Javascript ES6:

ECMAScript 2015 was the second major revision to JavaScript.

## JavaScript let:

The let keyword allows you to declare a variable with block scope.

<!DOCTYPE html>

<html>

<body>

<h2>Redeclaring a Variable Using let</h2>

<p id=*"demo"*></p>

<script>

let x = 10;

// Here x is 10

{

let x = 2;

// Here x is 2

}

// Here x is 10

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

## JavaScript const

The const keyword allows you to declare a constant (a JavaScript variable with a constant value).

Constants are similar to let variables, except that the value cannot be changed.

<!DOCTYPE html>

<html>

<body>

<h2>Declaring a Variable Using const</h2>

<p id=*"demo"*></p>

<script>

**var** x = 10;

// Here x is 10

{

**const** x = 2;

// Here x is 2

}

// Here x is 10

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

## Arrow Functions

Arrow functions allows a short syntax for writing function expressions.

You don't need the function keyword, the return keyword, and the **curly brackets**.

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Arrow Functions</h2>

<p>With arrow functions, you don't have to type the function keyword, the return keyword, and the curly brackets.</p>

<p>Arrow functions are not supported in IE11 or earlier.</p>

<p id=*"demo"*></p>

<script>

**const** x = (x, y) => x \* y;

document.getElementById("demo").innerHTML = x(5, 5);

</script>

</body>

</html>

Arrow functions do not have their own this. They are not well suited for defining **object methods**.

Arrow functions are not hoisted. They must be defined **before** they are used.

Using const is safer than using var, because a function expression is always a constant value.

You can only omit the return keyword and the curly brackets if the function is a single statement. Because of this, it might be a good habit to always keep them:

## The For/Of Loop

The JavaScript for/of statement loops through the values of an iterable objects.

for/of lets you loop over data structures that are iterable such as Arrays, Strings, Maps, NodeLists, and more.

The for/of loop has the following syntax:

for (*variable* of *iterable*) {  
  // *code block to be executed*  
}

variable - For every iteration the value of the next property is assigned to the variable. Variable can be declared with const, let, or var.

iterable - An object that has iterable properties.

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript For Of Loop</h2>

<p>The for of statement loops through the values of any iterable object:</p>

<p id=*"demo"*></p>

<script>

**const** cars = ["BMW", "Volvo", "Mini"];

let text = "";

**for** (let x of cars) {

text += x + "<br>";

}

document.getElementById("demo").innerHTML = text;

</script>

</body>

</html>

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript For Of Loop</h2>

<p>The for of statement loops through the values of an iterable object.</p>

<p id=*"demo"*></p>

<script>

let language = "JavaScript";

let text = "";

**for** (let x of language) {

text += x + "<br>";

}

document.getElementById("demo").innerHTML = text;

</script>

</body>

</html>

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Map Objects</h2>

<p>Objects as keys in a Map:</p>

<p id=*"demo"*></p>

<script>

// Create Objects

**const** apples = {name: 'Apples'};

**const** bananas = {name: 'Bananas'};

**const** oranges = {name: 'Oranges'};

// Create a new Map

**const** fruits = **new** Map();

// Add the Objects to the Map

fruits.set(apples, 500);

fruits.set(bananas, 300);

fruits.set(oranges, 200);

// Display Object Type

document.getElementById("demo").innerHTML = fruits;

</script>

</body>

</html>

## JavaScript Set Objects:

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Sets</h2>

<p>Add values to a Set:</p>

<p id=*"demo"*></p>

<script>

// Create a Set

**const** letters = **new** Set();

// Add Values to the Set

letters.add("a");

letters.add("b");

letters.add("c");

// Display set.size

document.getElementById("demo").innerHTML = letters.size;

</script>

</body>

</html>

## JavaScript Classes

JavaScript Classes are templates for JavaScript Objects.

Use the keyword class to create a class.

Always add a method named constructor():

### Syntax

class ClassName {  
  constructor() { ... }  
}

### Example

class Car {  
  constructor(name, year) {  
    this.name = name;  
    this.year = year;  
  }  
}

The example above creates a class named "Car".

The class has two initial properties: "name" and "year".

A JavaScript class is **not** an object.

It is a **template** for JavaScript objects.

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Class</h2>

<p>How to use a JavaScript Class.</p>

<p id=*"demo"*></p>

<script>

**class** Car {

constructor(name, year) {

**this**.name = name;

**this**.year = year;

}

}

**const** myCar = **new** Car("Ford", 2014);

document.getElementById("demo").innerHTML =

myCar.name + " " + myCar.year;

</script>

</body>

</html>

## Default Parameter Values

ES6 allows function parameters to have default values.

<!DOCTYPE html>

<html>

<body>

<h2>Default Parameter Values</h2>

<p id=*"demo"*></p>

<script>

**function** myFunction(x, y = 10) {

// y is 10 if not passed or undefined

**return** x + y;

}

document.getElementById("demo").innerHTML = myFunction(5);

</script>

</body>

</html>

## Function Rest Parameter

The rest parameter (...) allows a function to treat an indefinite number of arguments as an array:

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Function Rest Parameter</h2>

<p>The rest parameter (...) allows a function to treat an indefinite number of arguments as an array:</p>

<p id=*"demo"*></p>

<script>

**function** sum(...args) {

let sum = 0;

**for** (let arg of args) sum += arg;

**return** sum;

}

let x = sum(4, 9, 16, 25, 29, 100, 66, 77);

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

## String.includes()

The includes() method returns true if a string contains a specified value, otherwise false:

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript String Search</h2>

<p>Check if a string includes "world":</p>

<p id=*"demo"*></p>

<p>The includes() method is not supported in Internet Explorer.</p>

<script>

let text = "Hello world, welcome to the universe.";

document.getElementById("demo").innerHTML = text.includes("world");

</script>

</body>

</html>

## String.startsWith()

The startsWith() method returns true if a string begins with a specified value, otherwise false:

### Example

let text = "Hello world, welcome to the universe.";  
  
text.startsWith("Hello")   // Returns true

## String.endsWith()

The endsWith() method returns true if a string ends with a specified value, otherwise false:

### Example

var text = "John Doe";  
text.endsWith("Doe")    // Returns true

## Array.from()

The Array.from() method returns an Array object from any object with a length property or any iterable object.

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Arrays</h2>

<p>The Array.from() method returns an Array object from any object with a length property or any iterable object.</p>

<p id=*"demo"*></p>

<script>

**const** myArr = Array.from("ABCDEFG");

document.getElementById("demo").innerHTML = myArr;

</script>

<p>The Array.from() method is not supported in Internet Explorer.</p>

</body>

</html>

## Array find()

The find() method returns the value of the first array element that passes a test function.

This example finds (returns the value of ) the first element that is larger than 18:

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Array.find()</h2>

<p id=*"demo"*></p>

<script>

**const** numbers = [4, 9, 16, 25, 29];

let first = numbers.find(myFunction);

document.getElementById("demo").innerHTML = "First number over 18 is " + first;

**function** myFunction(value, index, array) {

**return** value > 18;

}

</script>

</body>

</html>

## New Math Methods

ES6 added the following methods to the Math object:

* Math.trunc()
* Math.sign()
* Math.cbrt()
* Math.log2()
* Math.log10()

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Math.trunc()</h2>

<p>Math.trunc(x) returns the integer part of x:</p>

<p id="demo"></p>

<script>

document.getElementById("demo").innerHTML = Math.trunc(4.7);

</script>

</body>

</html>

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Math.sign()</h2>

<p>Math.sign(x) returns if x is negative, null or positive:</p>

<p id=*"demo"*></p>

<script>

document.getElementById("demo").innerHTML = Math.sign(4);

</script>

</body>

</html>

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Math.cbrt()</h2>

<p>Math.cbrt(x) returns the cube root of x:</p>

<p id=*"demo"*></p>

<script>

document.getElementById("demo").innerHTML = Math.cbrt(8);

</script>

</body>

</html>

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Math.log2()</h2>

<p>Math.log2() returns the base 2 logarithm of a number.</p>

<p>How many times must we multiply 2 to get 8?</p>

<p id=*"demo"*></p>

<script>

document.getElementById("demo").innerHTML = Math.log2(8);

</script>

</body>

</html>

## New Number Properties

ES6 added the following properties to the Number object:

* EPSILON
* MIN\_SAFE\_INTEGER
* MAX\_SAFE\_INTEGER

<!DOCTYPE html>

<html>

<body>

<h2>Number Object Properties</h2>

<p>EPSILON</p>

<p id="demo"></p>

<script>

let x = Number.EPSILON;

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

<!DOCTYPE html>

<html>

<body>

<h2>Number Object Properties</h2>

<p>MIN\_SAFE\_INTEGER</p>

<p id="demo"></p>

<script>

let x = Number.MIN\_SAFE\_INTEGER;

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

<!DOCTYPE html>

<html>

<body>

<h2>Number Object Properties</h2>

<p>MAX\_SAFE\_INTEGER</p>

<p id="demo"></p>

<script>

let x = Number.MAX\_SAFE\_INTEGER;

document.getElementById("demo").innerHTML = x;

</script>

</body>

</html>

## New Number Methods

ES6 added 2 new methods to the Number object:

* Number.isInteger()
* Number.isSafeInteger()

## The Number.isInteger() Method

The Number.isInteger() method returns true if the argument is an integer.

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Number Methods</h2>

<p>The isInteger() method returns true if the argument is an integer.</p>

<p>Otherwise it returns false.</p>

<p id="demo"></p>

<script>

document.getElementById("demo").innerHTML =

Number.isInteger(10) + "<br>" + Number.isInteger(10.5);

</script>

</body>

</html>

## The Number.isSafeInteger() Method

A safe integer is an integer that can be exactly represented as a double precision number.

The Number.isSafeInteger() method returns true if the argument is a safe integer.

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Number Methods</h2>

<p>The isSafeInteger() method returns true if the argument is a safe integer.</p>

<p>Otherwise it returns false.</p>

<p id="demo"></p>

<script>

document.getElementById("demo").innerHTML =

Number.isSafeInteger(10) + "<br>" + Number.isSafeInteger(12345678901234567890);

</script>

</body>

</html>

## New Global Methods

ES6 added 2 new global number methods:

* isFinite()
* isNaN()

## The isFinite() Method

The global isFinite() method returns false if the argument is Infinity or NaN.

Otherwise it returns true:

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Number Methods</h2>

<p>The isFinite() method returns false if the argument is Infinity or NaN.</p>

<p>Otherwise it returns true.</p>

<p id="demo"></p>

<script>

document.getElementById("demo").innerHTML =

isFinite(10 / 0) + "<br>" + isFinite(10 / 1);

</script>

</body>

</html>

## The isNaN() Method

The global isNaN() method returns true if the argument is NaN. Otherwise it returns false:

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Number Methods</h2>

<p>The isNan() method returns true if the argument is NaN. Otherwise it returns false.</p>

<p id="demo"></p>

<script>

document.getElementById("demo").innerHTML =

isNaN("Hello") + "<br>" + isNaN("10");

</script>

</body>

</html>