating JavaScript modules to export live bindings to functions, objects, or primitive values from the module so they can be used by other programs with the import statement.

The following are some examples of exports:

Code example - Named export

```
export { myFunction, country };  
export let country = "Mexico"
```

Code example - Default export

```
export { myFunction as default };  
export default country
```

Export and require - node

Node.js follows the CommonJS module system, and the builtin require function is the easiest way to include modules that exist in separate files. The basic functionality of require is that it reads a JavaScript file, executes the file, and then proceeds to return the exports object

Code example - File1 - Export

```
const printMessage = (message) => {
  console.log(message);
  return;
};
const message = "This is a message!"
module.exports = {
  message,
  printMessage,
}
```

Code Example - File2 - Require

```
const Examples2 =
require("./examples2");
const message = Examples2.message;

Examples2.printMessage(message);
```

References & Resources

- [The JavaScript language](#)
- [Callbacks, Promises and Async/Await | by Sandeep Dinesh | Frontend Weekly | Medium](#)
- [JavaScript | MDN \(mozilla.org\)](#)
- [Basic JavaScript | freeCodeCamp.org](#)
- [JavaScript: Learn JavaScript, jQuery, Angular.JS & More — SitePoint](#)



BOOTCAMP

WOMEN WHO
CODE[®]

————— Thank You