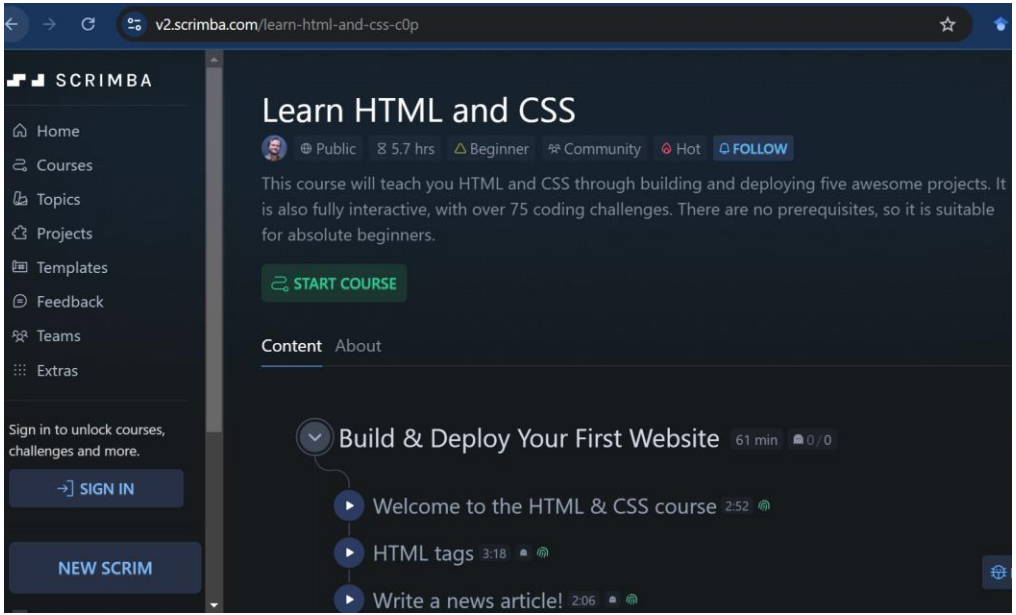
Introduction to JavaScript

2024-2025

Contents

0. Prerrequisites

This course is for students that have completed an introductory course to HTML and CSS. If you need a quick refresher on these topics we advice the HTML & CSS crash course at freeCodeCamp. (Full video course available at Youtube)

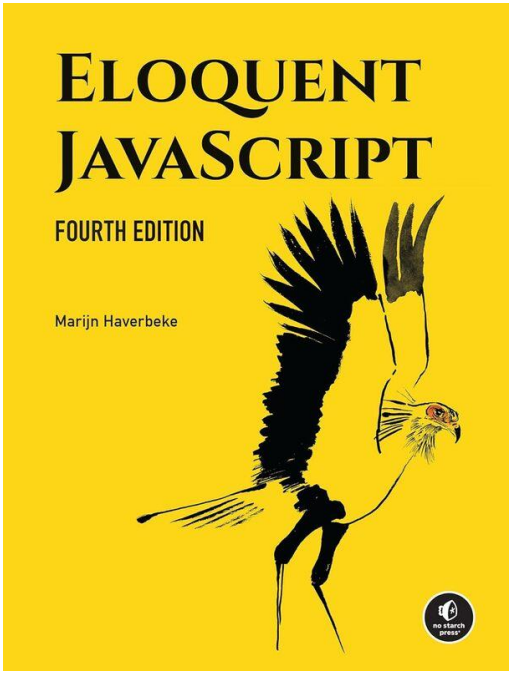


0. Documentation

Book

Excellent Open Source work created by Marijn Haverbeke. It is available for free download and you may purchase a printed copy (the fourth edition will not be available printed until November 2024).

It covers in detail all the JavaScript language with lots of examples that are available in the github repo. It is intended for an audience that has some background in programming.



https://eloquentjavascript.net/

0. Documentation

MDN Web Docs

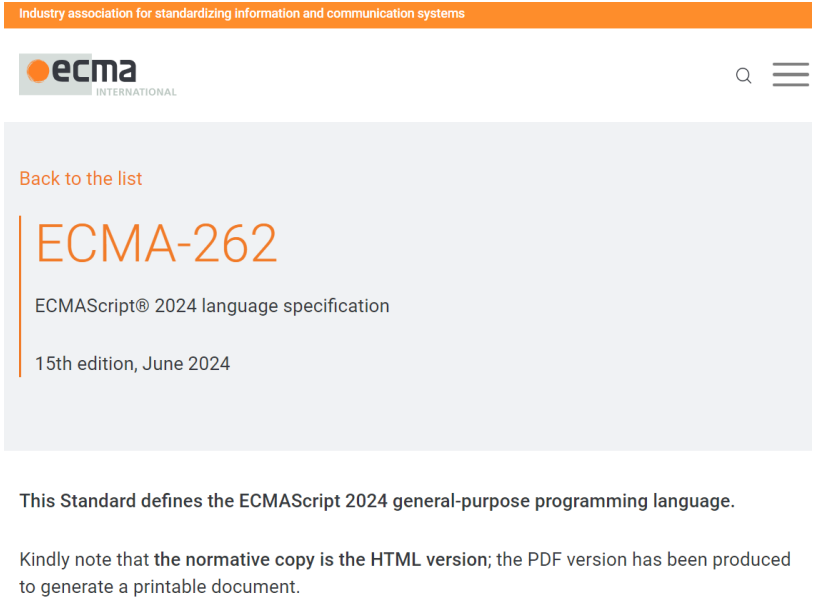
MDN Web Docs (formerly known as the Mozilla Developer Network or MDN) is a free resource for in-depth documentation on web standards such as HTML5, CSS, JavaScript, and much more. You may have a look at the HTML & CSS courses. All the examples that appear in these slides with grey background are taken from MDN Web Docs. The full course is pretty well written and paced.

https://developer.mozilla.org/en-US/docs/Learn/JavaScript

0. Documentation

ECMA documentation

An engineer must be able to read the original technical documentation. The ECMA standard is the ultimate source for any doubt about the JavaScript language.



https://ecma-international.org/publications-and-standards/standards/ecma-262/

1. Introduction

History

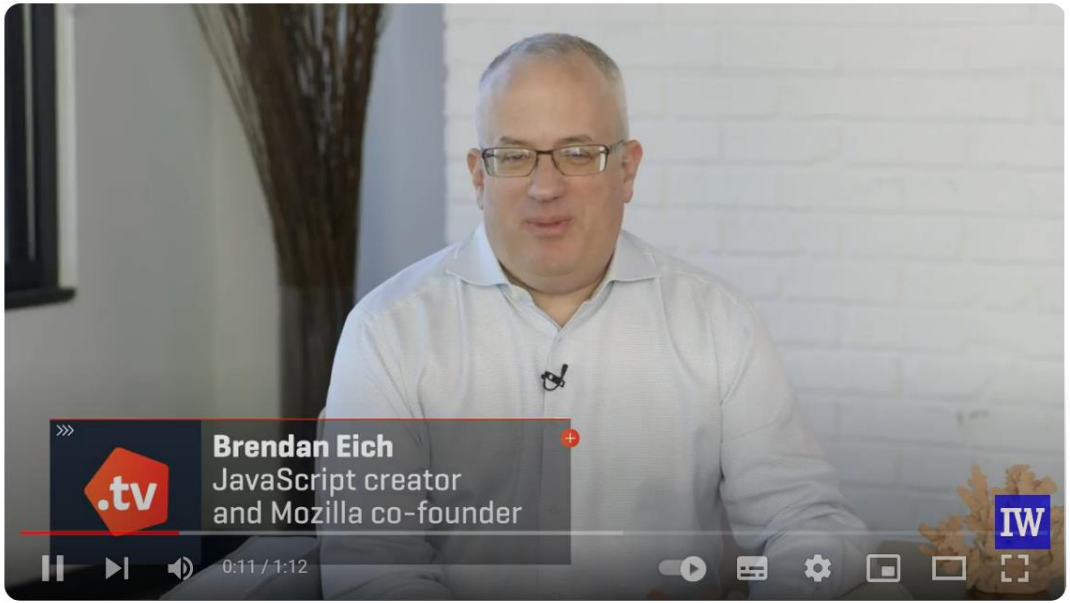
JavaScript was created in just 10 days in May 1995, by Brendan Eich, when he was working at Netscape. He aimed to add interactive behavior to the static HTML pages that were a vast majority by then. The original script features were very limited, and professional developers outside the web sphere, regarded JavaScript as a toy language.

The original name was Mocha, that changed first to LiveScript by September 1995 and JavaScript by December 1995. Microsoft developed its own scripting called JScript, and even it change later to JavaScript, there were blatant incompatibilities among different browsers and releases.

It was the dark age of Browser Wars…

```
if (/\bMSIE 6/.test(navigator.userAgent) && !window.opera) {
  // yep, browser claims to be IE6
}
```

1. Introduction



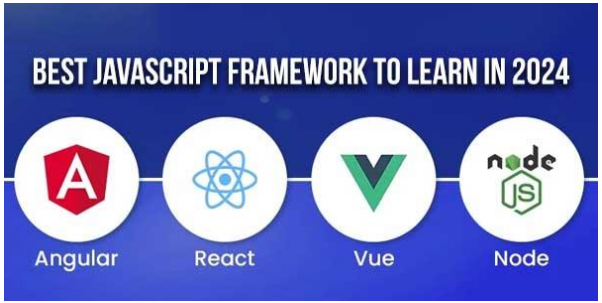
https://www.youtube.com/watch?v=kB32-Cvj0X4

1. Introduction

History

Later it was standardized under the name ECMAScript. The ECMA-262 set of specifications define the JavaScript language and today is a powerful language programming that powers up the web but is found as well in servers (NodeJS) and desktop applications.

It is the one of the most popular programming languages by 2024, according to different rankings such as TIOBE and IEEE Spectrum, because of the pervasive nature of the web. It is almost synonymous of “front end development”.



1. Introduction

JavaScript

Is a Just-in-time compiled language JIT.

Dynamically and weakly typed in contrast with strongly typed languages as Java or C

Internally, everything is an object, but it is not a purely OOP language like Java

Memory is garbage-collected

Single-threaded, concurrency is managed trough a clever asynchronous mechanism (events loop)

Multiparadigm, supports procedural, OOP and functional programming. Modern JavaScript has a strong FP flavor, with functions of first order (functions work as variables).

The syntax of common constructions is based on C

- if, if else, switch
- for, while, do while
- return, break, continue

1. Introduction

Why We Should Stop Using JavaScript According to Douglas Crockford (Inventor of JSON)

https://www.youtube.com/watch?v=lc5Np9OqDHU

1. Introduction

JavaScript

```
1   console.log("Hello World, this is JavaScript!");
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS ...

```
C:\Program Files\nodejs\node.exe .\hello.js
Hello World, this is JavaScript!
```

2. JavaScript Language

Syntax

- It is case sensitive, so the identifier `myLet` is different from `mylet`
- As a weakly typed language you do not define the type of a variable on declaration. This is valid in JavaScript:

```
1   let myvar = 8;
2   myvar = {Name: "John", Age: 46};
```

- Sentences end with a semicolon (;). Even it is not compulsory it is always a good programming practice.
- Comments.

```
// This is a line comment
/* This is a
   block-comment
   instead */
```

2. JavaScript Language

variables

- Identifier

  It is the name of the variable. It is a combination of digits, letters and symbols **$** y **_**. The first character may not be a digit. You can't use a reserved word of the language as an identifier.

- There are three ways to declare variables.

```
var number = 3; // Old way to declare a variable, try not to use it when writing new code
let anothernumber = 5;  // let is the proper way to declare variables after 2015
let edad;       // You can declare a variable and not give initial value
const PI = 3.141592  // const declares a constant reference but its behaviour is quite subtle
```

2. JavaScript Language

const

- You can use it to store a single value, an object or an array. When use with objets or arrays it is possible to modified them but never with the assignment operator. It is the preferable way to declare an object it it doesn`t need to be reassigned. Like let they are blocked-scoped local variables.

```
const myObject = {
    name: "John",
    age: 24
}
myObject.age = 25 //correct

myObject = { //NOT ALLOWED
    name: "Marc",
    age: 19
}
```

2. JavaScript Language

Reserved words					
abstract	debugger	final	instanceof	protected	transient
boolean	default	finally	int	public	true
break	delete	float	interface	return	try
byte	do	for	let	short	typeof
case	double	function	long	static	let
catch	else	goto	native	super	void
char	enum	if	new	switch	volatile
class	export	implements	null	synchronized	while
const	extends	import	package	this	with
continue	false	in	private	throw	yield

2. JavaScript Language

Data Types

- Simple
  - Number
  - String
  - Boolean
  - Null
  - Undefined
- Objects
  - Arrays
  - Regular Expressions
  - Functions
  - Objects

2. JavaScript Language

Data Types

`Number`

- 64 bit IEEE 754 format both for integer and decimal numbers
- JavaScript handles two special values for numbers: `NaN (not a number) and Infinite`

In ES6 there is also a BigInt type to deal with huge quantities

`String`

Any textual information is stored as an string, even individual characters. You may use double quotes, single quotes or back ticks(normally use with template literals) to enclose the text

```
let srting1 = "String defined with double quotes"
let string2 = 'String defined with single quotes'
let string3 = `You may use back ticks as well`
```

2. JavaScript Language

Template Strings

To use an expression or variable inside an string we use template strings. In this case it is not possible to use single or doble quotes, it is mandatory to use back ticks.

You can use ${expresion} to perform substitutions form embeded expressions.

```
console.log(`Hello ${myObject.name}`);
//is the same than
console.log ("Hello" + myObject.name);
```

2. JavaScript Language

```
let s = "John Wayne";
let firstName = s.substring(0, s.indexOf(" "));
let len = s.length;              // 10
let s2 = ´Clint Eastwood´;   // can use "" or ''
```

String methods: `charAt, charCodeAt, fromCharCode, indexOf, lastIndexOf, replace, split, substring, toLowerCase, toUpperCase`

`charAt` returns a one-letter string (there is no char type)

`length` is a property (not a method as in Java)

concatenation with + : 1 + 1 is 2, but "1" + 1 is "11" ("1" - 1 is 0)

strings can be compared with <, <=, ==, !=, >, >=

2. JavaScript Language

Data Types

`Boolean`

Two possible values: true, false. Non boolean types are evaluated as true except 0, false, null or empty string (“”).

`Arrays`

A set of indexed variables of any type.

For instance, a string is an array of charactersDeclaración e inicialización

```
let colors = ["blue", "red", "yellow"]
let matrix = [[2, 1], [7, 8]]
let mix = ["London", 1948, -8.73]
```

2. JavaScript Language

Operators

Assignment	Increase/Decrease	Comparison	Arithmetic	Logical
=	++	===	+ add (numbers)	&& (and)
+=	--	!==	+ concat (strings)	|| (or)
-=		<	-	! (not)
*=	++a (post increment)	<=	* (multiplication)	** (exponentiation)
/=		>	/ quotient	
%=	a-- (post decrement)	>=	% modulus	

2. JavaScript Language

Flow control sentences

```
if … else          // Identical to C flow control

     if ( condition ) {     // 0, "" and null are converted to false
                  // Block of code
     }
     else {
                  // A different blok of code
          }
```

for // There is one way to build for loops identical to for loops in C, but there are as well for .. of and for .. in for objects and arrays.

```
let result = 0;
for (let n=1; n< 10; n++)
    result += n;
```

2. JavaScript Language

`for … in`

```
for ( index in array ) {
        // Code
}
```

```
1  weekend = ["Saturday", "Sunday"];
2  // C like iteration of array
3  for (let i = 0; i < weekend.length; i++)
4      console.log(weekend[i]);
5
6  // for .. in iteration
7  for (let index in weekend)
8      console.log(weekend[index]);
9
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
C:\Program Files\nodejs\node.exe .\lesson1_B.js
Saturday
Sunday
Saturday
Sunday
```

2. JavaScript Language

Flow control sentences

`while` `do … while`

```
// Both loops are identical to
// their C counterparts
```

```
10  let sum = 0;
11  while (sum < 100){
12      sum = sum + 1;
13  }
14  console.log("The sum is %d",sum);
15
16  sum = 100;
17  do {
18      sum = sum + 1
19  } while (sum < 100);
20  console.log("The sum is "+sum);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS ···

```
C:\Program Files\nodejs\node.exe .\lesson1_B.js
The sum is 100
The sum is 101
```

2. JavaScript Language

Flow control sentences

```
switch     // Identical to c switch, but as string is a simple type it
           //  works with strings as well
```

```
switch ( expression ) {
    case value1:
       // code for value 1
    break; // to break thw switch clause
    case value2:
       // code for value 2
    break;
    default: // optional
       // Any other value
}
```

2. JavaScript Language

```
1  weekday = "Thursday";
2  let mymsg = "";
3  switch (weekday){
4      case "Monday":
5          mymsg = "is blue";
6          break;
7      case "Tuesday","Wednesday":
8          mymsg = "is grey";
9          break;
10     case "Thursday":
11         mymsg = "I don't care about you";
12         break;
13     case "Friday":
14         mymsg = "I'm in love";
15         break;
16     default:
17         mymsg = "Go ask The Cure how should you feel.";
18 }
19 console.log(weekday,mymsg);
20
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  ...
C:\Program Files\nodejs\node.exe .\lesson1_D.js
Thursday I don't care about you
```

https://www.youtube.com/watch?v=mGgMZpGYiy8

2. JavaScript Language

Dealing with exceptions

JavaScript provides a `try…catch` syntax very similar to Java to deal with exceptions. To raise an exception you use the `throw()` method.

```
function divide(dividend, divisor) {
    if (divisor === 0)
      throw new Error("Division by zero");
    if (divisor === undefined)   // You only provide one input parameter
      throw new Error("Syntax error");
    console.log(dividend / divisor);
  }

  try {
    divide(7,5);
    divide(9,0);
    divide(8);
  } catch (error) {
    console.log("Error: ", error.message);
  }
```

2. JavaScript Language

Functions

JS functions are first-class objects. That means you can manage them as any conventional object:

- Store its reference in a variable
- Store a list of function as an array
- Use a function as the input parameter of a different function and return a function as the result of the execution
- Define nested functions
- Define functions that can take an undefined number of parameters.
- Store properties inside functions
- Define anonymous functions.

2. JavaScript Language

Functions

Definition

```
function name(paramName, ..., paramName) {
    statements;
}
```

example:

```
    function AddTo(n) {
        let sum = 0;
        for (let i = 1; i <= n; i++) {
            sum += i;
        }
        return sum;
    }
```

2. JavaScript Language

Functions

```
function maybeReturn(n) {
    if (n % 2 == 0) {
        return "even";
    }
    // else return undefined
}
```

Parameter and return types are not declared

- the function can return anything it wants
- if a function returns nothing, it returns `undefined`
- a function can sometimes return values and sometimes not
- functions may be unnamed

2. JavaScript Language

Functions

Arrow functions were added in ECMA 6 to improve functional programming capabilities. They are just a different type of syntax, but behave as conventional functions. To giving it a name uses a const reference.

```
function mysum(a,b)    // conventional definition
{
    return a + b;
}

// Is equivalent to the arrow syntax
const mysumarr = (a,b) => a+b;
```

2. JavaScript Language

Functions

Arrow functions may be unnamed, and when there is just one input parameter the parenthesis may be omited.

```
function square(x)    // conventional definition
{
    return x * x;
}
let mynumbers = [1,2,4,5]
let mysquares = [];
for (i=0; i < mynumbers.length; i++)
    mysquares.push(square(mynumbers[i]));
console.log("Result with conventional function", mysquares.toString());

// Same code with arrow function and array map
mysquares = mynumbers.map(x=>x*x);
console.log("Now with arrow function "+mysquares.toString());
```

2. JavaScript Language

Arrow functions

It is possible to ommit the curly braces and the return word when there is only one statement with the return expression.

```
const square = x => x*x;

//is the same than
const square2 = (x) => {return x*x};

function square3 (x) {
    return x*x;
}
```

2. JavaScript Language

Closures

A closure refers to a function that retains access to its lexical scope, even when that function is executed outside its original scope. Essentially, a closure allows a function to "remember" the environment in which it was created, including any variables or input parameters that were in scope at that time.

Key Characteristics of Closures:

- **Function Within a Function**. A closure is created when a function is defined inside another function. The inner function has access to the variables of the outer function
- **Access to Outer Variables**: The inner function retains access to the outer function's variables even after the outer function has finished executing. This means that the inner function can continue to use those variables.

2. JavaScript Language

Closures

In this example, we use a closure to generate two different functions that yield sequences on user IDs.

```
1  function getUserID(country) {
2      let count = 0; // This is a local variable
3
4      return function() {
5          count += 1; // The inner function has access to `count`
6          return country+count;
7      };
8  }
9  const userID_ES = getUserID("ES"); // `userID` is now a closure
10 const userID_FR = getUserID("FR"); // `userID` is now a closure
11
12 console.log(userID_ES());
13 console.log(userID_ES());
14 console.log(userID_FR());
15 console.log(userID_ES());
16
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  ...  Filter (e.g. text, !exclude, \escape)
C:\Program Files\nodejs\node.exe .\closure_id.js
ES1
ES2
FR1
ES3
```

2. JavaScript Language

Closures

In this second example, the closure returns an object with the natural and decimal logarithm functions.

```
function createLogFunctions() {
    // Inner function to compute the natural logarithm (base e)
    function naturalLog(x) {
        return Math.log(x);
    }
    // Inner function to compute the decimal logarithm (base 10)
    function decimalLog(x) {
        return Math.log10(x);
    }
    // Return an object with both functions
    return {
        naturalLog: naturalLog,
        decimalLog: decimalLog
    };
}
// Create an instance of the log functions
const logFunctions = createLogFunctions();
// Using the natural logarithm function
console.log("Natural Log of 10:", logFunctions.naturalLog(10)); // Output: 2.302585092994046
// Using the decimal logarithm function
console.log("Decimal Log of 10:", logFunctions.decimalLog(10)); // Output: 1
```

2. JavaScript Language

Classes and Objects

With ES6, JavaScript provides a syntax for OOP quite similar to Java.

Class - a template – Teacher

Method or Message - A defined capability of a class - UploadGrades()

Attribute - A defined data item in a class - Department

Object or Instance – Mary Smith

Constructor – Special method that is invoked when an object is created.

2. JavaScript Language

Classes and Objects

In JavaScript is possible to declare an object without a previous declaration of class. This was the way of working before ES6

```
const myObject = {
  city: "Madrid",
  greet() {
    console.log(`Greetings from ${this.city}`);
  },
};

myObject.greet(); // Greetings from Madrid
```

2. JavaScript Language

Classes and Objects

Now, it is possible and convenient, to work with classes as you do in Java or Python

```
class Person {
  name;

  constructor(name) {
    this.name = name;
  }

  introduceSelf() {
    console.log(`Hi! I'm ${this.name}`);
  }
}
```

2. JavaScript Language

Classes and Objects

JavaScript supports inheritance and encapsulation.

Private data properties and methods must be declared in the class declaration, and their names start with #

```
class Professor extends Person {
  teaches;

  constructor(name, teaches) {
    super(name);
    this.teaches = teaches;
  }

  introduceSelf() {
    console.log(
      `My name is ${this.name}, and I will be your ${this.teaches} professor.`,
    );
  }

  grade(paper) {
    const grade = Math.floor(Math.random() * (5 - 1) + 1);
    console.log(grade);
  }
}
```

Parent class constructor

2. JavaScript Language

JSON (JavaScript Object Notation)

JSON is a standard format based on JavaScript object syntax used for representing structured data. It is quite useful for inter process communication in web applications (for instance, transferring information from server to client for display on a webpage, or the opposite).

In JSON, you can use the same fundamental data types as you would in a regular JavaScript object, such as strings, numbers, arrays, booleans, and other object literals.

Strings and property names are enclosed in double quotes while in JavaScript code, object properties can be unquoted. Single quotation marks are only acceptable when enclosing the entire JSON string.

2. JavaScript Language

```
1   const myJSONObject = '{"Name": "Jhon", "age": 42}';
2   console.log('JSON object: ', myJSONObject);
3   console.log('myJSONObject.age: ', myJSONObject.age); //undefined
4
5   //convert string JSON to JS object
6   const myObject = JSON.parse(myJSONObject);
7   console.log('JS object: ', myObject);
8   console.log('myObject.age: ', myObject.age); //42
9
10  myObject.age++;
11  //convert object to string JSON
12  const myNewJSONObject = JSON.stringify(myObject);
13  console.log('new JSON object: ', myNewJSONObject);
14
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS        bash

$node test/jsonexample.js
JSON object:  {"Name": "Jhon", "age": 42}
myJSONObject.age:  undefined
JS object:  { Name: 'Jhon', age: 42 }
myObject.age:  42
new JSON object:  {"Name":"Jhon","age":43}
```

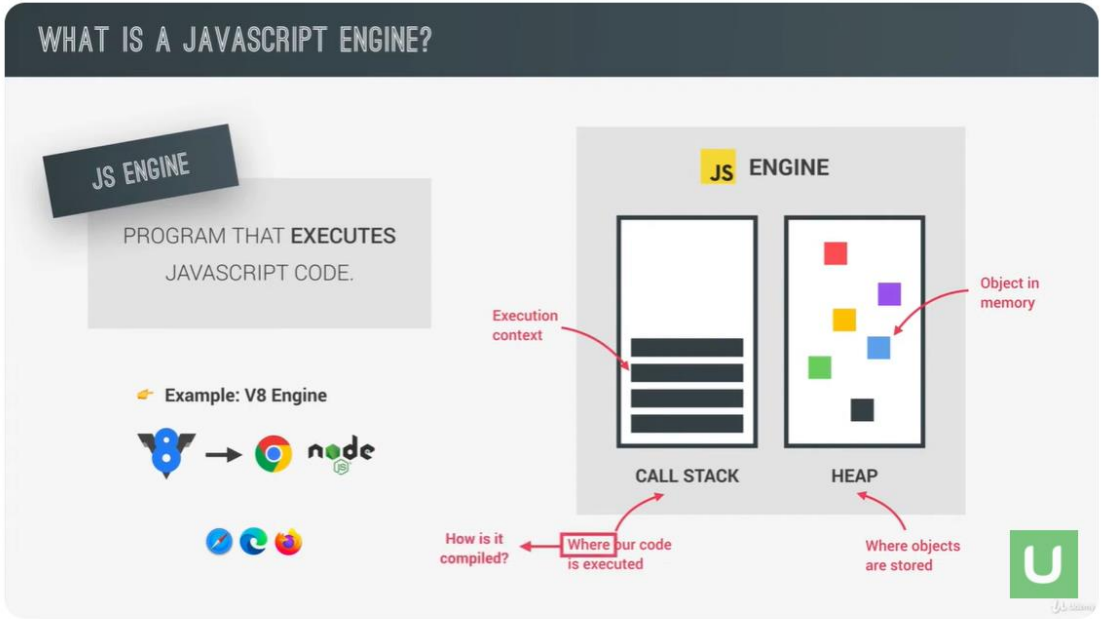
2. JavaScript Language

Functional programming

As functions are first-class orders, functional programming is a natural thing in JavaScript. Arrays are well-suited for this purpose. With methods such as forEach(), filter(), map() and reduce() the syntax becomes stylish.

```
1  myarray = [-1,5,3,2,0.98]
2  // Sum of the square of each element in the traditional way
3  let sumsq = 0;
4  for (let i=0; i < myarray.length; i ++)
5      sumsq = sumsq + myarray[i]**2;
6  console.log(sumsq);
7  // Functional way of doing things
8  sumsq = myarray.map(x => x*x).reduce((sum,element) => sum+element);
9  console.log(sumsq);
```

The JavaScript engine



4 The JavaScript Engine and Runtime

https://www.youtube.com/watch?v=cNhLMDkDuao

3. JavaScript + HTML

There are three ways to embed JavaScript code in any HTML document

- Inside an HTML tag

```
<p onclick = "alert('You clicked!!')"> Click </p>
```

- Write a code snippet inside the HTML page (as we do in this course, to show together both HTML and JavaScript)

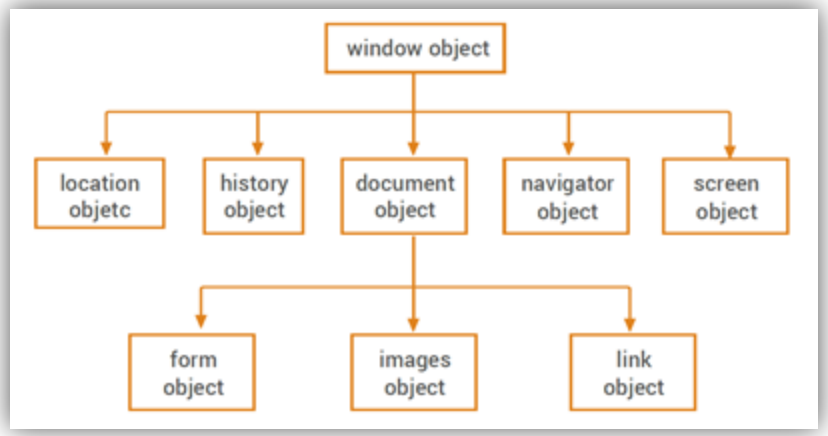
```
<head>
  <script type = "text/javascript">
    document.write("Hello World!!!")
</script>
</head>
```

- Import an external file (use this method as your first choice)

```
<head>
  <script type = "text/javascript" src = "archivo.js"></script>
</head>
```

3. JavaScript + HTML

JavaScript interacts with the Browser (or the Node Server), using a hierarchy of classes, called the BOM (Browser Object Model)

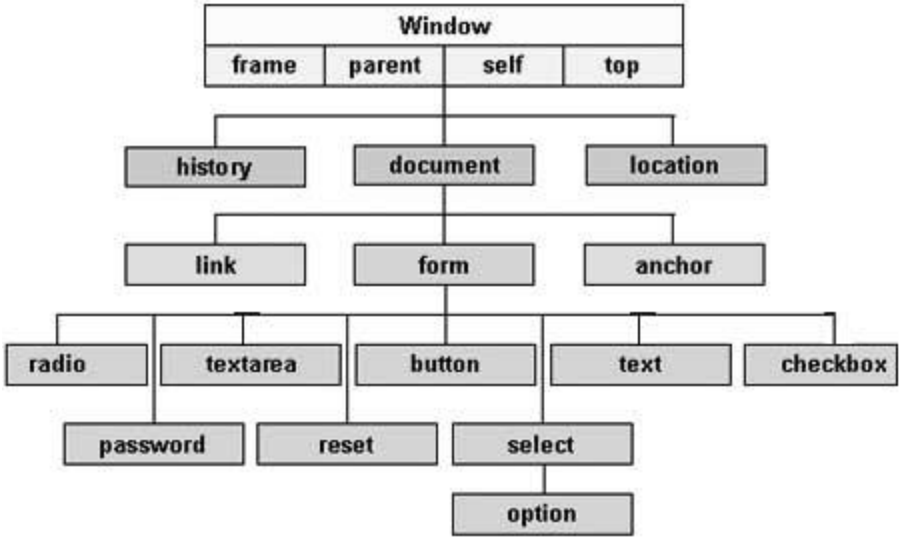


The `window` object is the top-level object, stores the properties of the window. When working with frames, there is a window object for each frame. The document object holds the properties of the current document, such as its background colour, links, images, etc.

3. JavaScript + HTML

The DOM

It is a programming interface for HTML documents. It defines the objects and properties of HTML elements and the methods for accessing them. DOM objects allow you to inspect and modify HTML elements.

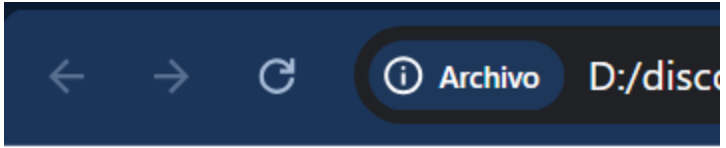


3. JavaScript + HTML

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>DOM Example</title>
</head>
<script type="text/javascript">
    function showInfo() {
      var element = document.getElementById("opener");
      var buffer = element.id + " tag is " + element.tagName;
      alert(buffer);
      element = document.getElementById("actionItem");
      buffer = element.id + " tag is " + element.tagName;
      buffer += ", type is "+element.type;
      alert(buffer);
    }
    </script>
  </head>
  <body>
    <p id="opener">The id attribute is very helpful.</p>
    <p id="closer">This is the closing paragraph.</p>
    <form>
    <button id="actionItem" type="button" onclick="showInfo()">Show Info</button>
    </form>
  </body></html>
```

3. JavaScript + HTML

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>DOM Example</title>
</head>
<script type="text/javascript">
    function showInfo() {
      var element = document.getElementById("opener");
      var buffer = element.id + " tag is " + element.tagName;
      alert(buffer);
      element = document.getElementById("actionItem");
      buffer = element.id + " tag is " + element.tagName;
      buffer += ", type is "+element.type;
      alert(buffer);
    }
    </script>
  </head>
  <body>
    <p id="opener">The id attribute is very helpful.</p>
    <p id="closer">This is the closing paragraph.</p>
    <form>
    <button id="actionItem" type="button" onclick="showInfo()">Show Info</button>
    </form>
  </body></html>
```



The id attribute is very helpful.

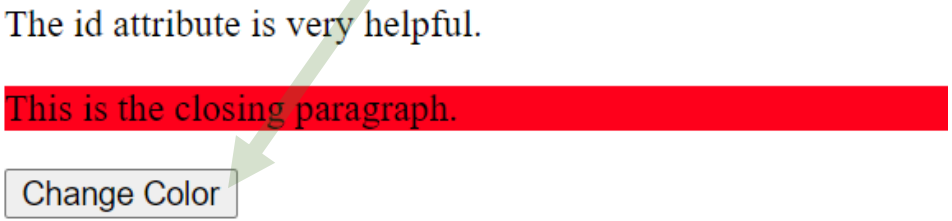
This is the closing paragraph.

Show Info

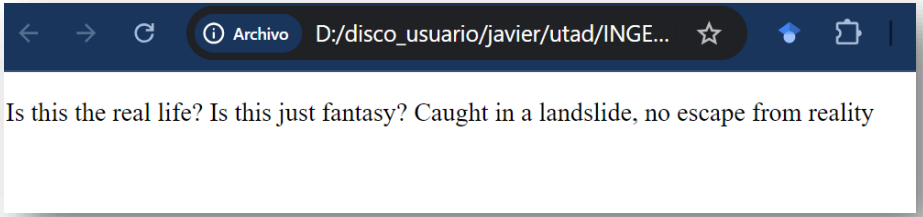
3. JavaScript + HTML

You can add, modify or remove HTML tags or CSS properties using the `document` object

```
    function changeTextColor(){
            document.querySelector('#closer').style.background = 'red';
            return false;
    }
    </script>
</head>
<body>
    <p id="opener">The id attribute is very helpful.</p>
    <p id="closer">This is the closing paragraph.</p>
    <form>
    <button id="actionItem" type="button" onclick="changeTextColor()">Change Color</button>
    </form>
</body></html>
```



3. JavaScript + HTML



We want to modify the HTML text identified as `#mytext`, to properly display the opening three verses of “Bohemian Rhapsody”. We are going to add a button, as in the previous example, to trigger the change.

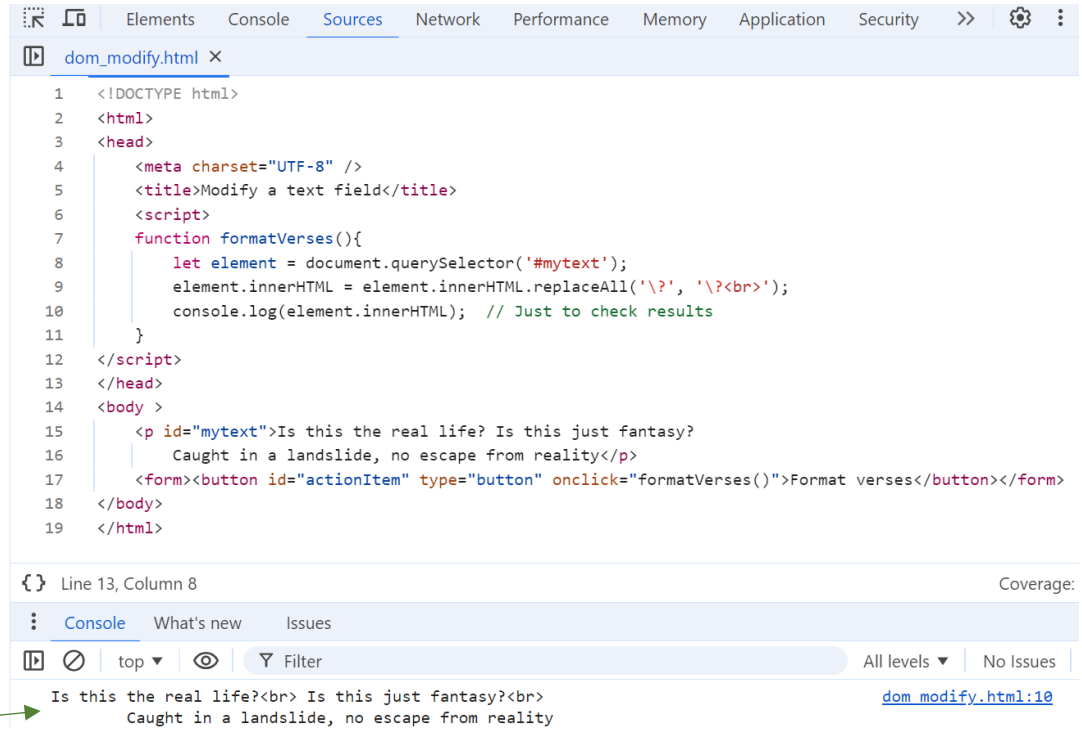
The text is a paragraph. To get the DOM element there are two ways. The older style is using the `getElementById()` method. There are methods for tags and classes as well: `getElementsByClassName()`, `getElementsByTagName()`. Since the arrival of ES6 is better practice using the generic `querySelector()` with the CSS-style tag as input selector. For instance, `#mytex` for and Id.

The raw HTML of any DOM element can be retrieved using the property `innerHTML` of the object

3. JavaScript + HTML

Is this the real life?
Is this just fantasy?
Caught in a landslide, no escape from reality

Format verses



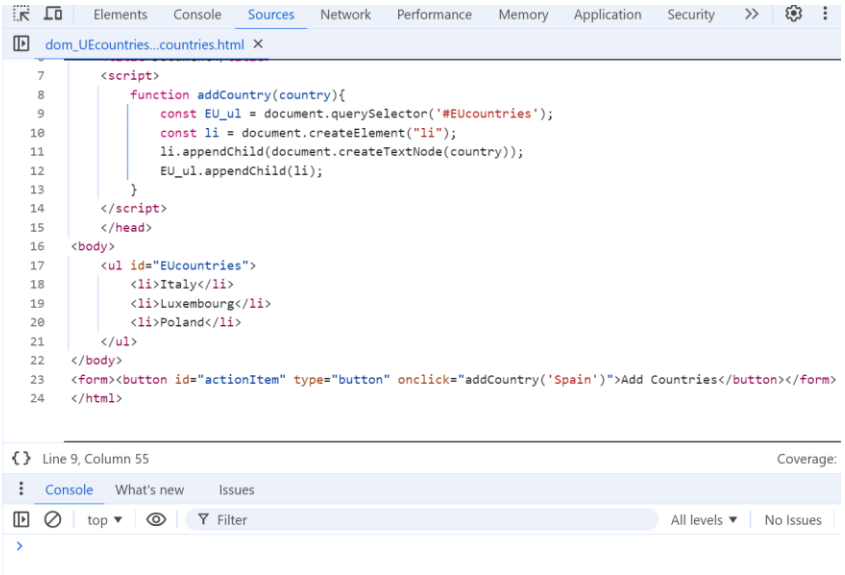
HTML of mytext after calling formatVerses()

3. JavaScript + HTML

We can add as many elements to the DOM as we need. In this example, the HTML page creates a list three EU countries. We want to add a new entry to the list on clicking the button. What happens if you keep on clicking the button? How could you solve this problem?

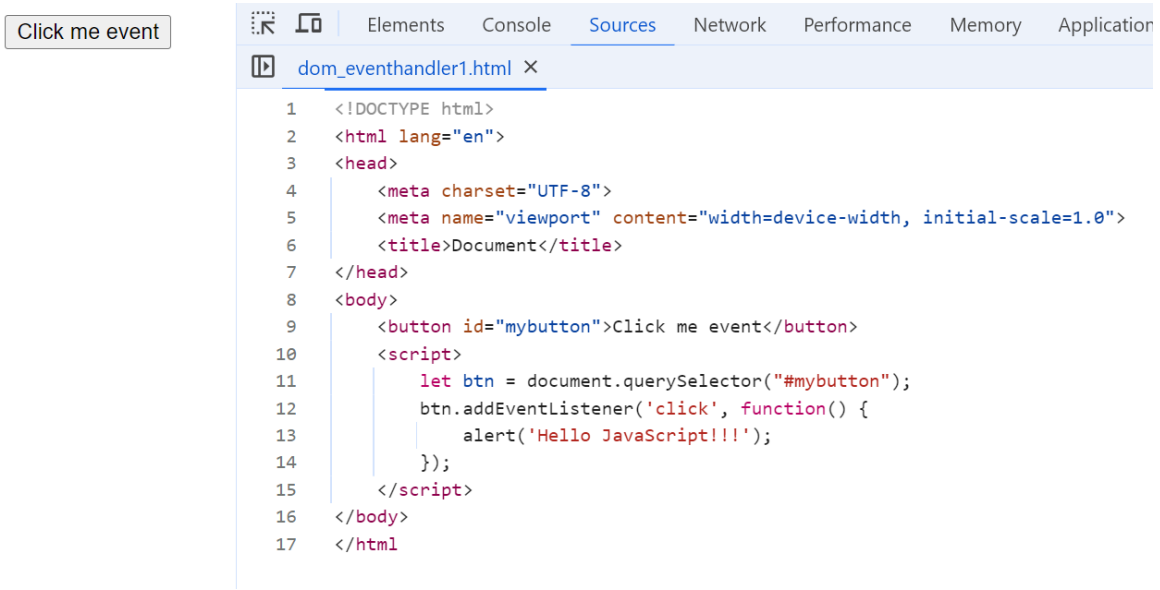
- Italy
- Luxembourg
- Poland
- Spain

Add Countries



3. JavaScript + HTML

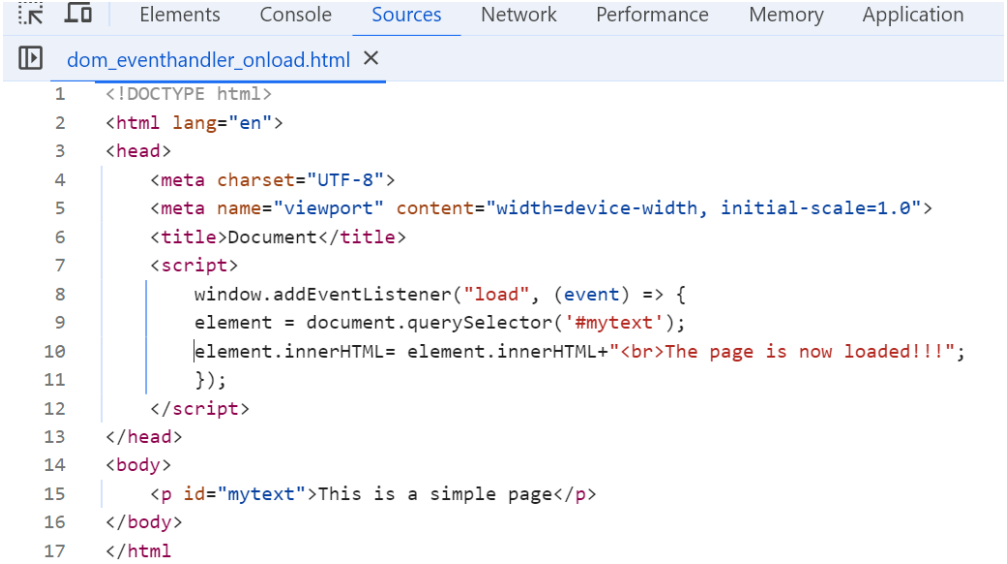
Event handlers may be managed using JavaScript in a more elegant and flexible way than as HTML tag modifiers.



3. JavaScript + HTML

Events may be not user-generated. For instance, we can perform some task when the page is finally loaded. A common technique in modern Java is adding all event handlers within a general function that is called when the page is loaded.

This is a simple page
The page is now loaded!!!



3. JavaScript + HTML

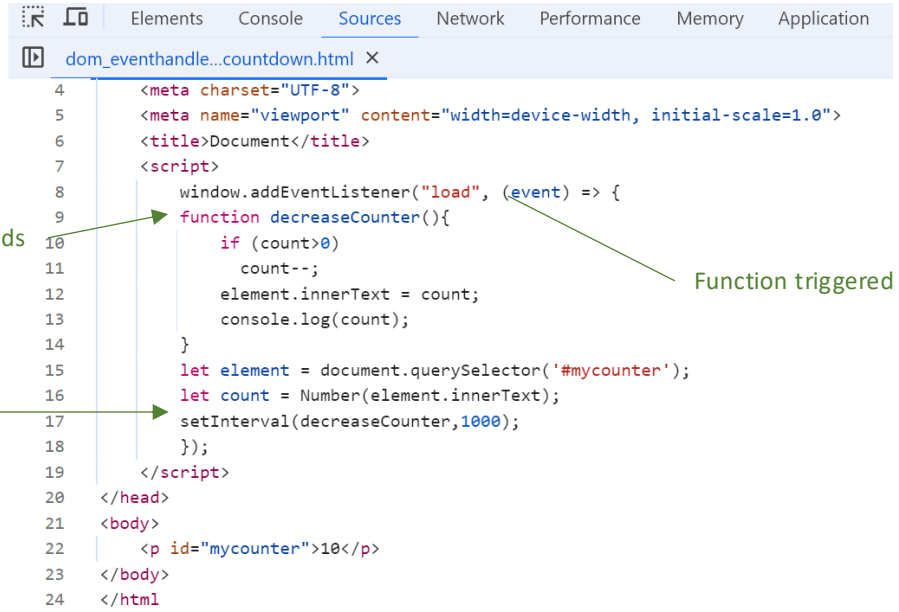
Another common application of events is adding some kind of periodic refresh within our page.

7

Nested function to count backwards

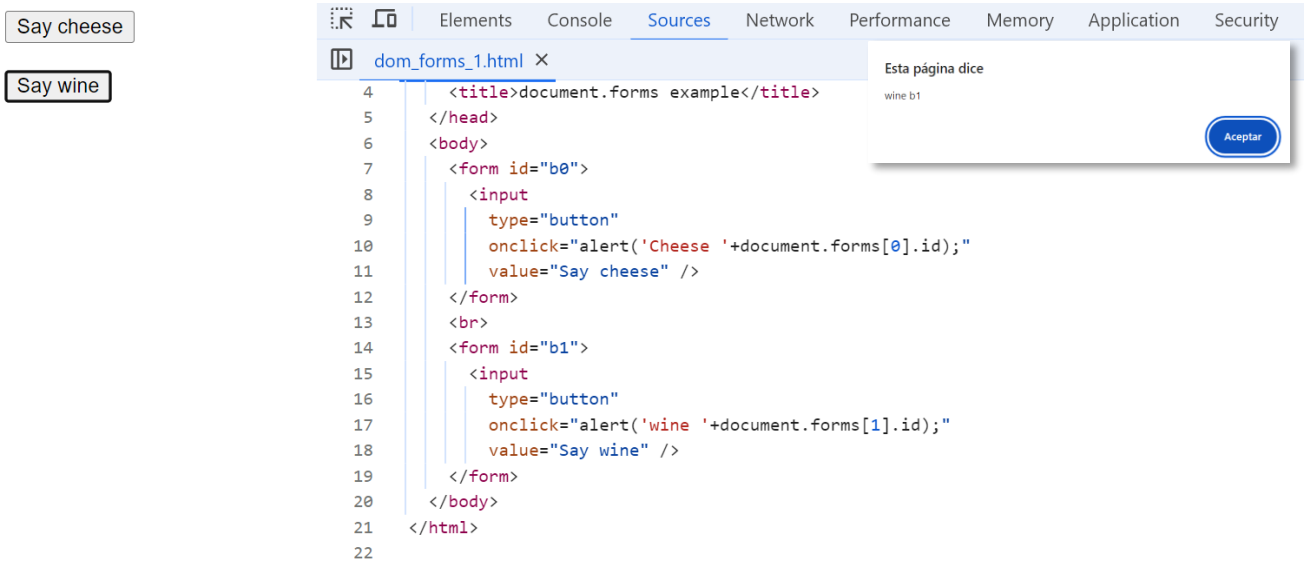
Set the execution rate of the nested function

Function triggered by onload event



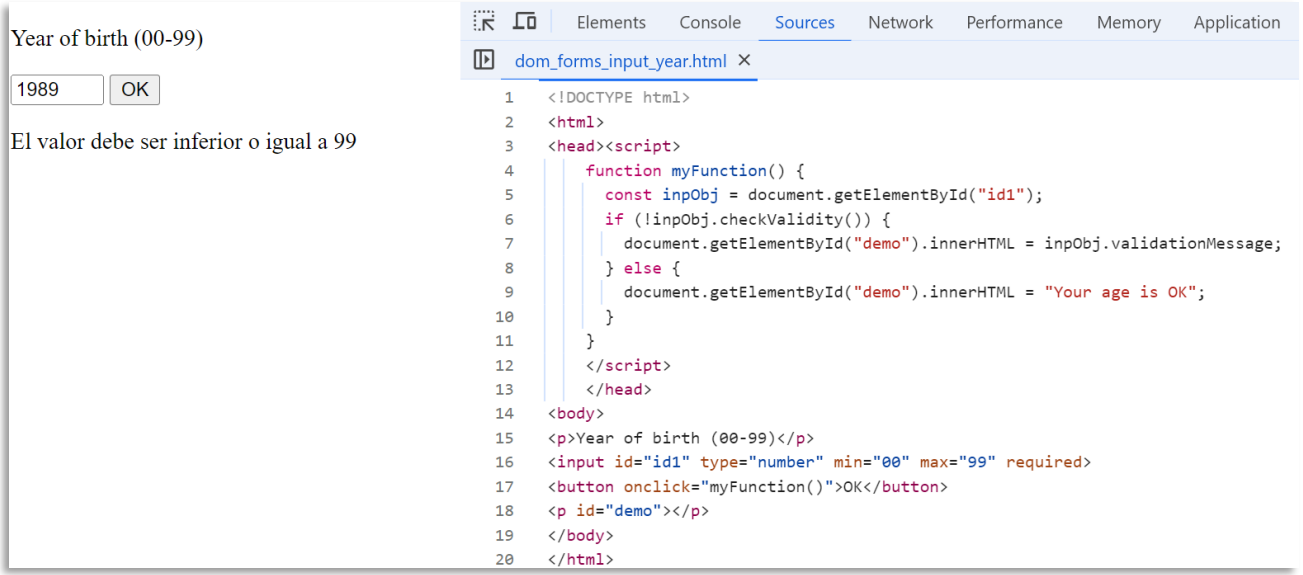
3. JavaScript + HTML

Forms are the property `document.forms` of the DOM. Each form is an array of elements. On loading the web page, the browser creates an array called 'forms' which contains the reference to all the forms on the page: `document.forms[i].elements[j]`



3. JavaScript + HTML

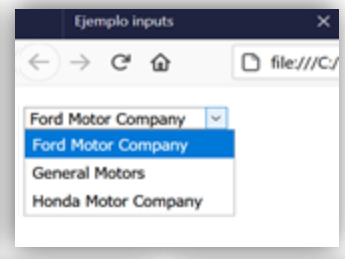
Forms are used to fill information, so you can send the data to a server to be processed. There are different kinds of input fields, such as text, radio button, checkbox… You can check that data are valid before posting it to the remote edge.



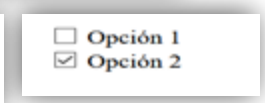
3. JavaScript + HTML

Input types		
Text	tel	time
password	number	email
submit	range	radio
reset	date	checkbox
search	week	file
url	month	image

```
<select id = "company">
    <option value="ford">
        Ford Motor Company
    </option>
    <option value="generalm">
        General Motors
    </option>
    <option value="honda">
        Honda Motor Company
    </option>
</select>
```



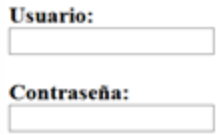

```
<input type="checkbox" id = "check1" value="opc1">
    Opción 1 <br>
<input type="checkbox" id = "check2" value="opc2">
    Opción 2 <br>
```




```
<input type="search" placeholder="Escriba su búsqueda">
```

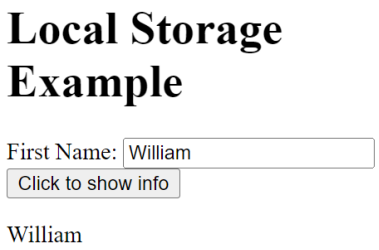

```
<input type="date" id="fecha"> <br>
```


```
<form>
    <b>Usuario:</b><br>
    <input type="text" id ="usuario"><br/><br>
    <b>Contraseña:</b><br>
    <input type="password" id ="passwd">
</form>
```

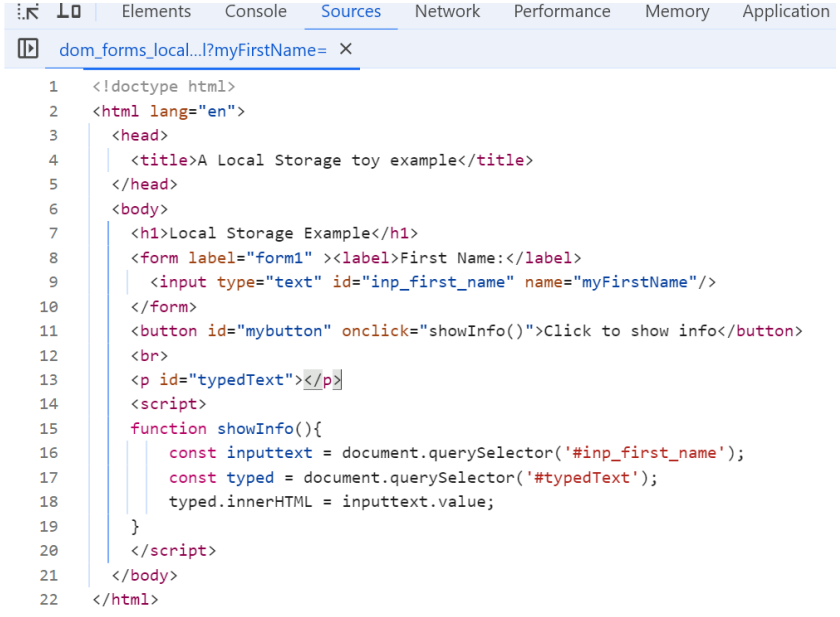


3. JavaScript + HTML

It is quite annoying when you fill a complex form with many fields and you lose the information if by accident you close the tab or there is a connection failure. The browser API provides a mechanism to store data as pairs key-value, so you can retrieve it if necessary.



This code just displays what we typed inside the input field. If we reload the page we lose the information



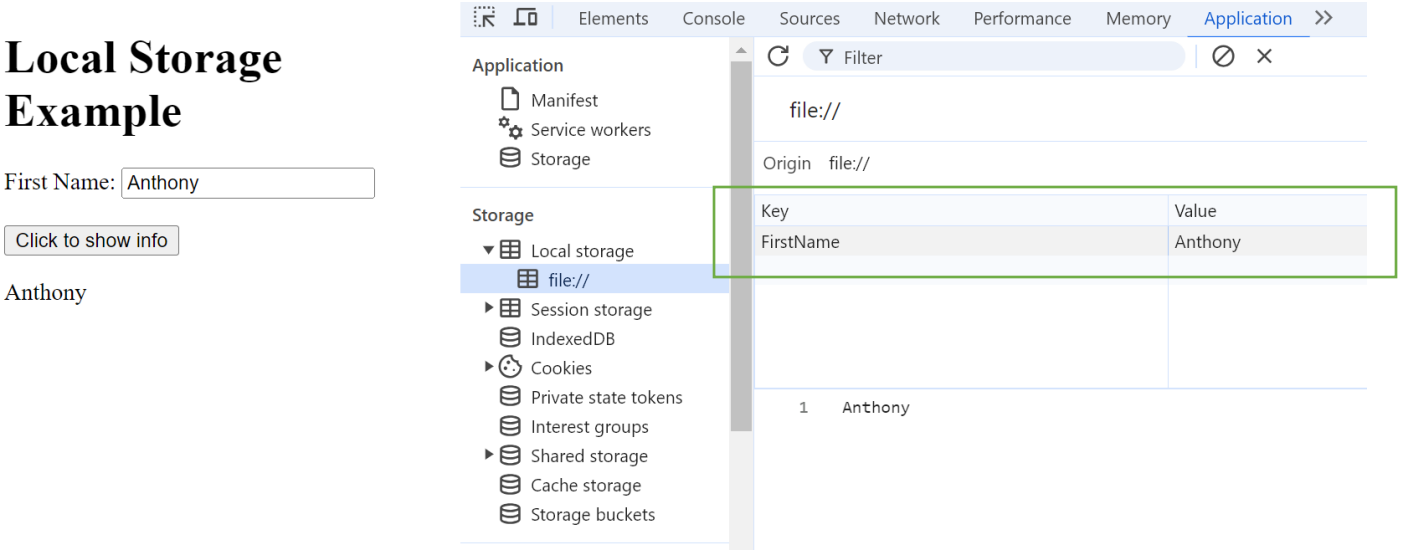
3. JavaScript + HTML

We now store the value the user has typed, with a new event listener imported from an external final. Pay attention to the way we add listeners from JavaScript. A wrapper function is called when the page is fully loaded, to be sure that all elements are available. Otherwise the script may find null values when searching for a DOM element.

```
1   // We add the event listener after the page is loaded
2   window.addEventListener('load', function() {
3       const btn = document.querySelector('#mybutton');
4       btn.addEventListener('click',showandStoreInfo);
5       function showandStoreInfo(){
6           const typed = document.querySelector('#typedText');
7           const inputtext = document.querySelector('#inp_first_name');
8           typed.innerHTML = inputtext.value;
9           localStorage.setItem("FirstName",inputtext.value);
10          return true;
11      }
12  })
```

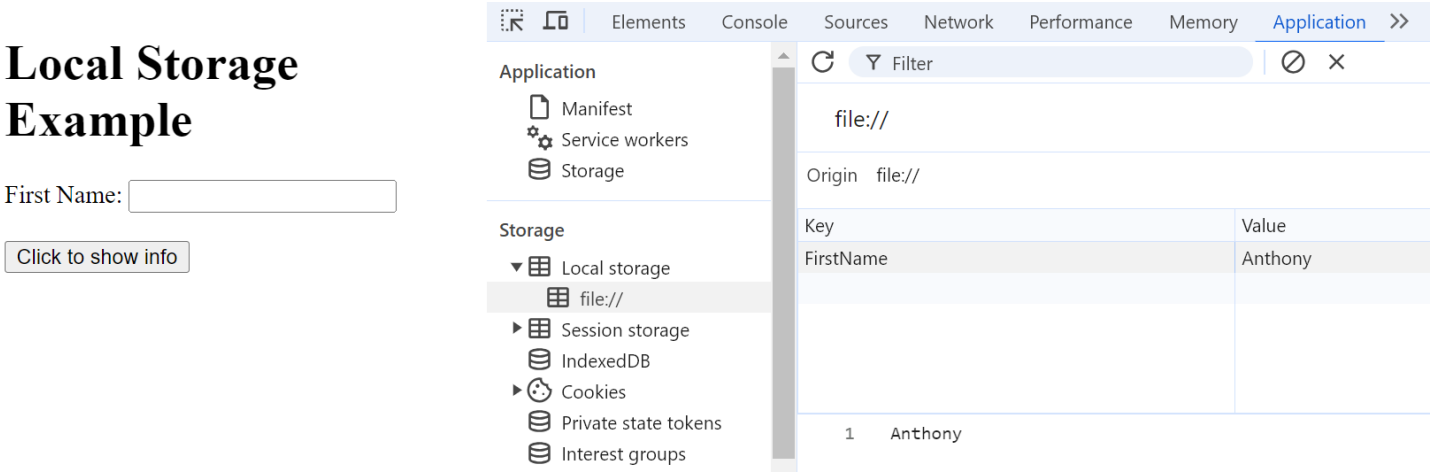
3. JavaScript + HTML

When the user clicks the button, information is stored by the browser. Opening the Application tab of the Inspect option you may find the value and even manually change it.



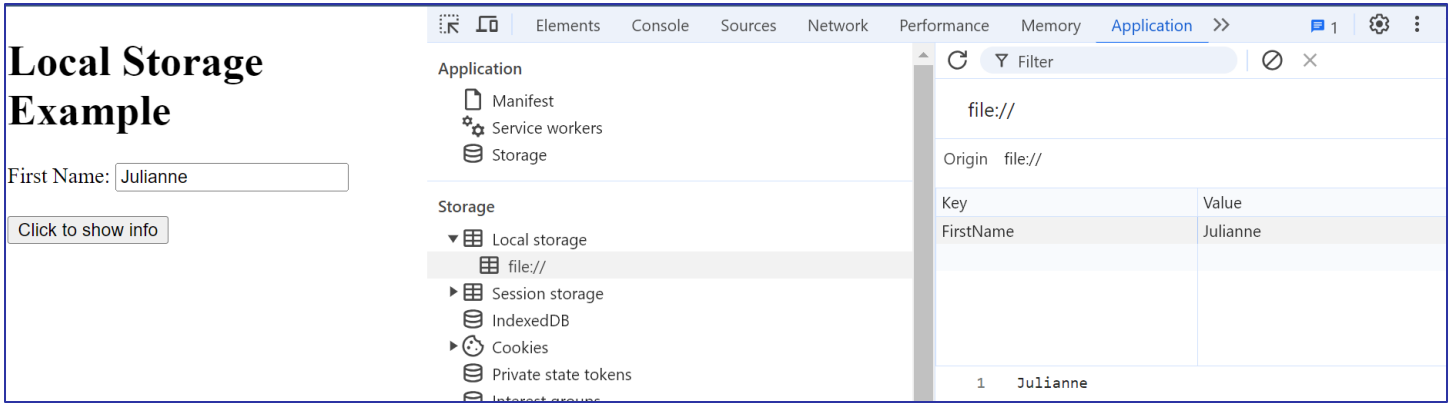
3. JavaScript + HTML

If the user reloads the page the input box is cleared, but the local storage value persists. Is up to you to check if the variable exists locally to retrieve the value and fill the input box on reload.



3. JavaScript + HTML

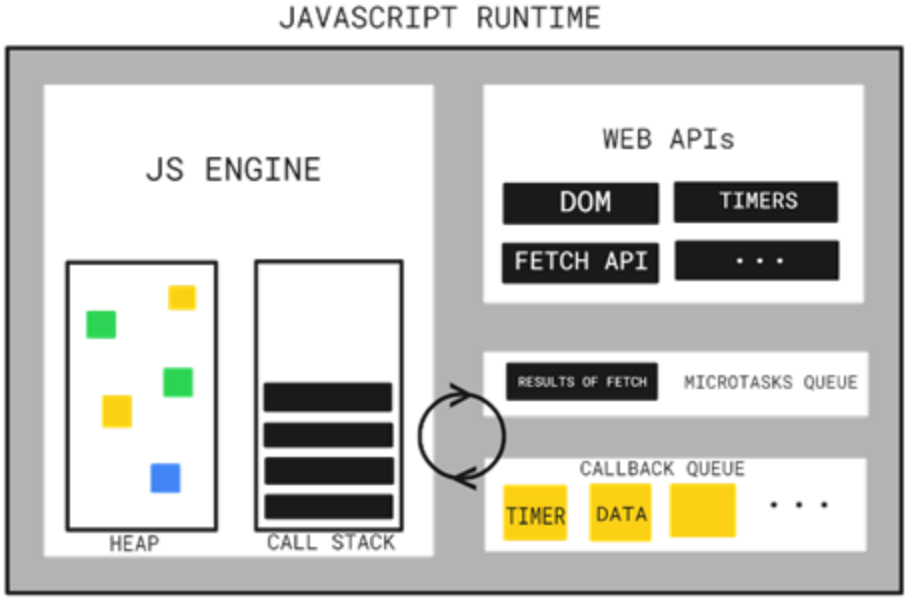
If the user reloads the page the input box is cleared, but the local storage value persists. Is up to you to check if the variable exists locally to retrieve the value and fill the input box on reload.




```
const valueFN = localStorage.getItem("FirstName");
if (valueFN != null)
    inputtext.value = valueFN;
```

4. Asynchronous programming

JavaScript runs on a single-thread, and cannot invoke low-level services. How is it possible to detect user-events, communicate through the network or manage timers? The answer is simple, those tasks do not run inside the JavaScript engine, they are provided by the browser (or Node) through a web API.



4. Asynchronous programming

So, when we invoke a JavaScript instruction that involves a web API call, that task is run outside the JavaScript engine. The JavaScript thread cannot get blocked until the task is finished, it will receoeive, instead, an asynchronous result of the operation when it is completed. During that lapse of time, the variables involved in that operation are undefined.

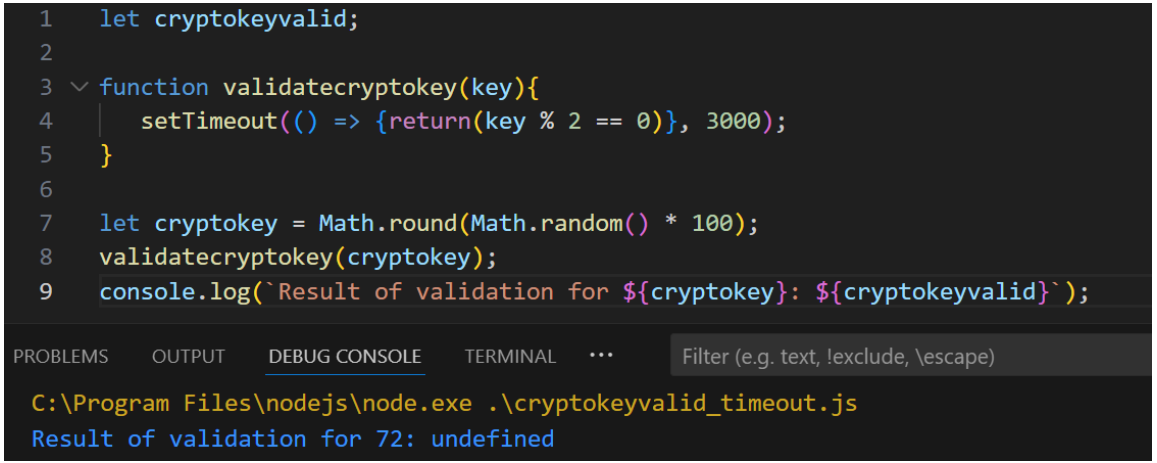
Let's start with a toy example. We have a random cryptographic key generator. After the key is generated it must be checked is valid. For our purposes, we will asume that only even numbers are valid. The synchronous code of this program is fairly simple.

```
1  let cryptokeyvalid;
2  function validatecryptokey(key){
3      return(key % 2 ==0);
4  }
5
6  cryptokey = Math.round(Math.random()*100);
7  cryptokeyvalid = validatecryptokey(cryptokey);
8  console.log(`Result of validation for ${cryptokey}: ${cryptokeyvalid}`);
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  ···  Filter (e.g. text, !exclude, \escape)
C:\Program Files\nodejs\node.exe .\cryptokeyvalid.js
Result of validation for 54: true
```

4. Asynchronous programming

Now, imagine that the validation process is performed by a remote server and that it takes 3 seconds to complete a full validation. To simulate it, we include a `setTimeout()` call, that is asynchronous. When we run the program, there is an issue. When `console.log()` is called, the asynchronous task hasn't finished yet and the variable `cryptokeyvalid` is still undefined. It will take 3000 milliseconds, until `validatecryptokey` ends.



4. Asynchronous programming

To fis this issue, we use a **callback**, a function that is passed to validatecryptokey() that includes the final call to `console.log()`. This syntax enforces the execution of the callback function after the times expires. The `console.log()` takes 3 seconds to be displayed.

```
1   let cryptokeyvalid;
2   function perfValidation(key){
3       cryptokeyvalid = key % 2 == 0;
4       console.log(`Result of validation for ${key}: ${cryptokeyvalid}`);
5   }
6
7   function validatecryptokey(callbackf, key){
8      setTimeout(() => callbackf(key), 3000);
9   }
10
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   ...   Filter (e.g. text, !exclude, \escape)

C:\Program Files\nodejs\node.exe .\cryptokeyvalid_timeout_callbackOK.js
Result of validation for 74: true
```

4. Asynchronous programming

Promises are objects to handle asynchronous results. A promise may be resolved when the result is OK or rejected when there is an error. The code on successful completion is written after the `then` clause, on error whe add a `catch` statement (not included in this toy example).

```
let cryptokeyvalid;
function perfValidation(key) {
    return new Promise((resolve, reject) => {
        cryptokeyvalid = key % 2 === 0;
        resolve(cryptokeyvalid);
    });
}
function validatecryptokey(key) {
    return new Promise((resolve) => {
        setTimeout(() => {
            resolve(perfValidation(key));
        }, 3000);
    });
}
let cryptokey = Math.round(Math.random() * 100);
validatecryptokey(cryptokey)
    .then((isValid) => {
        console.log(`Result of validation for ${cryptokey}: ${isValid}`);
    })
    .catch((error) => {
        console.error(`Validation failed: ${error}`);
    });

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR  Filter (e.g.
C:\Program Files\nodejs\node.exe .\cryptokeyvalid_promise.js
Result of validation for 48: true
```

4. Asynchronous programming

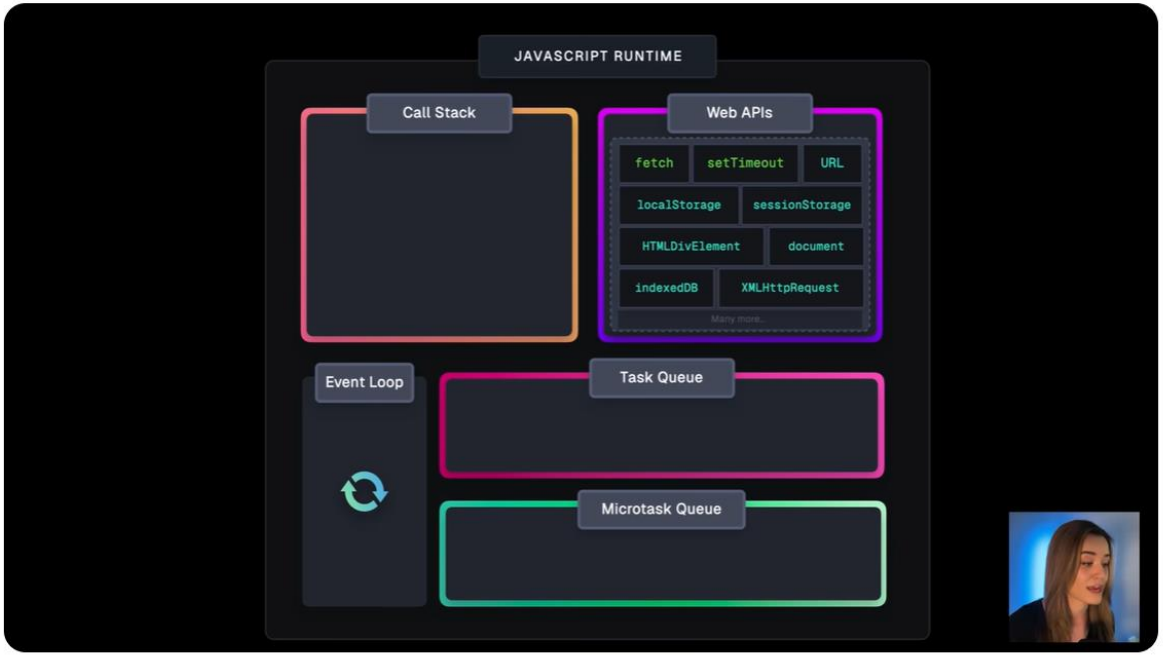
Finally, there is a third method to deal with asynchronous events. The `async … await` construction provides a syntax quite similar to synchronous programming to deal with promises.

```
let cryptokeyvalid;
function perfValidation(key) {
    return new Promise((resolve, reject) => {
        cryptokeyvalid = key % 2 === 0;
        resolve(cryptokeyvalid);
    });
}
function validatecryptokey(key) {
    return new Promise((resolve) => {
        setTimeout(() => {
            resolve(perfValidation(key));
        }, 3000);
    });
}
async function main() {
    let cryptokey = Math.round(Math.random() * 100);

    try {
        let isValid = await validatecryptokey(cryptokey);
        console.log(`Result of validation for ${cryptokey}: ${isValid}`);
    } catch (error) {
        console.error(`Validation failed: ${error}`);
    }
}
main();
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR  Filte
C:\Program Files\nodejs\node.exe .\cryptokeyvalid_asyncawait.js
Result of validation for 27: false
```

4. Asynchronous programming



JavaScript Visualized - Event Loop, Web APIs, (Micro)task Queue

https://www.youtube.com/watch?v=eiC58R16hb8

5.HTTP requests

- HTTP (HyperText Transfer Protocol) requests are the foundation of communication between clients (like web browsers) and servers on the web. They are used to send and retrieve data, enabling the client to interact with web resources.
- Key HTTP Request Methods:
  - **GET**: Retrieves data from the server. This is the most common method used to fetch web pages and API data.
  - **POST**: Sends data to the server, often used when submitting forms or uploading files.
  - **PUT**: Updates existing resources on the server.
  - **DELETE**: Removes resources from the server.
  - **PATCH**: Partially updates an existing resource.

5.HTTP requests

- **Status Codes**: HTTP responses return status codes to indicate the outcome (e.g., 200 OK, 404 Not Found, 500 Internal Server Error).
- **Headers**: Both requests and responses contain headers that provide metadata (e.g., content type, authorization tokens, etc.).
- **Request Body**: Only present in certain requests (like POST or PUT), used to send data to the server.

5.HTTP requests

- **fetch**() is a modern, built-in JavaScript method used to make HTTP requests to servers. It provides a simple way to retrieve data asynchronously, returning a Promise that resolves to the Response object. This allows you to handle both success and error scenarios effectively.
- The fetch() method **always returns a Promise**, making it easy to handle asynchronous requests.
- The promise **resolves to a Response object**, which provides several methods like **json()**, text(), blob(), etc., to read the response body.
- fetch() will only reject the promise for network errors, not for HTTP status errors like 404 or 500.

5.HTTP requests

- Fetch example:

```
fetch("https://jsonplaceholder.typicode.com/users")
.then(response => response.json())
.then(json => console.log(json))
```