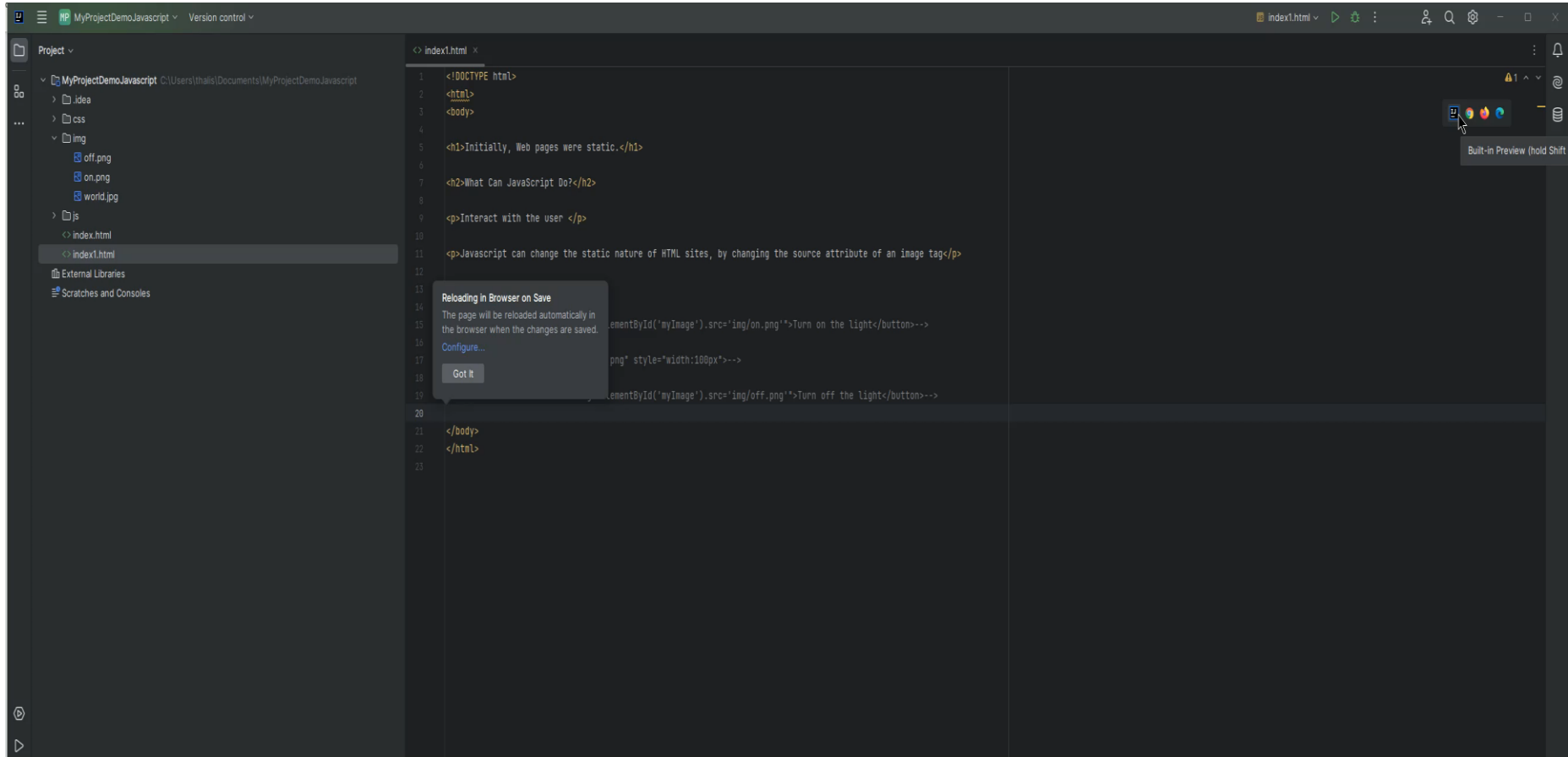


JavaScript

An introduction to JavaScript Programming

Why we need JavaScript?



Why we need JavaScript?

- JavaScript is a high-level language like Python and Java
- JavaScript, HTML and CSS are the primary languages used to build a website's front-end applications
- All browsers include a JavaScript engine to execute code
- JavaScript can even be used for server-side operations with node.js

Main attributes and properties

- JavaScript is a dynamically typed language
 - Is a weakly typed language and prototype-based (objects)
 - It is a multi-paradigm language, supporting functional and event-driven behavior
 - It has an application programming interface (API), for handling text, arrays and HTML Document Object Model (DOM)
 - JavaScript does not include input/output (I/O) module for networking, storage or any graphics interface
-

How to Add JavaScript into an HTML Document

Internal JavaScript

placed in head area of a document

```
<script>  
    // code block  
</script>
```

External JavaScript

async: scripts with the async attribute are executed asynchronously

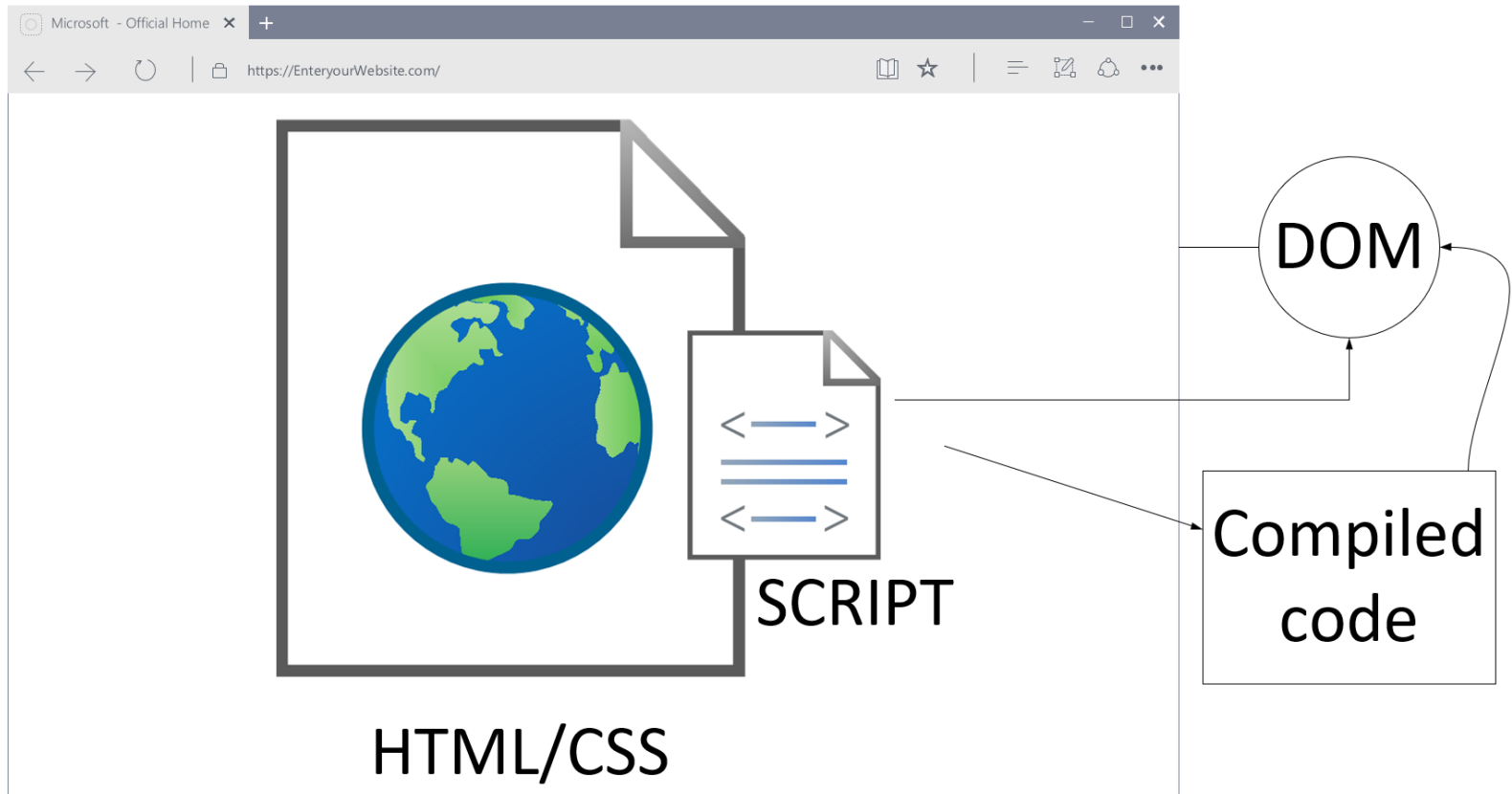
defer: the script is downloaded in parallel while parsing the page

```
<script src = "scriptfile.js"  
    async>  
</script>
```

```
<button  
    onclick="createParagraph()  
    "Click!!</button>
```

Inline JavaScript

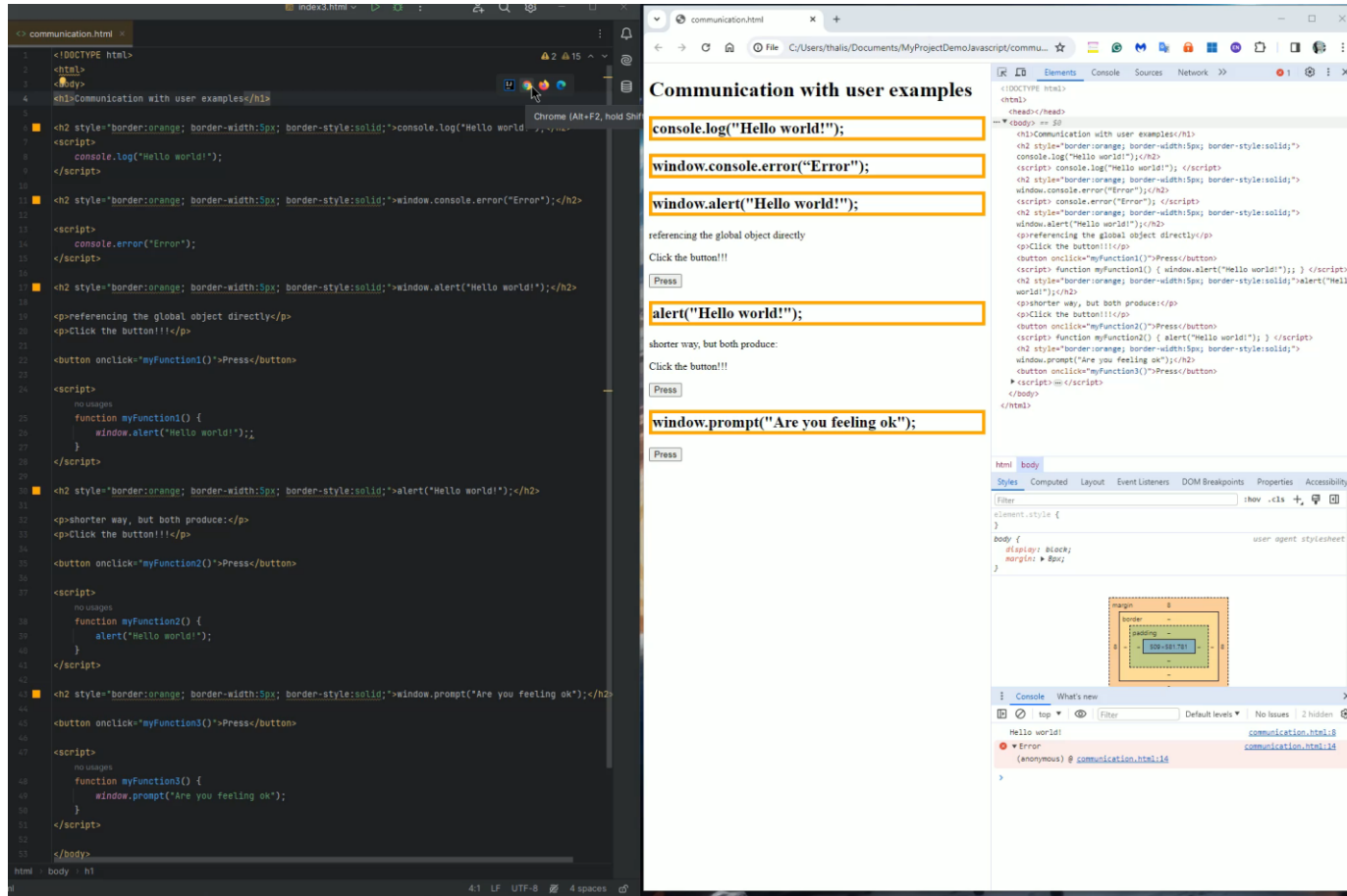
Just-in-time compiler (JIT)



JavaScript communication with user examples

- console.log("Hello world!");
- window.console.error("Error");
- window.alert("Hello world!");
//referencing the global object directly
- alert("Hello world!");
//shorthand way
- window.prompt("Are you feeling ok");

JavaScript communication with user examples



The image shows a code editor on the left and a web browser on the right. The code editor displays the following HTML and JavaScript code:

```
<!DOCTYPE html>
<html>
<body>
  <h1>Communication with user examples</h1>

  <div style="border:orange; border-width:5px; border-style:solid;">
    <script>
      console.log("Hello world!");
    </script>
  </div>

  <div style="border:orange; border-width:5px; border-style:solid;">
    <script>
      console.error("Error");
    </script>
  </div>

  <div style="border:orange; border-width:5px; border-style:solid;">
    <script>
      window.alert("Hello world!");
    </script>
  </div>

  <p>referencing the global object directly</p>
  <p>Click the button!!</p>
  <button onclick="myFunction1()">Press</button>

  <script>
    function myFunction1() {
      window.alert("Hello world!");
    }
  </script>

  <p>shorter way, but both produce:</p>
  <p>Click the button!!</p>
  <button onclick="myFunction2()">Press</button>

  <script>
    function myFunction2() {
      alert("Hello world!");
    }
  </script>

  <div style="border:orange; border-width:5px; border-style:solid;">
    <script>
      window.prompt("Are you feeling ok?");
    </script>
  </div>

  <p>shorter way, but both produce:</p>
  <p>Click the button!!</p>
  <button onclick="myFunction3()">Press</button>

  <script>
    function myFunction3() {
      window.prompt("Are you feeling ok?");
    }
  </script>
</body>
</html>
```

The web browser on the right shows the rendered page with the title "Communication with user examples". It displays three orange-bordered boxes containing the following JavaScript code snippets:

```
console.log("Hello world!");
window.console.error("Error");
window.alert("Hello world!");
```

Below these boxes, there are three buttons labeled "Press". The first button is associated with the first code snippet, the second with the second, and the third with the third. The browser's console shows the following output:

```
Hello world!
Error
Are you feeling ok?
```

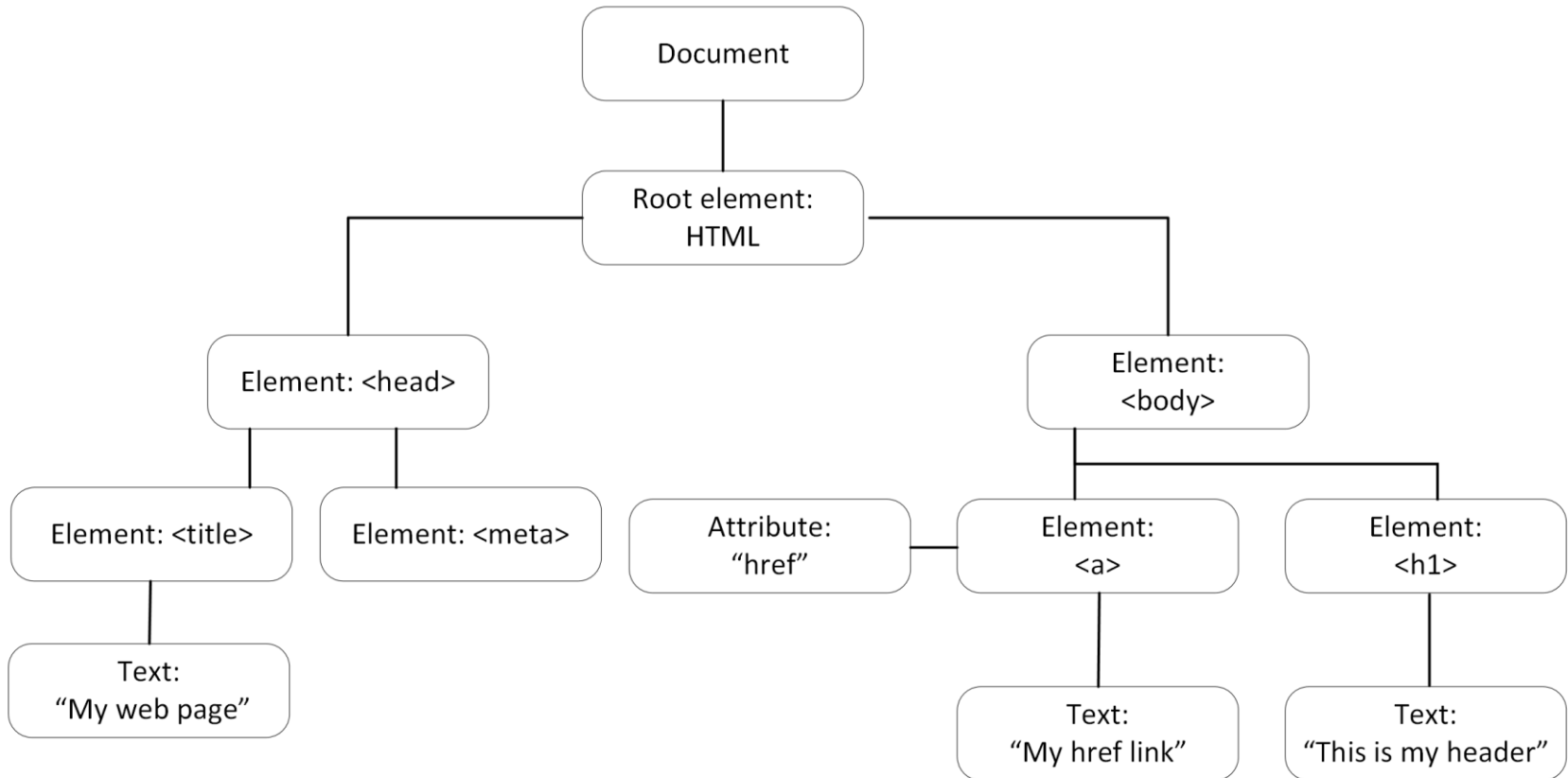

JavaScript engines

Every browser starts two processes for every web page

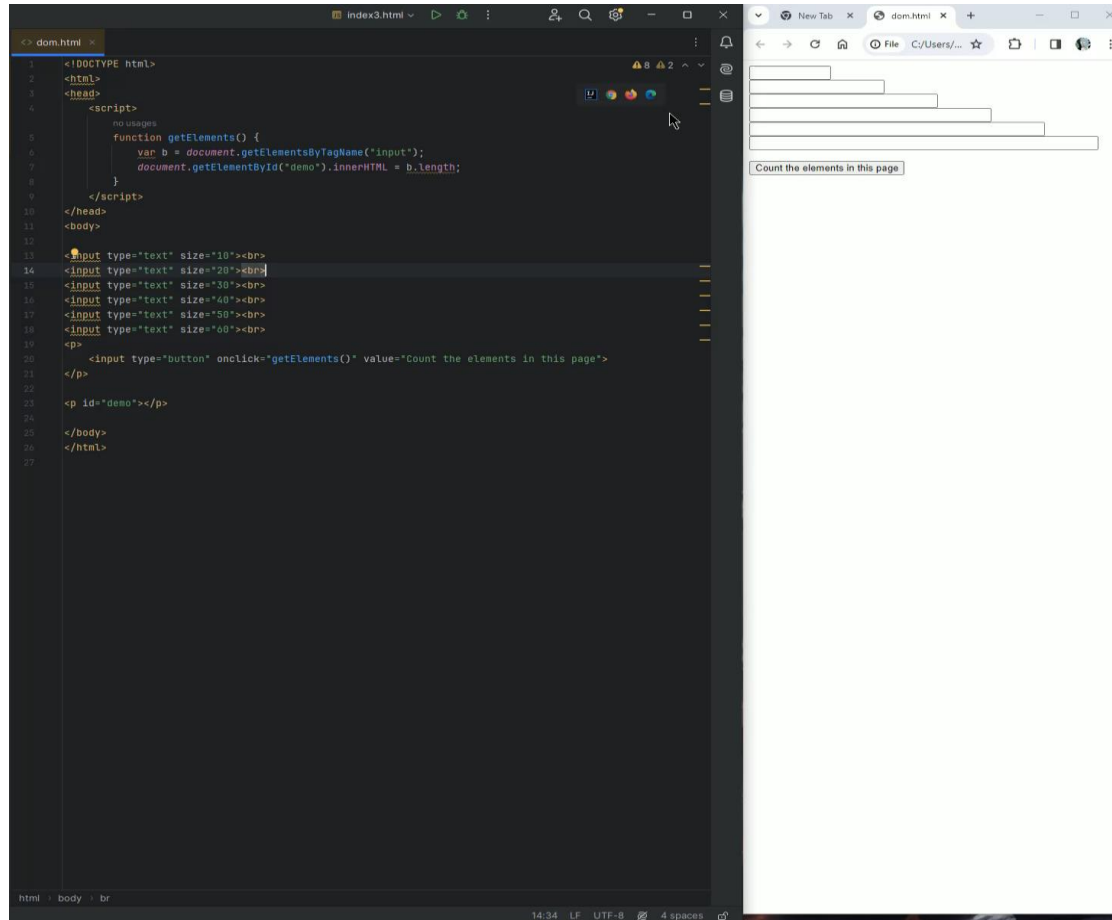
- **Rendering engine:** it is responsible to display the web page. Parses the HTML and CSS and displays the content on the screen
- **JavaScript engine:** This is where JavaScript code gets executed

JavaScript is interpreted, not a compiled language. Modern browsers use a technology called Just-In-Time (JIT) compilation that compiles the code to an executable bytecode.

Document Object Model (DOM) tree structure



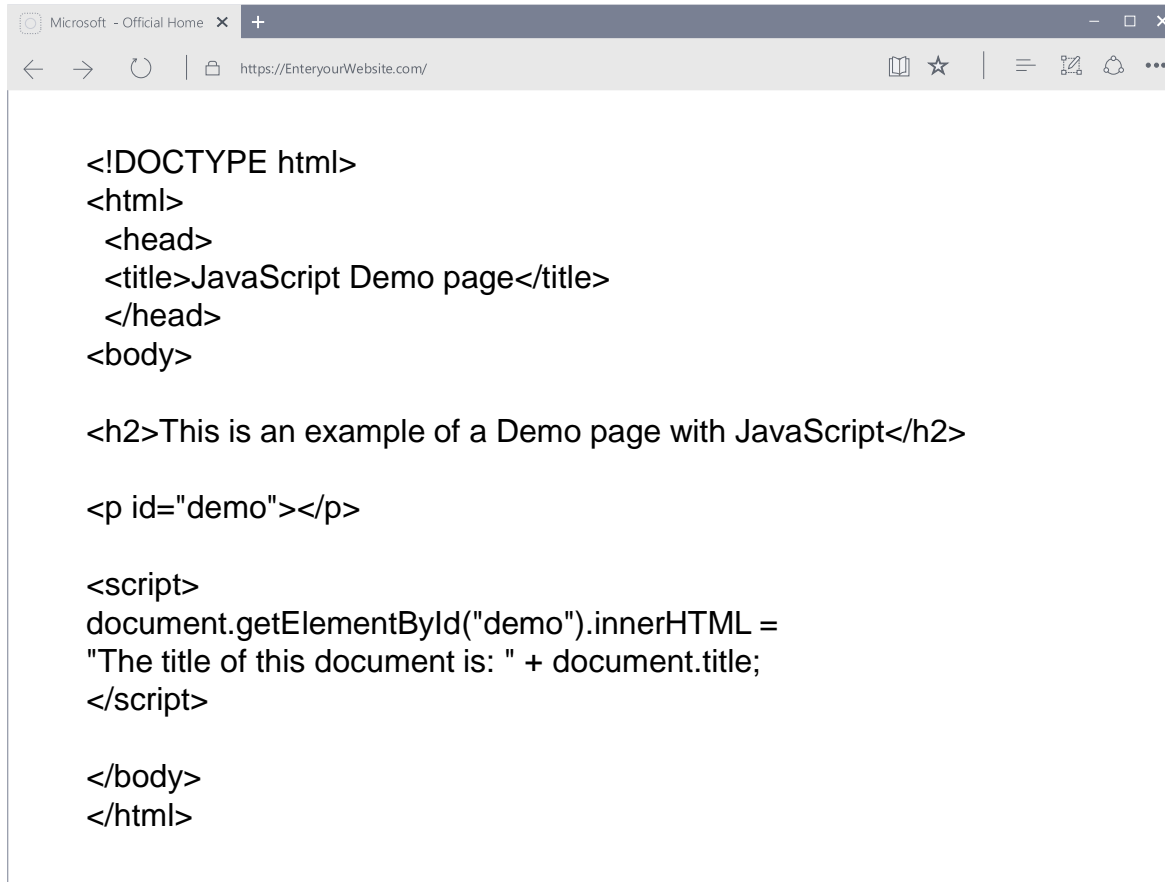
Can you count the elements in your structure?



The screenshot shows a web browser window on the right and a code editor on the left. The browser displays a page with five horizontal bars of increasing length and a button labeled "Count the elements in this page". The code editor shows the HTML and JavaScript for this page.

```
<? dom.html >
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script>
5     no-usage
6     function getElements() {
7       var b = document.getElementsByTagName('input');
8       document.getElementById("demo").innerHTML = b.length;
9     }
10  </script>
11 </head>
12 <body>
13   <input type="text" size="10"><br>
14   <input type="text" size="20"><br>
15   <input type="text" size="30"><br>
16   <input type="text" size="40"><br>
17   <input type="text" size="50"><br>
18   <input type="text" size="60"><br>
19   <p>
20     <input type="button" onclick="getElements()" value="Count the elements in this page">
21   </p>
22   <p id="demo"></p>
23 </body>
24 </html>
```

DOM Manipulation in JavaScript

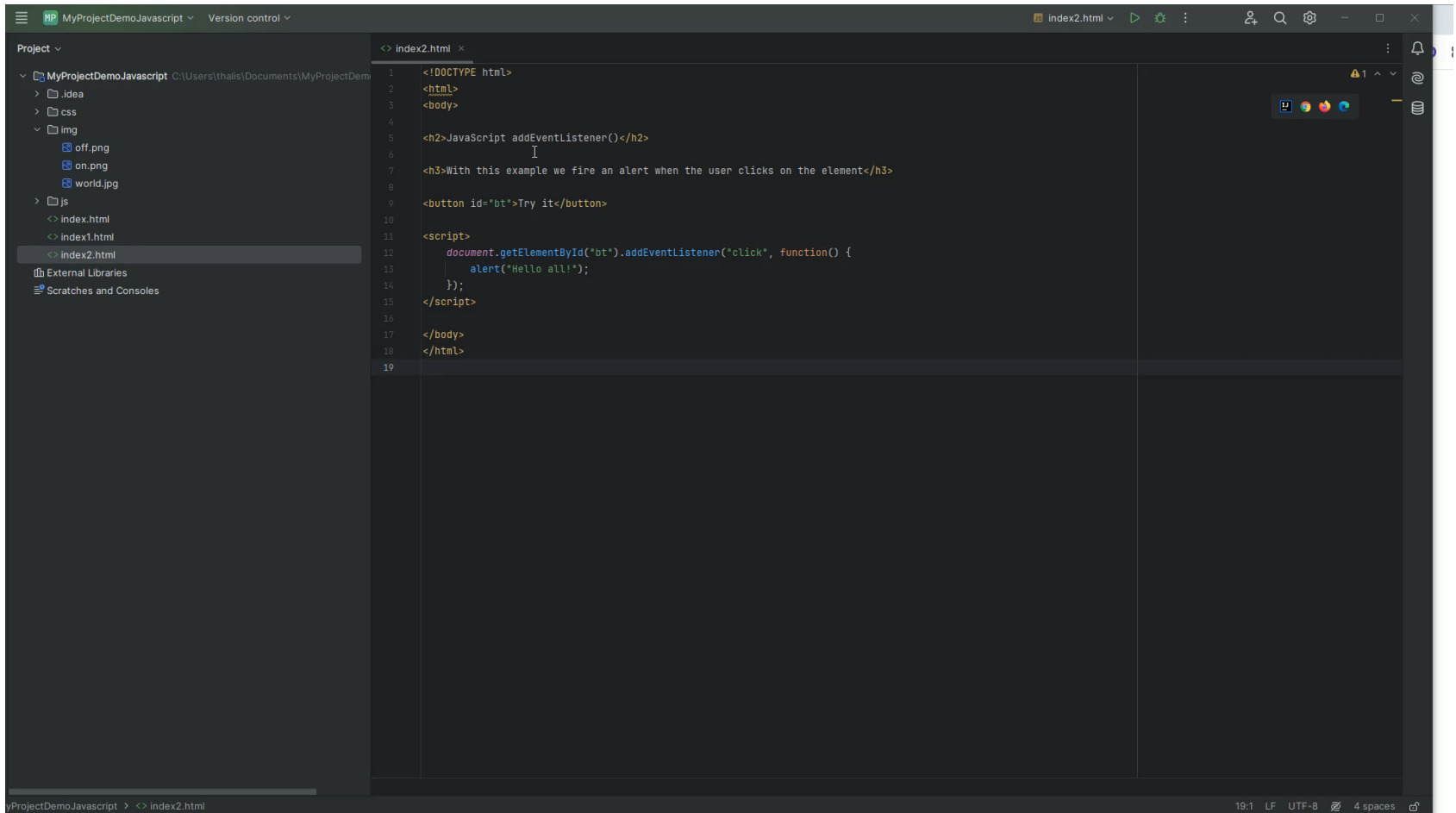


Navigator
(user agent)

Window
(tabs,
Window.innerWidth
Window.innerHeight)

Main document
(DOM, HTML)

JavaScript HTML DOM EventListener



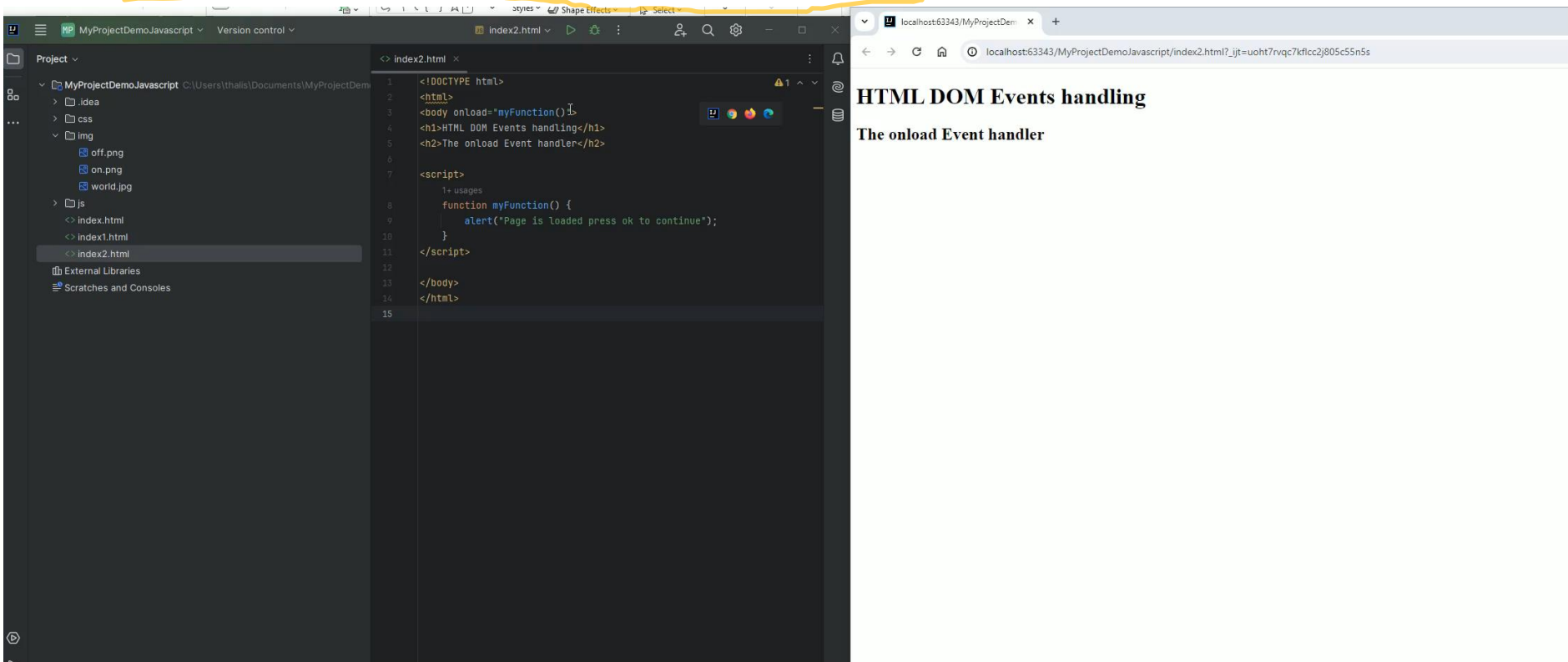
The screenshot shows a code editor with a project named 'MyProjectDemoJavaScript'. The file 'index2.html' is open, displaying the following HTML and JavaScript code:

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <h2>JavaScript addEventListener()</h2>
6
7 <h3>With this example we fire an alert when the user clicks on the element</h3>
8
9 <button id="bt">Try it</button>
10
11 <script>
12     document.getElementById("bt").addEventListener("click", function() {
13         alert("Hello all!");
14     });
15 </script>
16
17 </body>
18 </html>
19
```

The status bar at the bottom indicates the file is 'index2.html' in the 'MyProjectDemoJavaScript' project, with a line length of 19, 19:1, UTF-8 encoding, and 4 spaces for indentation.

How to Make JavaScript Execute After HTML Load onload Event

If we use JavaScript to handle a Document Object Model (DOM) then our code executes after HTML (blocked code until OK!)



The screenshot shows a development environment with an IDE on the left and a web browser on the right. The IDE displays the code for index2.html, which includes an onload event handler. The browser shows the rendered HTML with the title 'HTML DOM Events handling' and the subtitle 'The onload Event handler'.

```
<!DOCTYPE html>
<html>
<body onload="myFunction()">
<h1>HTML DOM Events handling</h1>
<h2>The onload Event handler</h2>

<script>
  1+ usages
  function myFunction() {
    alert("Page is loaded press ok to continue");
  }
</script>

</body>
</html>
```

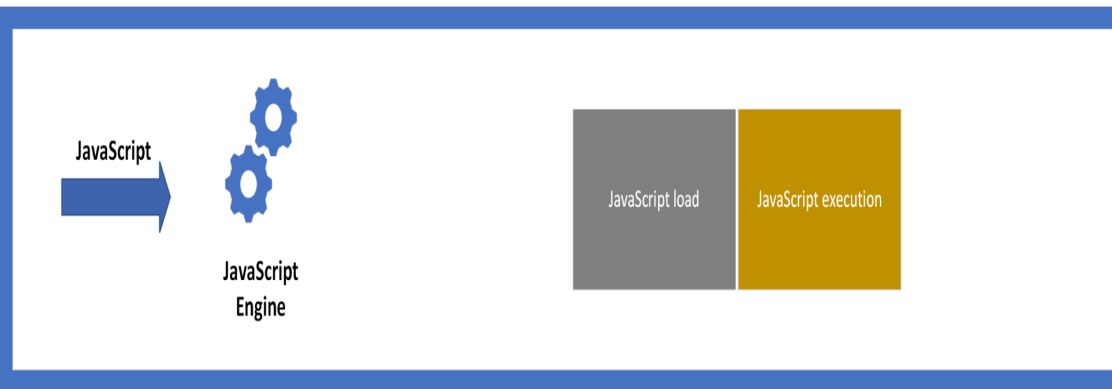
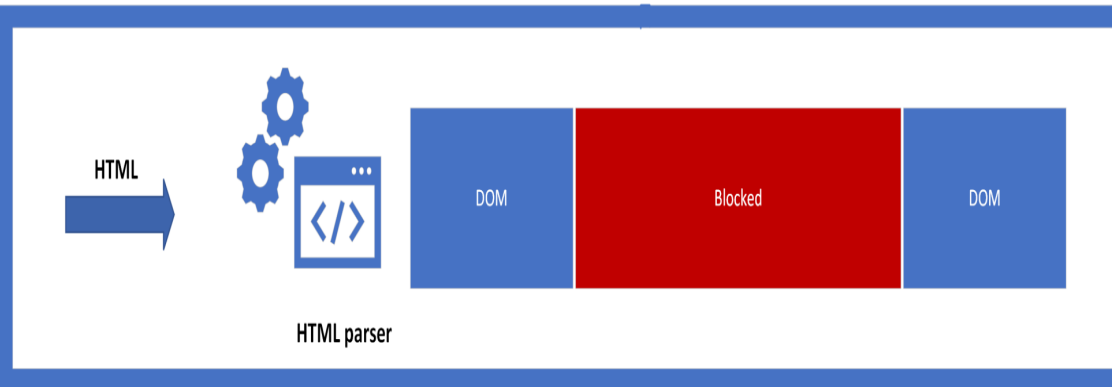
HTML DOM Events handling

The onload Event handler

Execute JavaScript: parser-blocking scripts (synchronous)



Execute JavaScript: parser-blocking scripts (synchronous)



Asynchronous JavaScript

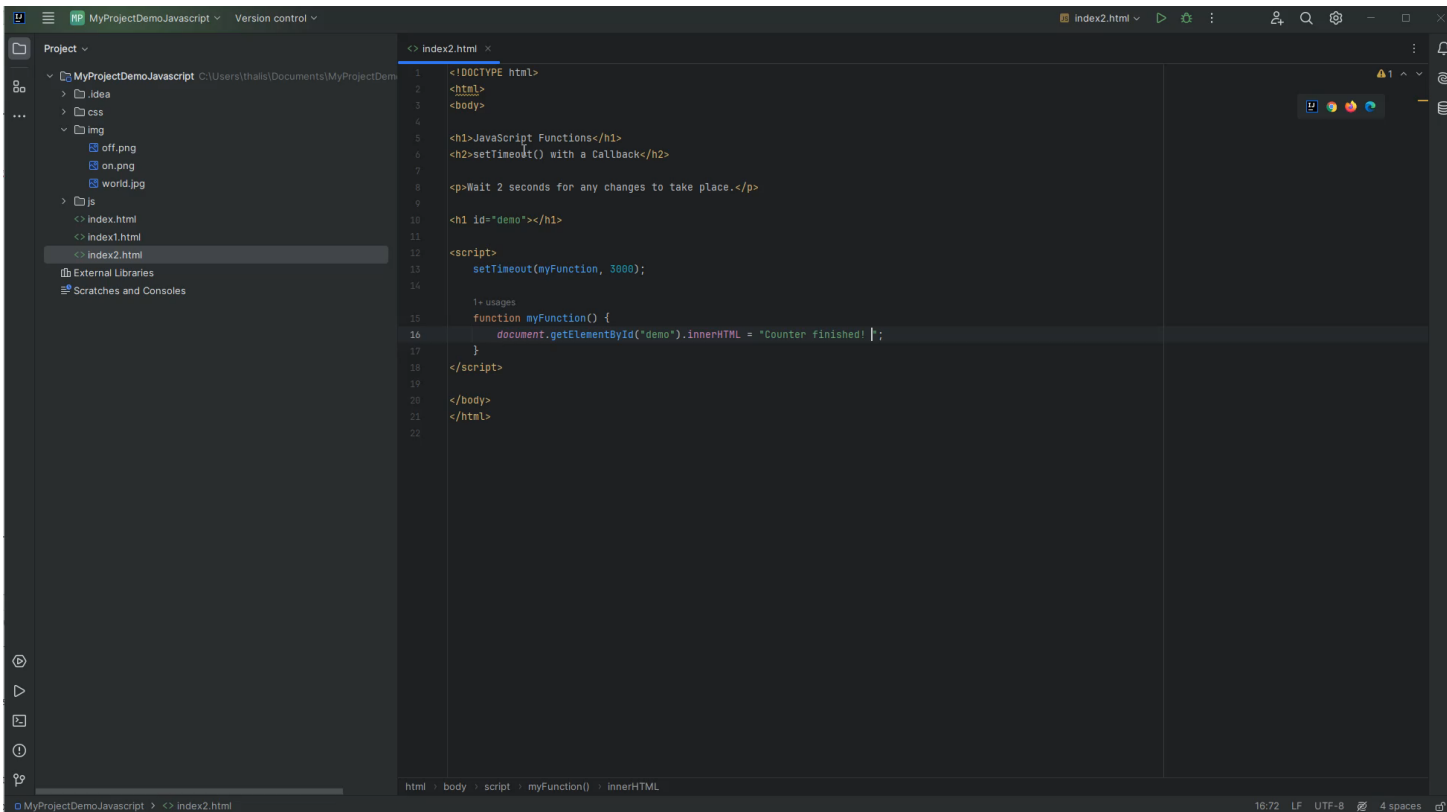
异步与同步编码：

We can run function in parallel Asynchronously vs Synchronous coding:

is executed line by line (JavaScript has single thread execution)

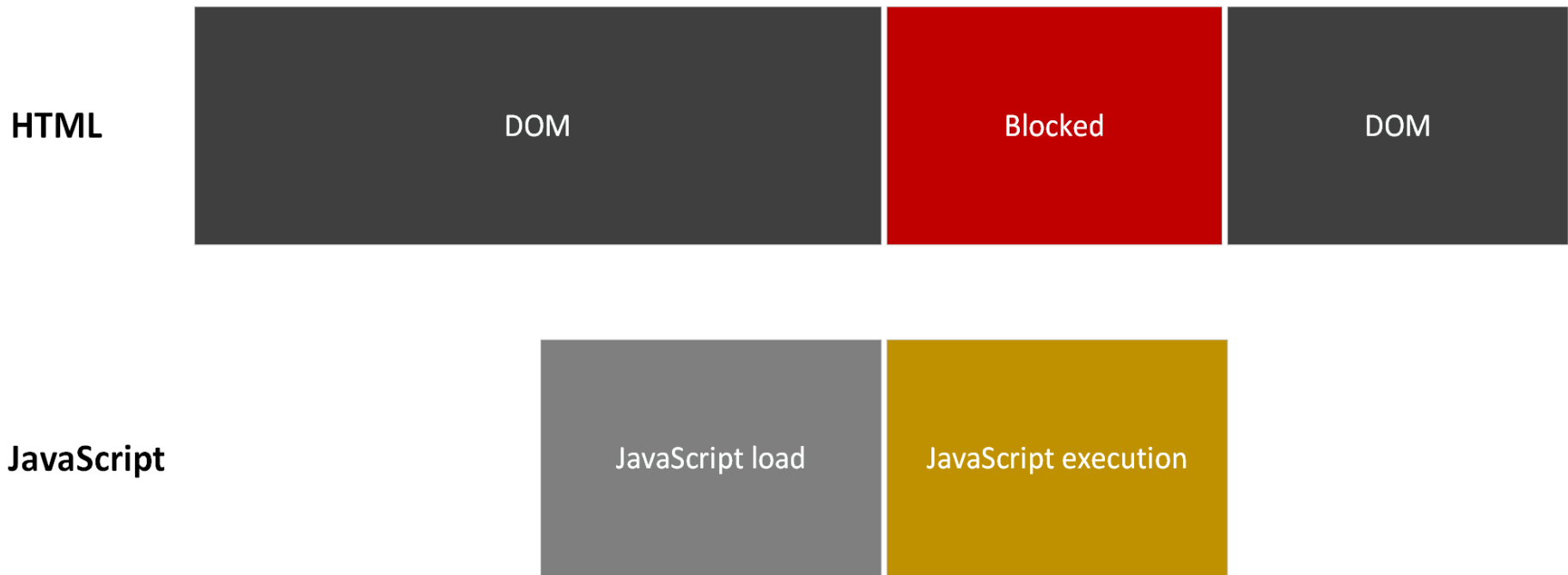
each line waits for previous line to finish

long time operation



```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <h1>JavaScript Functions</h1>
6 <h2>setTimeout() with a Callback</h2>
7
8 <p>Wait 2 seconds for any changes to take place.</p>
9
10 <h1 id="demo"></h1>
11
12 <script>
13   setTimeout(myFunction, 3000);
14
15   1<- usages
16   function myFunction() {
17     document.getElementById("demo").innerHTML = "Counter finished! ";
18   }
19 </script>
20 </body>
21 </html>
22
```

Asynchronous JavaScript



Variable Names

- JavaScript variable names cannot include any Unicode character
- First character can be any of a-z, A-Z or (_) or \$

let \$4 = 1; → Block

let _4 = 10;

var \$_\$ = "money"; → Function

var I_AM_HUNGRY = true;

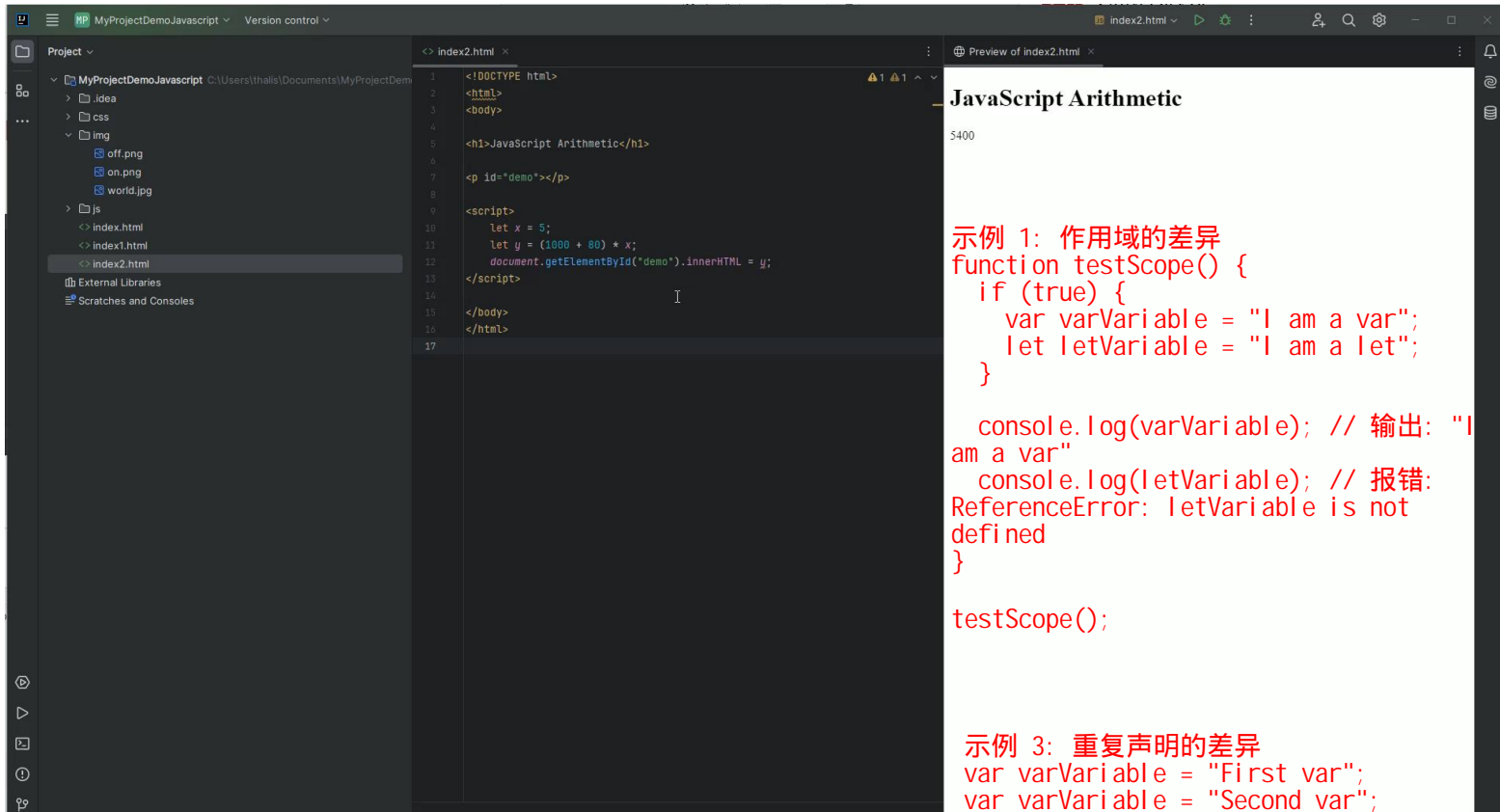
validVariable
_privateVar
\$money
myVariable123

typeof Operator

We can use the **typeof** operator to identify the data type of a Javascript variable:

```
let x = 10;  
console.log(typeof x);  
> number
```

JavaScript Arithmetic Operators Demo



The screenshot shows an IDE with a project named 'MyProjectDemoJavaScript'. The file explorer on the left shows a directory structure with files like 'index.html', 'index2.html', and 'index3.html'. The main editor shows the content of 'index2.html', which includes a script tag with the following code:

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <h1>JavaScript Arithmetic</h1>
6
7 <p id="demo"></p>
8
9 <script>
10   let x = 5;
11   let y = (1000 + 80) * x;
12   document.getElementById("demo").innerHTML = y;
13 </script>
14
15 </body>
16 </html>
17

```

The preview window on the right shows the rendered HTML, titled 'Preview of index2.html'. It displays the heading 'JavaScript Arithmetic' and the value '5400' below it.

示例 1: 作用域的差异

```

function testScope() {
  if (true) {
    var varVariable = "I am a var";
    let letVariable = "I am a let";
  }

  console.log(varVariable); // 输出: "I
                             am a var"
  console.log(letVariable); // 报错:
                             ReferenceError: letVariable is not
                             defined
}

testScope();

```

示例 3: 重复声明的差异

```

var varVariable = "First var";
var varVariable = "Second var";
console.log(varVariable); // 输出: "Second var"

```

```

let letVariable = "First let";
let letVariable = "Second let"; // 报错:
SyntaxError: Identifier 'letVariable' has
already been declared

```

String in JavaScript

Declare:

```
let abc = 'hi  
all'; → The  
same  
let def = "hi  
all";
```

\ → escape point

```
let "I\'m feeling  
lucky";
```

作用域 (Scope) :

var 声明的变量具有函数作用域，或者如果在函数外部声明，则为全局作用域。这意味着如果 var 在一个函数内部声明，它只能在该函数内部访问，但如果在函数外声明，它可以在全局范围内访问。

let 声明的变量具有块级作用域 (block scope)，即只能在包含它们的代码块 (例如：for 循环、if 语句等) 中访问。

concatenate them using +

```
let one =  
"Hello";  
let two =  
"world!!!";  
let concatenation  
= one + two
```

String methods in JavaScript

```
let myName = 'Manolis'  
> myName.length;  
7  
> myName[0];  
'M'  
> myName[myName.length-1];  
's'  
> myName.slice(0,3);  
'Man'
```

```
> myName.indexOf('lis');  
4  
> myName.slice(3);  
'olis'  
> myName.toLowerCase();  
'manolis'  
> myName.toUpperCase();  
'MANOLIS'  
>  
myName.replace('lis','s');  
'Manos'
```

Arrays in JavaScript

Arrays are objects which can hold multiple values:

```
> let numbers = ['one', 'two', 'three'];
```

```
undefined
```

```
> numbers
```

```
[ 'one', 'two', 'three' ]
```

在交互式JavaScript环境中，当你输入一个语句时，它会执行该语句并返回其结果。在这种情况下，当你输入 `let numbers = ['one', 'two', 'three'];` 时，它会声明一个名为 `numbers` 的变量，并将一个包含三个字符串元素的数组赋值给它。因为这个语句本身没有返回值，所以会显示 `undefined`。实际上，`let` 声明的结果通常是 `undefined`，表示该变量已被成功声明，但尚未赋值。

```
> let values = [1, 2, 3, 4, 5, 6, 10];
```

```
undefined
```

```
> let variety = ['one', 7896, [0, 1, 2]];
```

```
undefined
```

```
> variety
```

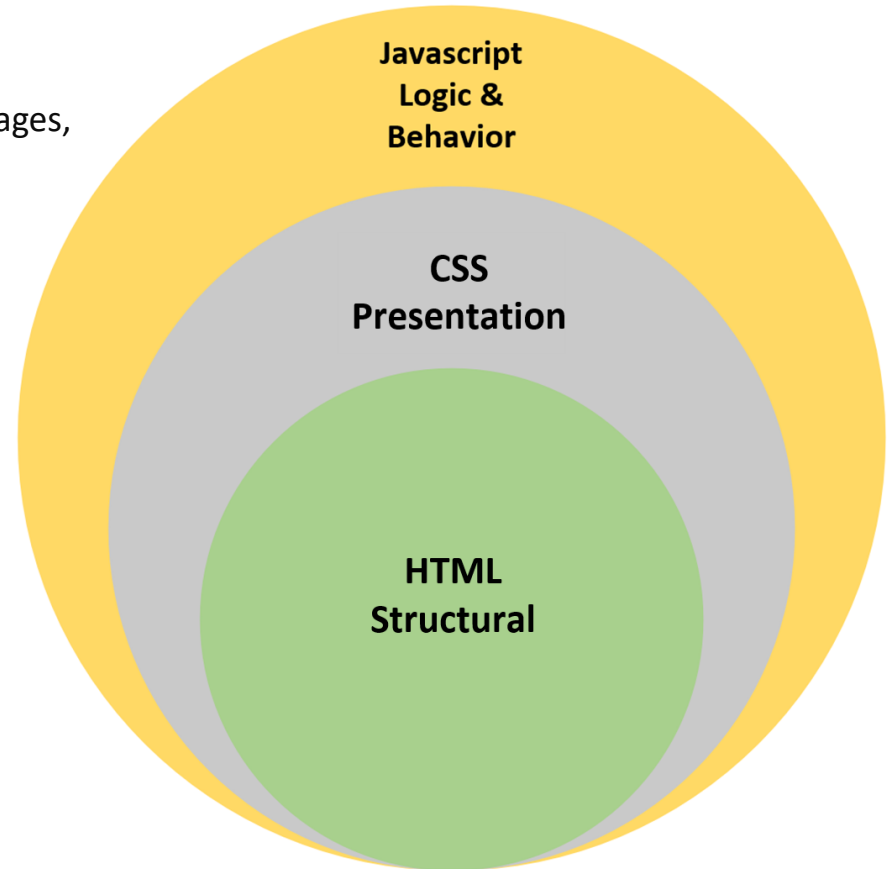
```
[ 'one', 7896, [ 0, 1, 2 ] ]
```


JavaScript and html

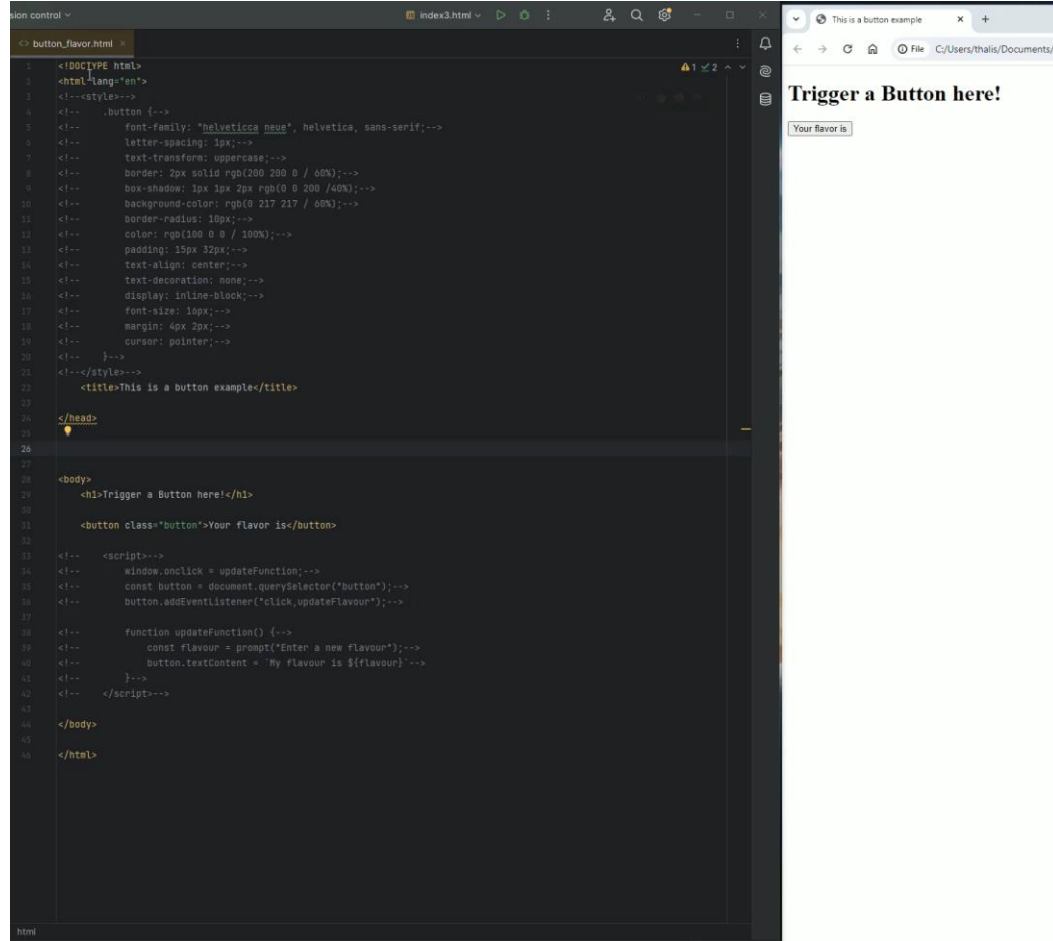
JavaScript is a scripting language utilized within an HTML document, to add functionality to the document or if you prefer to make a document dynamic.

With JavaScript, what we add complex features on web pages, specifically, we can:

- Dynamically update html page content
- Control multimedia\images content of a page
- Control, change text



Trigger a click event JavaScript– Button example HTML



The image shows a code editor on the left and a web browser on the right. The code editor displays the HTML and JavaScript code for a button click event example. The web browser shows the rendered output, which is a button labeled "Your flavor is" with the text "Trigger a Button here!" above it.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <!--style-->
4 <!-- .button {-->
5 <!--   font-family: "helvetica neue", helvetica, sans-serif;-->
6 <!--   letter-spacing: 1px;-->
7 <!--   text-transform: uppercase;-->
8 <!--   border: 2px solid rgb(200 200 0 / 40%);-->
9 <!--   box-shadow: 1px 1px 2px rgb(0 0 200 / 40%);-->
10 <!--   background-color: rgb(0 217 217 / 60%);-->
11 <!--   border-radius: 10px;-->
12 <!--   color: rgb(100 0 0 / 100%);-->
13 <!--   padding: 15px 32px;-->
14 <!--   text-align: center;-->
15 <!--   text-decoration: none;-->
16 <!--   display: inline-block;-->
17 <!--   font-size: 16px;-->
18 <!--   margin: 4px 2px;-->
19 <!--   cursor: pointer;-->
20 <!-- }-->
21 <!--/style-->
22 <title>This is a button example</title>
23
24 </head>
25
26
27 <body>
28 <h1>Trigger a Button here!</h1>
29
30 <button class="button">Your flavor is</button>
31
32
33 <!-- script -->
34 <!-- window.onclick = updateFunction;-->
35 <!-- const button = document.querySelector("button");-->
36 <!-- button.addEventListener("click,updateFlavour");-->
37
38 <!-- function updateFunction() {-->
39 <!--   const flavour = prompt("Enter a new flavour");-->
40 <!--   button.textContent = "My flavour is ${flavour}";-->
41 <!-- }-->
42 <!-- /script -->
43
44 </body>
45
46 </html>
```

JavaScript Functions

Every function contains a set of instructions in a logical sequence so that a specific result is always achieved. For function in JavaScript:

- Every function must have a specific and unique name (e.g., myFunction)
- Same rules for naming variables apply to function names
- Word "function" must precede the name of a function (e.g., function myFunction).
- Each function name is followed by a pair of parentheses (e.g., function myFunction()).
- The set of instructions for each function is contained within curly braces { and } (e.g., function myFunction() { . . . })
- The function parameters may be included within the parentheses, and there may be 0, 1, or more parameters
- Each function is called by its name (e.g., myFunction())
- Instructions within a function are executed when the function is called by its name
- Each function could be called multiple times within a script
- An HTML document may contain more than one function
- All the functions are contained within the <script> and </script> tags or wherever else code is inserted

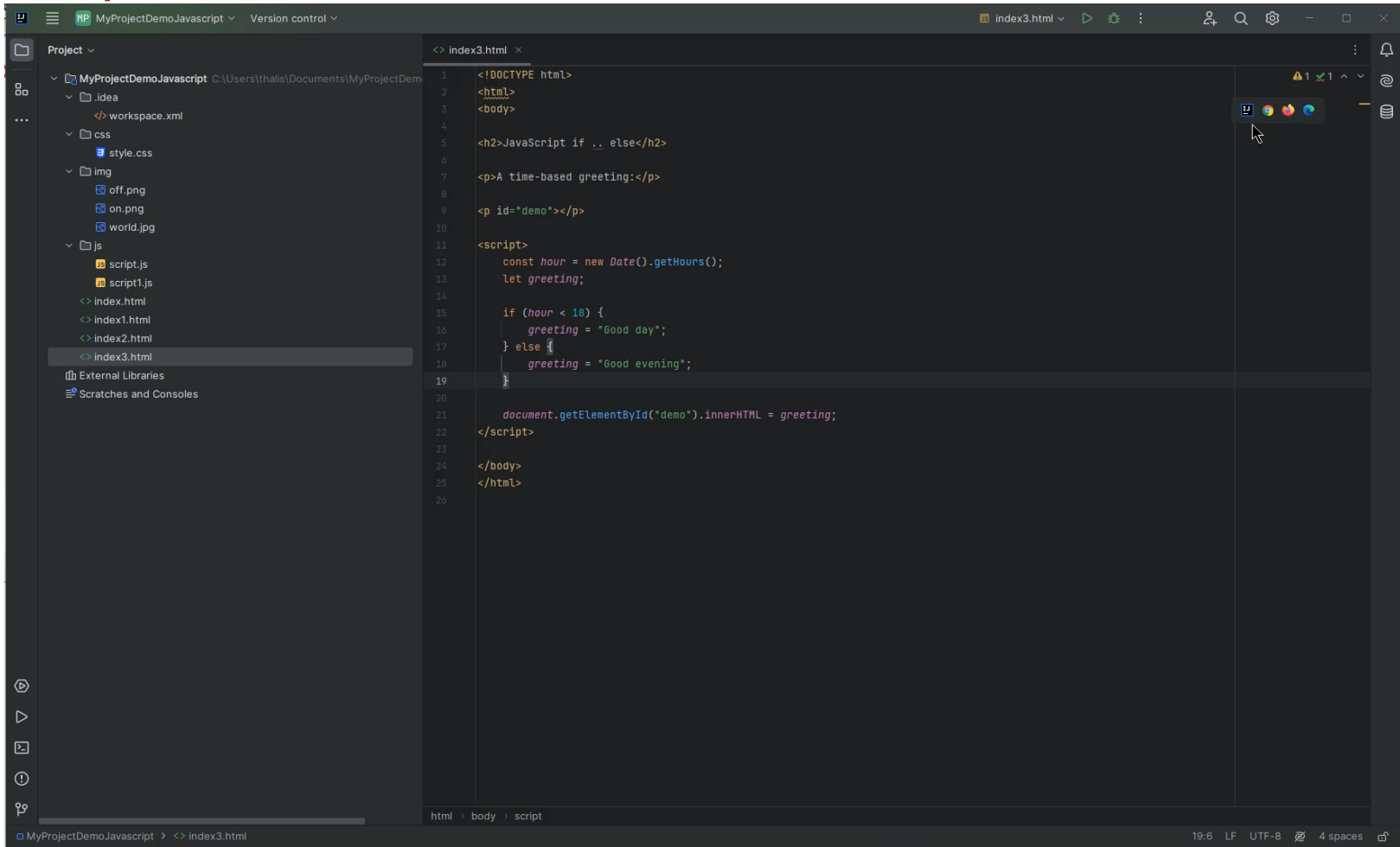
General syntax:

```
function FunctionName([param[, param[, ... param]]])  
{  
  ...  
}
```

Example:

```
<script>  
  function Hello() {  
    alert("Welcome!");  
  }  
</script>
```

JavaScript Conditions



```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <h2>JavaScript if .. else</h2>
6
7 <p>A time-based greeting:</p>
8
9 <p id="demo"></p>
10
11 <script>
12   const hour = new Date().getHours();
13   let greeting;
14
15   if (hour < 18) {
16     greeting = "Good day";
17   } else {
18     greeting = "Good evening";
19   }
20
21   document.getElementById("demo").innerHTML = greeting;
22 </script>
23
24 </body>
25 </html>
26
```

JavaScript Loops

for (Initial counter value; Condition; Counting step) {
 // code block to be executed
}

Example:

```
JavaScript + No-Library (pure JS) ▼
1  var i;
2  for (i=0; i<5; i++)
3  {
4      document.write("Number is: " + i);
5      document.write("<br />");
6  }
```

Number is: 0
Number is: 1
Number is: 2
Number is: 3
Number is: 4

while (condition) {
 // code block to be executed
}

Example:

```
JavaScript + No-Library (pure JS) ▼
1  var i=0;
2  while (i<5)
3  {
4      document.write("Number is: " + i);
5      document.write("<br />");
6      i=i+1;
7  }
```

Number is: 0
Number is: 1
Number is: 2
Number is: 3
Number is: 4

JavaScript Cookies

Cookies are small text files that are stored on the computer and contain data in the form of name=value pairs. For example: visitor=vist123.

To write (or store) and read cookies on the machine, the command used is `document.cookie`.
`document.cookie = "cookieName=cookieValue";`

Overall, the cookie is stored as a string, and any additional parameters you can pass are separated by a question mark (;).

Example: `document.cookie = "userid=Fe80gRCCijyH4mgdO; expires=Sun, 13 Jun 2021 20:31:59 GMT; path=/"`

This is a JavaScript function to create a cookie:

Cookies store user/visitor information

When a browser requests a web page from the server, page cookies are added to that request



```
function setCookie(name, value, days) {  
    var expires = "";  
    if (days) {  
        var date = new Date();  
        date.setTime(date.getTime() + (days * 24 * 60 * 60 * 1000));  
        expires = "; expires=" + date.toUTCString();  
    }  
    document.cookie = name + "=" + value + expires + "; path=/";  
}
```



Privacy concerns

- Tracking
- Profiling
- Data Collection
- Third-Party vs First-Party Cookies

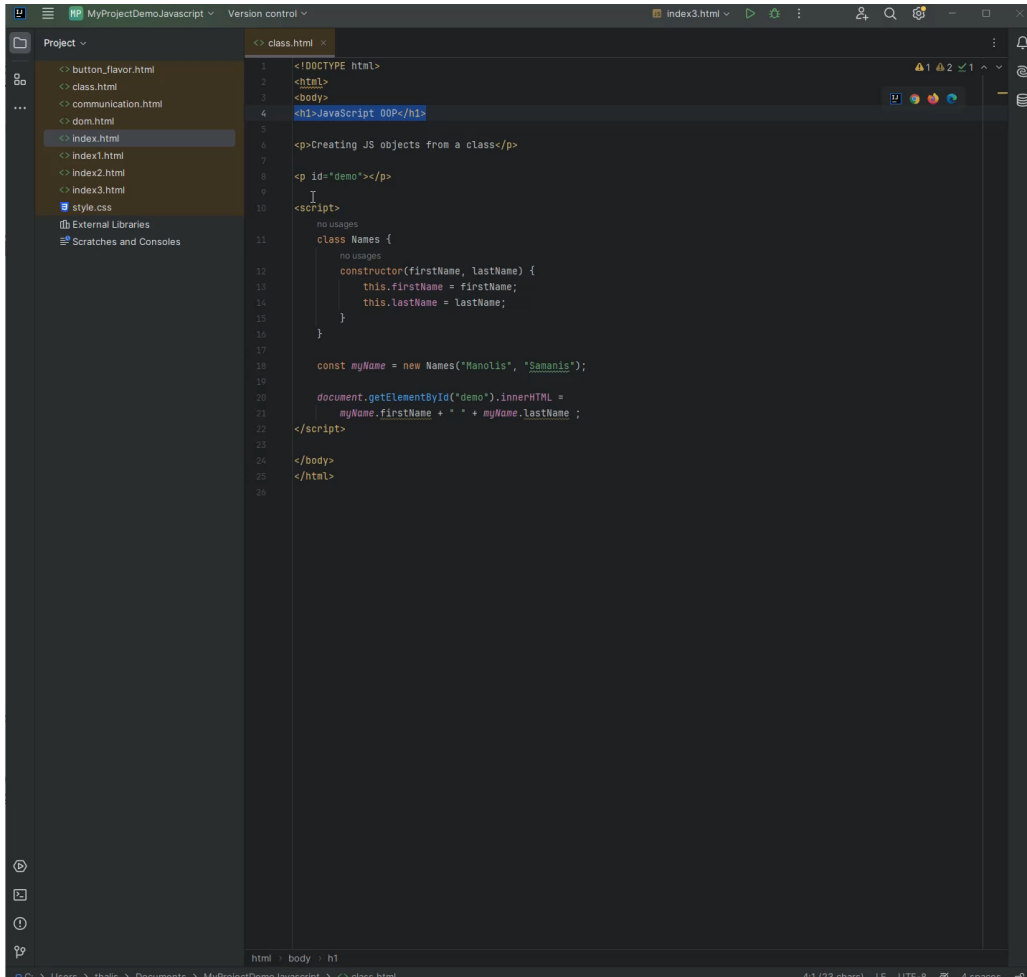
How to protect

- Browser Privacy Settings: private browsing, cookie blocking, and third-party cookie restrictions
- Cookie Blockers, extensions or browsers
- Anonymization with Tor (not 100% safe)
- Use of Tails, a portable operating system that protects against surveillance and hides your identity

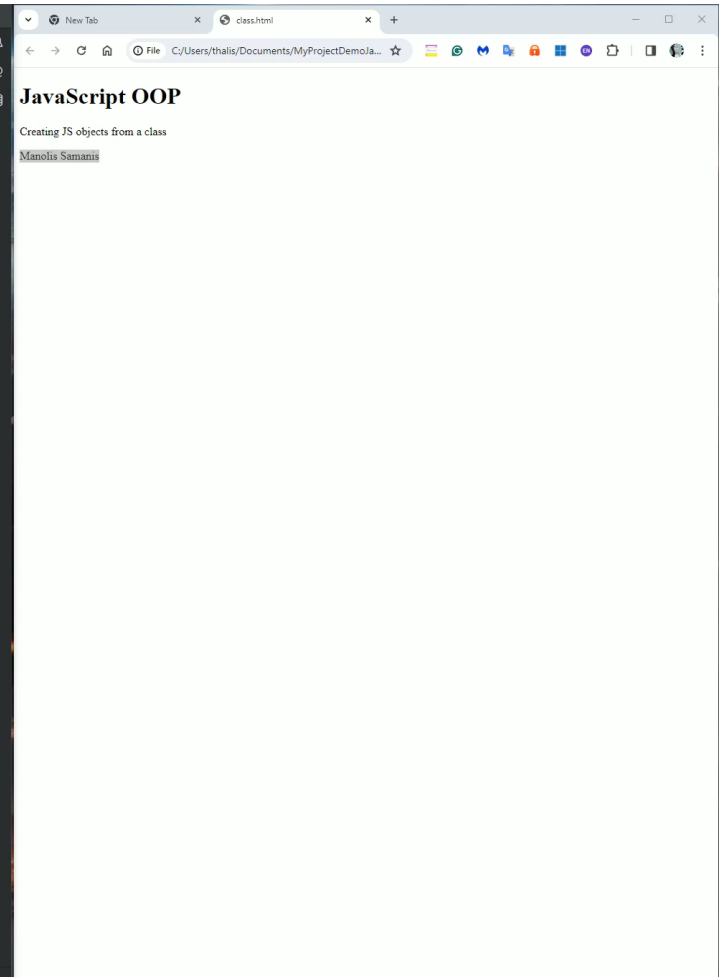
Object-Oriented Programming With JavaScript (OOP)

The JavaScript language has been designed on an object-oriented basis. With OOP a developer can:

- Create their own objects and organize better the code by making it more flexible and maintainable
- Use the pre-made built-in objects provided by JavaScript (JSON, Date, Math)
- We use OOP to model or better describe real-world examples
- The objects can contain data or methods. With objects we incorporate the data and their behavior into a block of code
- Objects can interact with one another
- We can use an API methods to access and communicate with the object



```
1 <!DOCTYPE html>
2 <html>
3 <body>
4 <h1>JavaScript OOP</h1>
5
6 <p>Creating JS objects from a class</p>
7
8 <p id="demo"></p>
9
10 <script>
11     class Names {
12         constructor(firstName, lastName) {
13             this.firstName = firstName;
14             this.lastName = lastName;
15         }
16     }
17
18     const myName = new Names("Manolis", "Samanis");
19
20     document.getElementById("demo").innerHTML =
21         myName.firstName + " " + myName.lastName ;
22 </script>
23
24 </body>
25 </html>
26
```



JavaScript OOP

Creating JS objects from a class

Manolis Samanis

JavaScript Built-in objects

JSON (JavaScript Object Notation) is used for storing and transmitting data, primarily in web applications.

- JSON objects share many similarities with Array objects.
- Data is recorded in the format key:value.

Example:

```
JavaScript + No-Library (pure JS) ▼ Tidy
1 let products = {};
2
3 products.apples = 1.5;
4 products.bananas = 2.5;
5 products.oranges = 1.1;
6 products.pears = 1.9;
7
8 for(key in products) {
9   console.log(key, products[key]);
10 }
```

"apples", 1.5
"bananas", 2.5
"oranges", 1.1
"pears", 1.9
>_

The date object is used for managing dates and times

Example:

```
1 <div id = "timer"></div>
2 <script>
3   var d = new Date();
4   var t = document.getElementById("timer");
5   t.innerHTML = d.getHours();
6 </script>
```

20

Objects and Instances

In JavaScript, anything stored in a variable is an object. To use an object, we first need to create an instance of that object.

Creating an object is done using the new operator -> **var d = new Date();**

Each instance (or object, as we will call it) has:

- Properties or characteristics (features, properties, or fields)
- Functions or methods
- Ability to handle events

An object can have sub objects (child objects) -> **window.document**

To add an event handler, we use the method -> **addEventListener("eventname", functionname);**

```
1 var btn = document.getElementById("myButton");
2 btn.addEventListener("click", getValue);
3
4 function getValue() {
5     ...
6 }
```

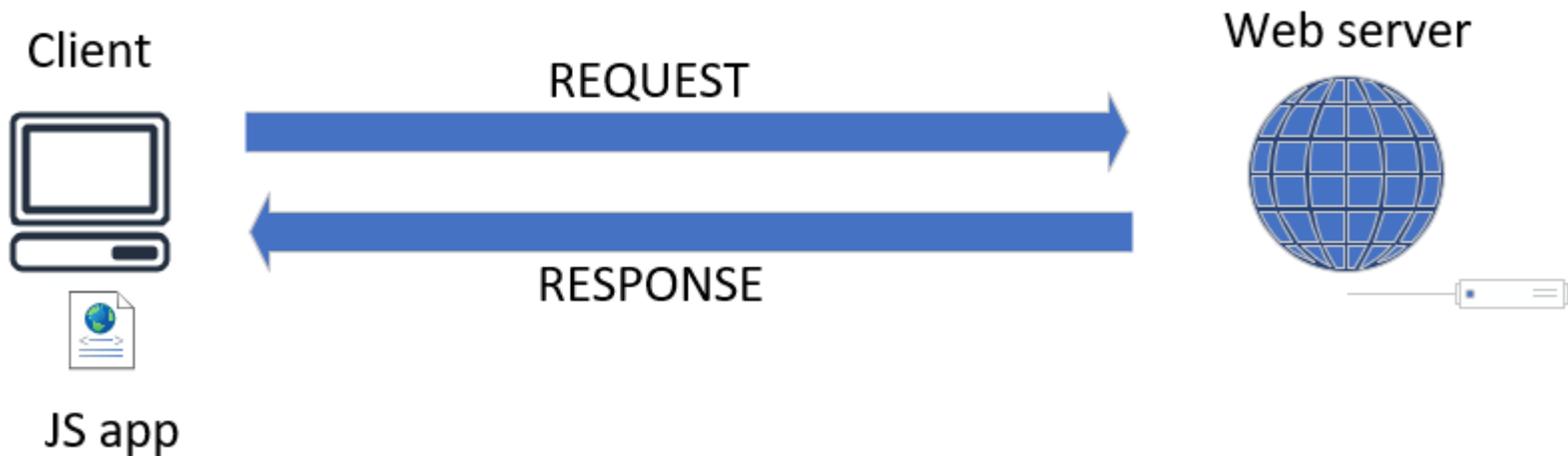
JavaScript Custom Objects with functions

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <h2>JavaScript Custom Objects with functions</h2>
6
7 <p id="demo"></p>
8
9 <button id="displayButton">What are my car details?</button>
10
11 <div id="output"></div>
12
13 <script>
14     var myObject = {
15         make: "Corse",
16         year: "2015",
17         price: 5000,
18         color: "blue"
19     };
20
21     no images
22     function displayObjectValue() {
23         var outputDiv = document.getElementById("output");
24         outputDiv.innerHTML = "make: " + myObject.make + "<br>" + "year: " + myObject.year + "<br>" +
25             "price: " + myObject.price + "<br>" + "color: " + myObject.color;
26     }
27     document.getElementById("displayButton").addEventListener("click", displayObjectValue);
28 </script>
29
30
31
32 </body>
33 </html>
34
```

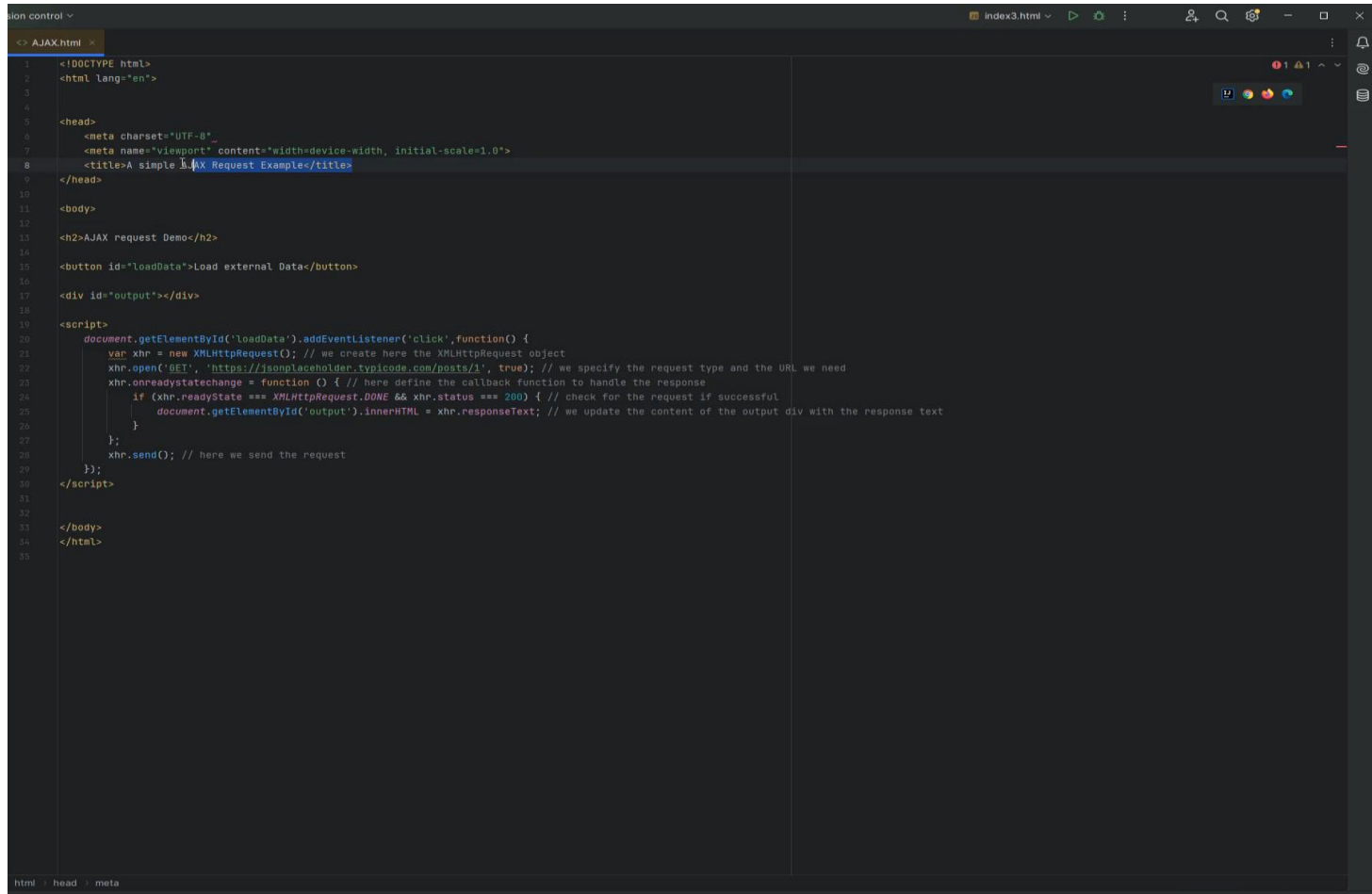
Asynchronous JavaScript and AJAX

AJAX (Asynchronous JavaScript And XML):

- It can communicate with remote web servers in an asynchronous manner
- It requests data from web servers dynamically



Asynchronous JavaScript and AJAX



The screenshot shows a code editor with a dark theme. The file is named 'AJAX.html'. The code is an HTML document with a title 'A simple AJAX Request Example'. It contains a button with the text 'Load external Data' and a div with the id 'output'. The script section uses XMLHttpRequest to fetch data from 'https://jsonplaceholder.typicode.com/posts/1'. The callback function checks if the request is successful (status 200) and updates the 'output' div with the response text. The browser's developer tools are open, showing the 'html' tab with the document structure.


```
1 <!DOCTYPE html>
2 <html lang="en">
3
4
5 <head>
6   <meta charset="UTF-8">
7   <meta name="viewport" content="width=device-width, initial-scale=1.0">
8   <title>A simple AJAX Request Example</title>
9 </head>
10
11 <body>
12
13   <h2>AJAX request Demo</h2>
14
15   <button id="loadData">Load external Data</button>
16
17   <div id="output"></div>
18
19   <script>
20     document.getElementById('loadData').addEventListener('click', function() {
21       var xhr = new XMLHttpRequest(); // we create here the XMLHttpRequest object
22       xhr.open('GET', 'https://jsonplaceholder.typicode.com/posts/1', true); // we specify the request type and the URL we need
23       xhr.onreadystatechange = function () { // here define the callback function to handle the response
24         if (xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) { // check for the request if successful
25           document.getElementById('output').innerHTML = xhr.responseText; // we update the content of the output div with the response text
26         }
27       };
28       xhr.send(); // here we send the request
29     });
30   </script>
31
32
33 </body>
34 </html>
35
```

Suggested resources for further reading

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

JavaScript: The Definitive Guide, Author: David Flanagan

JS editor VS code



Visual Studio Code

Docs Updates Blog API Extensions FAQ Learn

Search Docs

Download

Code editing. Redefined.

Free. Built on open source. Runs everywhere.

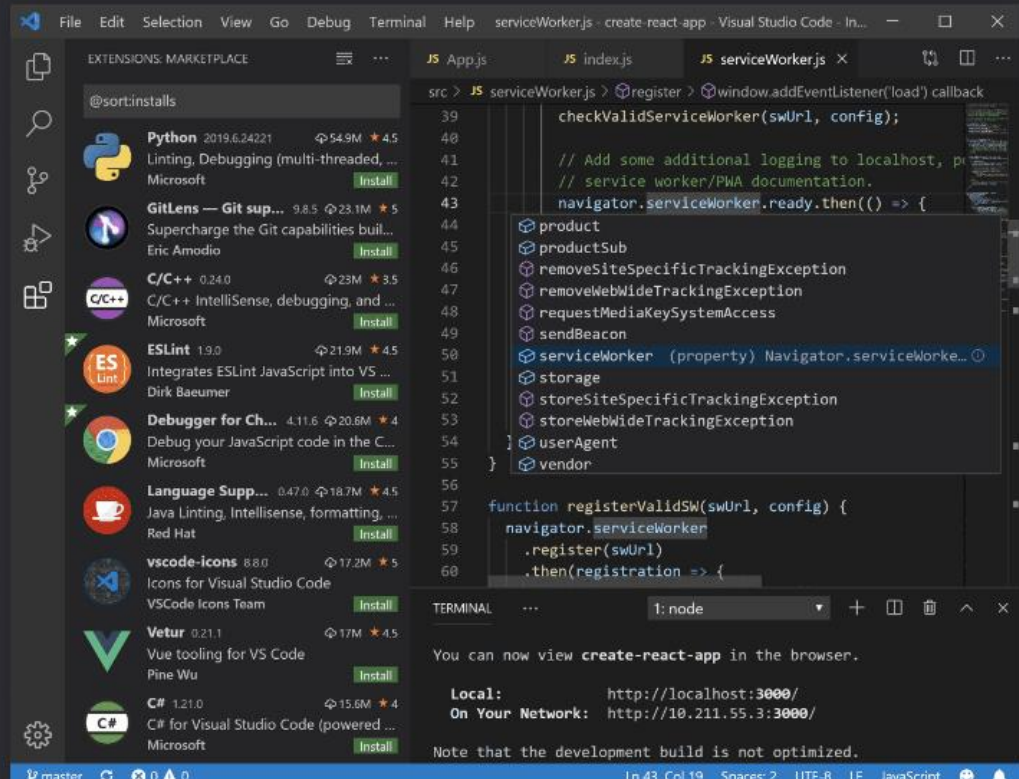
Download for Windows

Stable Build

Stable Insiders

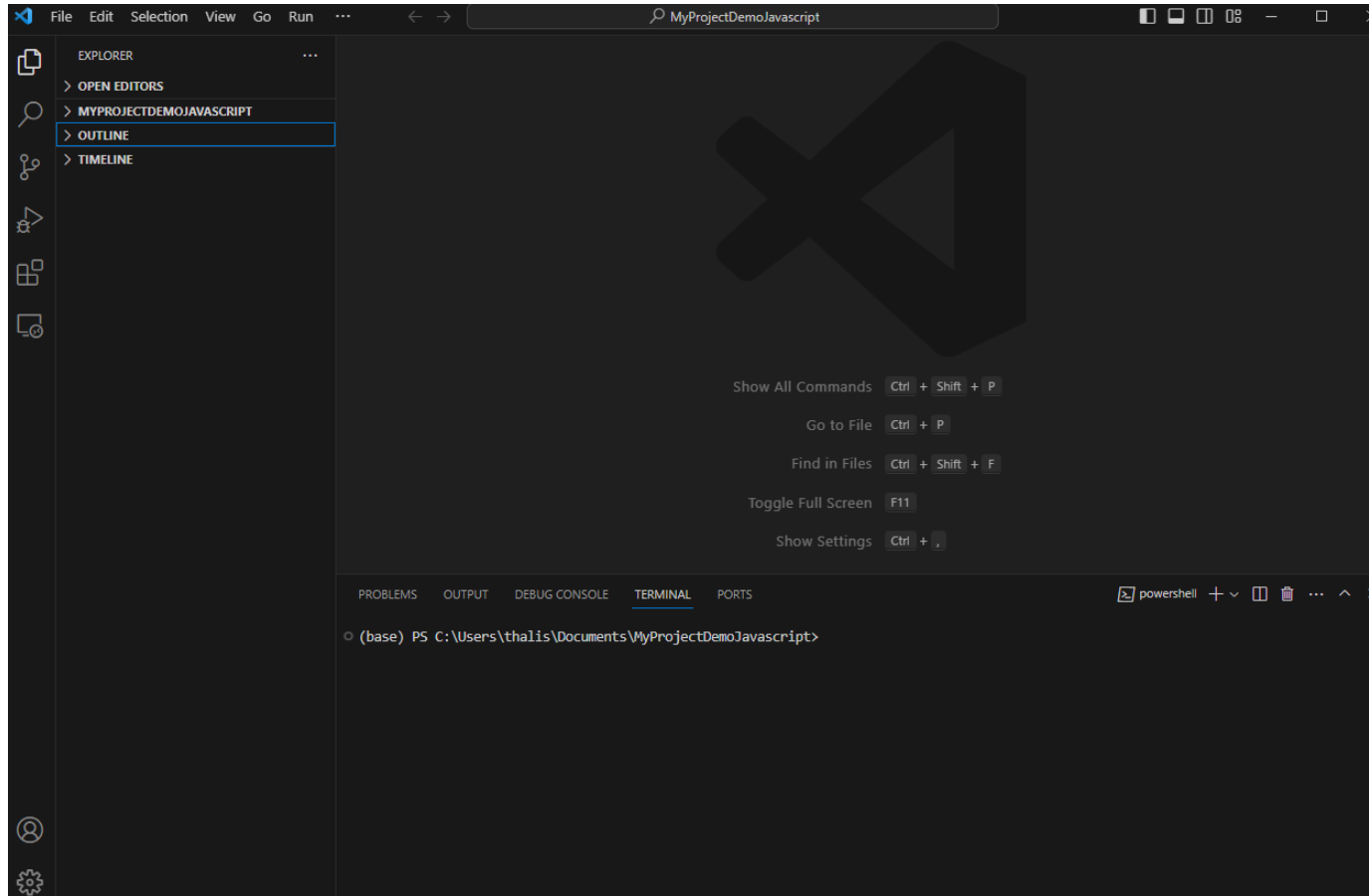
macOS	Universal	↓	↓
Windows x64	User Installer	↓	↓
Linux x64	.deb .rpm	↓ ↓	↓ ↓

Other downloads or open on web

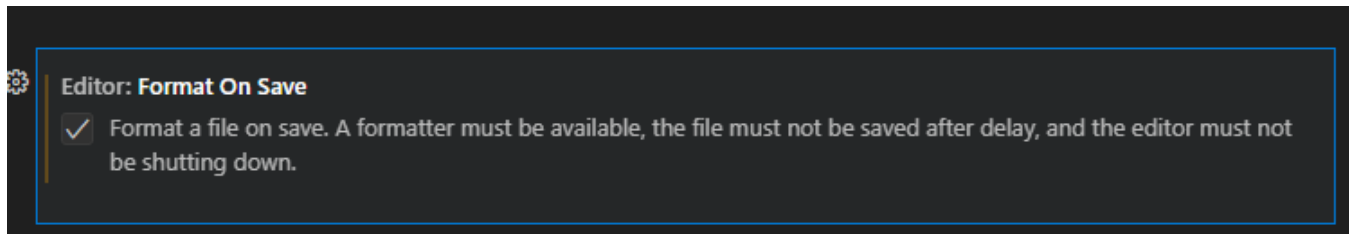
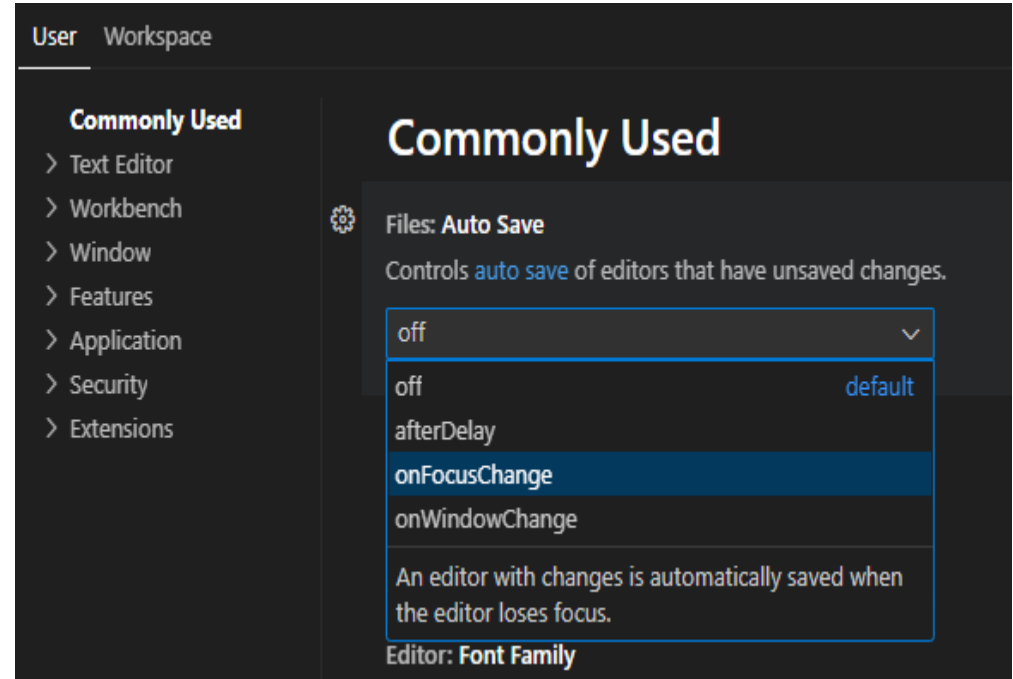
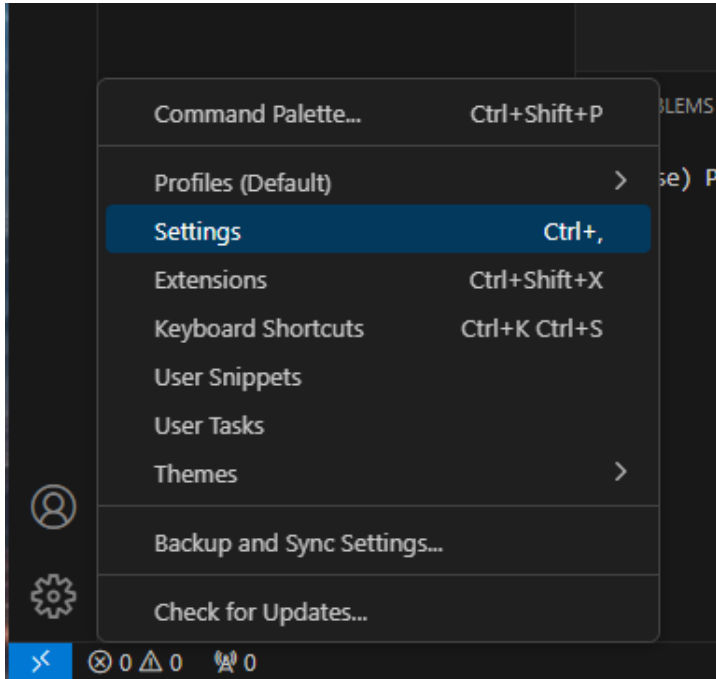


The screenshot shows the Visual Studio Code interface. On the left is the 'EXTENSIONS: MARKETPLACE' sidebar with a list of extensions including Python, GitLens, C/C++, ESLint, Debugger for Chrome, Language Support for Java, vscode-icons, Vetur, and C#. The main editor area shows a JavaScript file named 'serviceWorker.js' with code for registering a service worker. A dropdown menu is visible over the code, showing various browser APIs like 'product', 'productSub', 'removeSiteSpecificTrackingException', etc. At the bottom is a terminal window showing the output of a command, indicating that the application is running on localhost:3000.

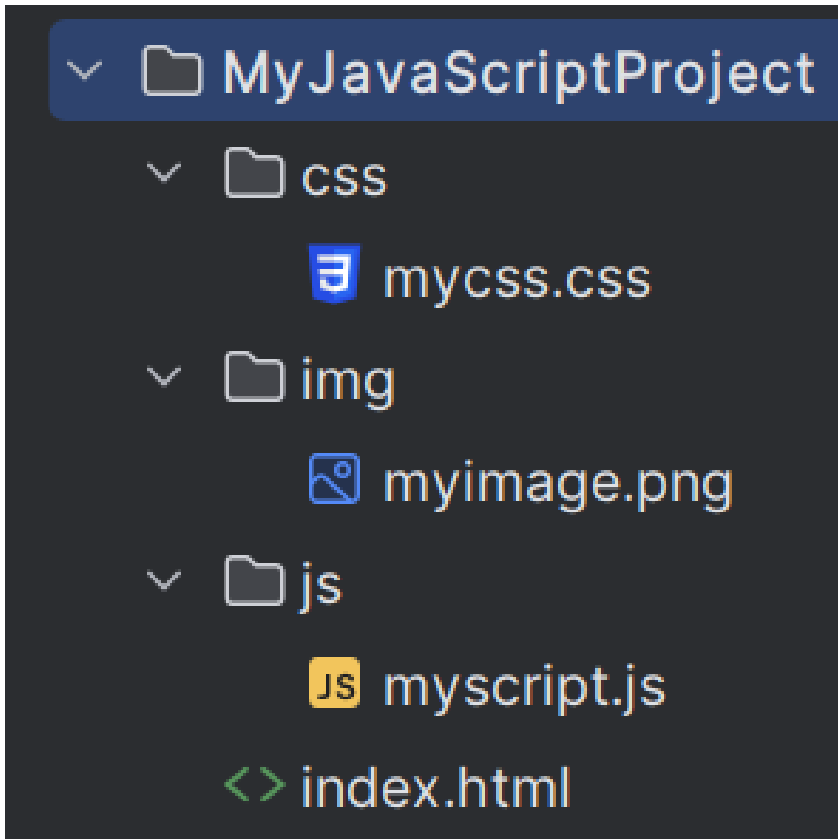
JS editor VS code



JS editor VS code



How to organize your project folder and file structure

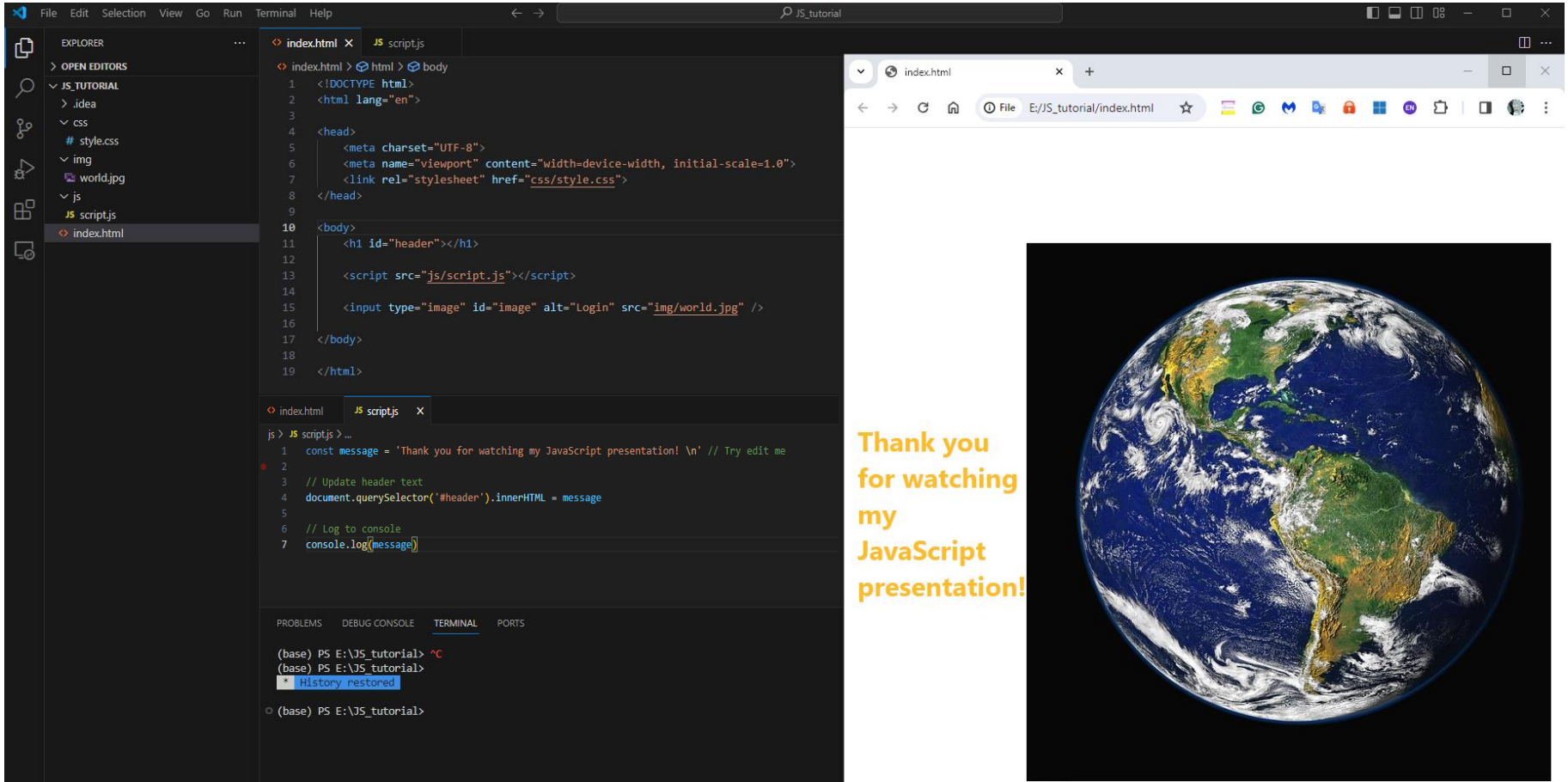


`<link rel="stylesheet" type="text/css"
href="css/mycss.css">`

``

`<script src="js/myscript.js"></script>`

JS editor VS code



The screenshot displays the Visual Studio Code (VS Code) editor interface. The Explorer panel on the left shows the project structure with files like `style.css`, `world.jpg`, `js`, and `script.js`. The main editor area shows the `index.html` file with the following content:

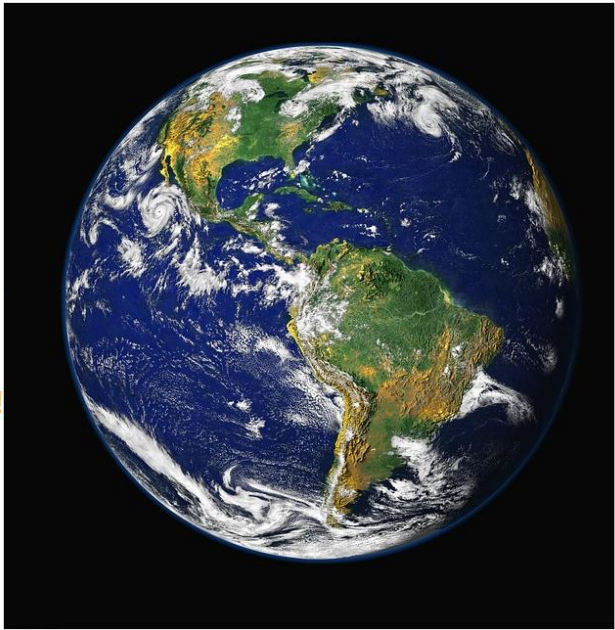
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="css/style.css">
</head>
<body>
  <h1 id="header"></h1>
  <script src="js/script.js"></script>
  <input type="image" id="image" alt="Login" src="img/world.jpg" />
</body>
</html>
```

The `JS script.js` file is also open, showing the following JavaScript code:

```
js > JS script.js > ...
1  const message = 'Thank you for watching my JavaScript presentation! \n' // Try edit me
2
3  // Update header text
4  document.querySelector("#header").innerHTML = message
5
6  // Log to console
7  console.log(message)
```

The browser preview on the right shows the rendered HTML, displaying the text "Thank you for watching my JavaScript presentation!" and a login button with a world map image. The terminal window at the bottom shows the command prompt with the message "History restored".

Thank you
for watching
my
JavaScript
presentation!



Have any questions?

