

Contents

Day-1.....	4
1.HTML ATTRIBUTES:.....	4
2. HTML INPUT TAGS:	5
CSS:.....	6
INLINE CSS:.....	6
INTERNAL CSS:.....	7
EXTERNAL CSS:	8
CSS SELECTORS:.....	9
CSS ID SELECTORS:.....	10
CLASS SELECTOR:.....	11
UNIVERSAL SELECTOR:.....	12
CSS VARIABLES	13
Local Variables:.....	13
Override/Global Variables:	13
DAY-2	14
Internal Js:.....	14
External Js:	15
Functions	16
SCOPE:	16
LOCAL SCOPE:.....	16
GLOBAL SCOPE:	17
ES6 STANDARDS:.....	17
let:.....	17
Const:.....	18
Arrow Function:.....	18
Promises:.....	19
JS CLASSES:.....	19
JS class syntax:.....	19
Js Class Methods:.....	20
JavaScript Event Handlers.....	20
addEventListener.....	21

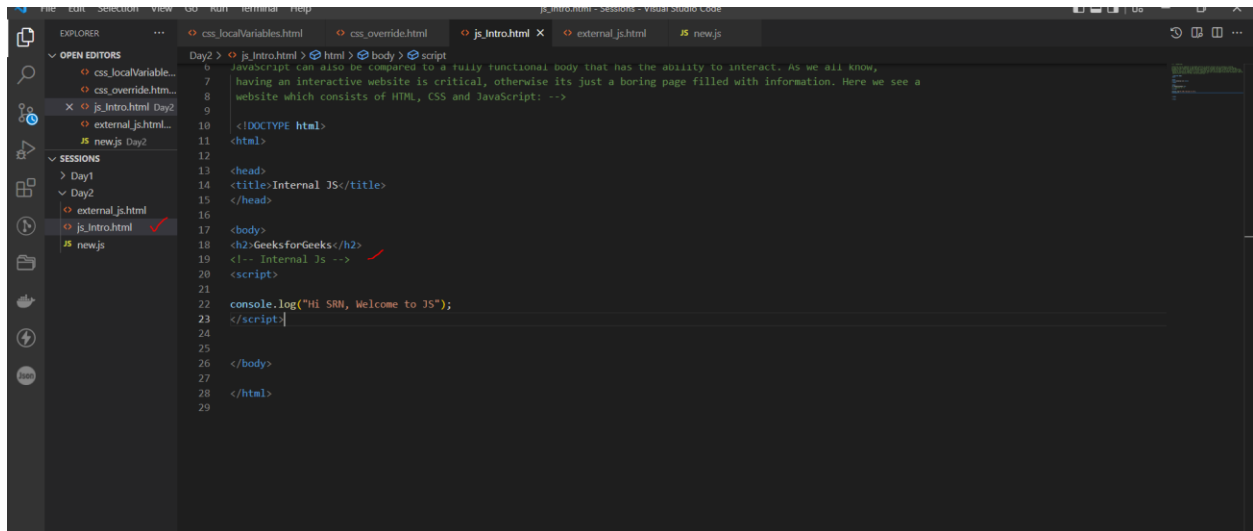
remove:.....	21
DAY-3	22
<i>What are the different types of event handlers provided by JavaScript?.....</i>	<i>22</i>
Pass by Value:.....	24
Pass by Reference:.....	25
JS ASYNC/AWAIT:	29
Async/Await:	29
Object Keys:	30
Object Entries:.....	32
HTML_JS_CUSTOM ATTRIBUTES:	34
EXAMPLE-1:	34
EXAMPLE-2	35
Html & ExternalCss:	36
EXAMPLE:3	36
DAY-4	37
LOCAL STORAGE:	37
Example1:.....	37
Objects:.....	38
Creating a JavaScript Object.....	38
Using an Object Literal.....	39
Using the JavaScript Keyword new.....	40
Example1:.....	40
Example2:.....	41
Example3:.....	41
Example4:.....	41
ArrayIteration:.....	42
arrayForEach:	42
ArrayMap:	43
ArrayFilter:	43
ArrayReduce:.....	44
ArrayIndexOf:.....	45
ArrayLastIndexOf:.....	45
ArrayFind:.....	45

CreateElement	46
JSON:.....	47
Why Use JSON?	47
Storing Data	48
JSON PARSE:	48
Example1:.....	48
Example2:.....	49
Example3:.....	49
JSON Stringify:.....	49
Stringify object:	49
Example-1:.....	50
Stringify Array:.....	50
Example-2:.....	50
JS STRING PROPERTIES:	51
Sample example:	51
Example1:.....	51
JS STRING METHODS:	51
indexOf():	51
Example2:.....	51
lastIndexOf():.....	52
Example3:.....	52
Replace():	53
Example4:.....	53
Slice():	53
Example5:.....	54
toString():	54
Example6:.....	54
toLowerCase():	55
Example7:.....	55
toUpperCase():	55
Example8:.....	55
JS Current Date and Time:	56

DAY-2

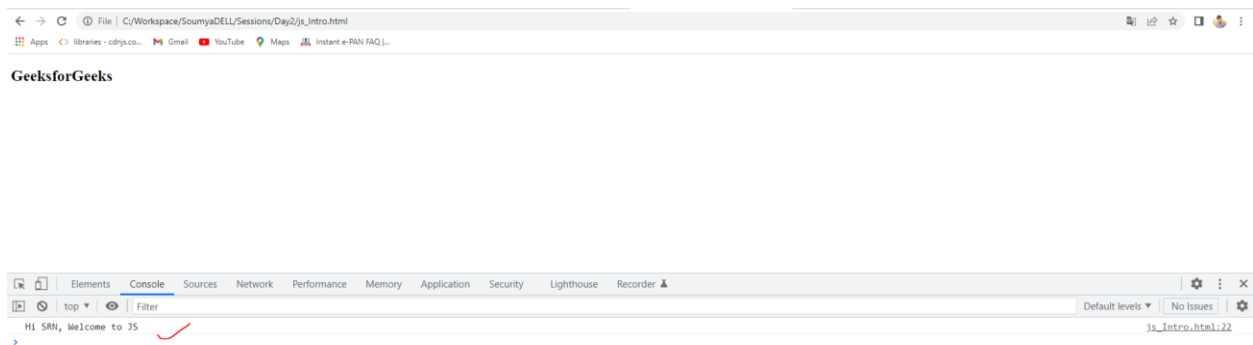
JavaScript :

Internal Js:



```
Day2 > js_intro.html > html > body > script
6  JavaScript can also be compared to a fully functional body that has the ability to interact. As we all know,
7  having an interactive website is critical, otherwise its just a boring page filled with information. Here we see a
8  website which consists of HTML, CSS and JavaScript: -->
9
10 <!DOCTYPE html>
11 <html>
12
13 <head>
14 <title>Internal JS</title>
15 </head>
16
17 <body>
18 <h2>GeeksforGeeks</h2>
19 <!-- Internal JS -->
20 <script>
21
22 console.log("Hi SRN, Welcome to JS");
23 </script>
24
25
26 </body>
27
28 </html>
29
```

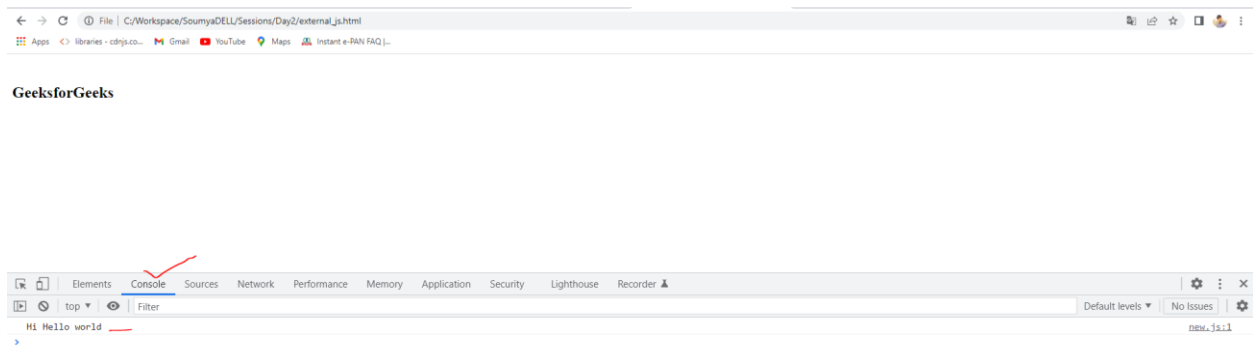
OUTPUT: check outputs in browser console.



External Js:

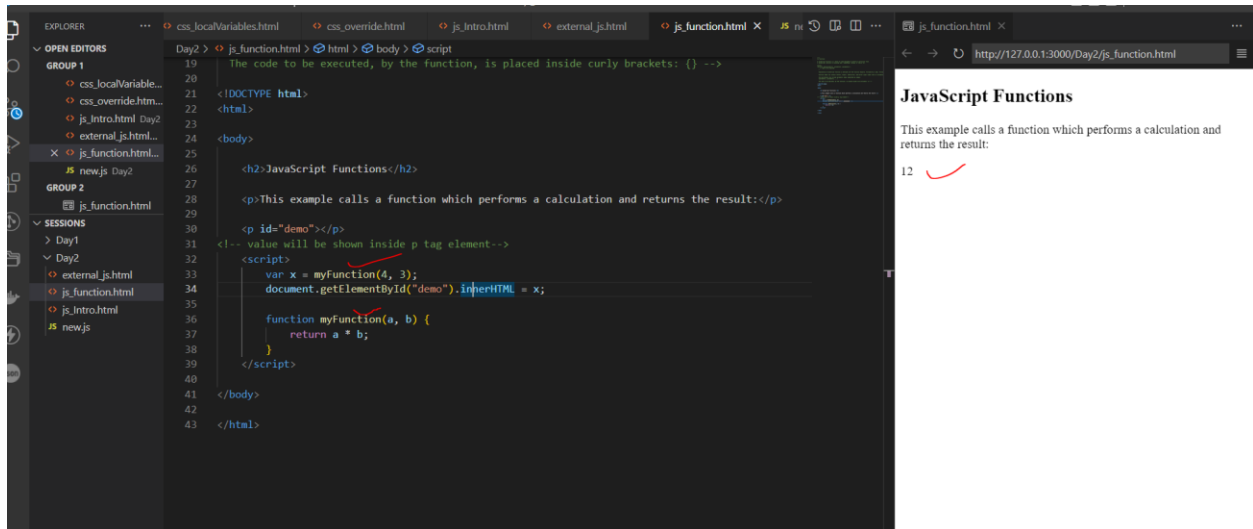
```
4 <head>
5 <title>Internal JS</title>
6 </head>
7
8 <body>
9   <!-- External Js -->
10
11   <!-- Scripts can also be placed in external files:
12   External scripts are practical when the same code is used in many different web pages.
13
14   JavaScript files have the file extension .js.
15
16   To use an external script, put the name of the script file in the src (source) attribute of a <script> tag: -->
17
18   <script src="new.js"></script>
19 </body>
20 </html>
```

OUTPUT: check outputs in browser console.



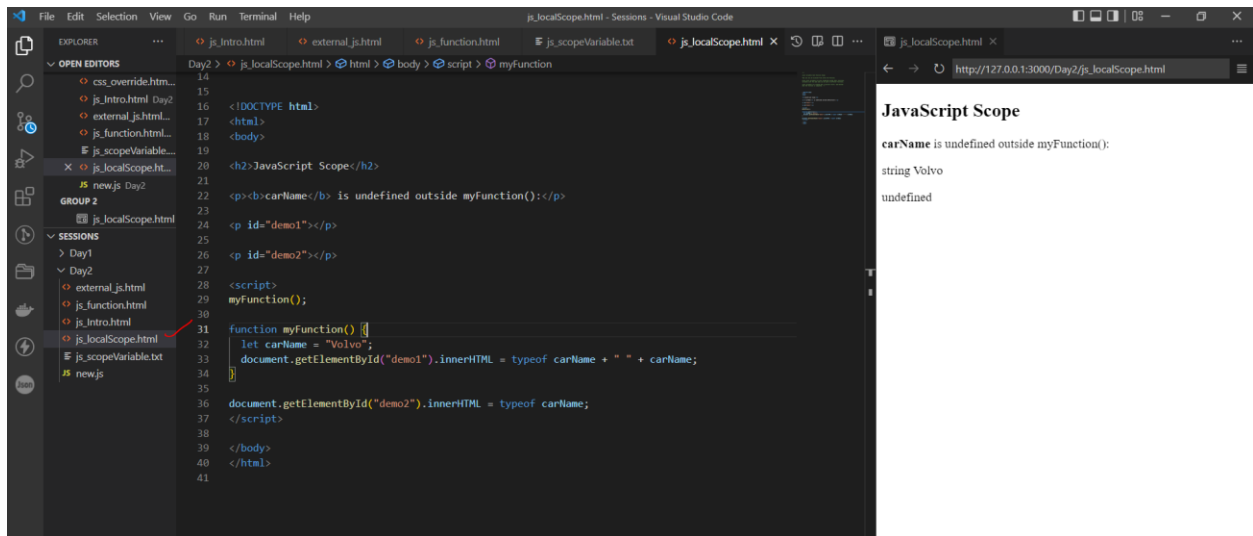
Functions

Code with outputs:



SCOPE:

LOCAL SCOPE:



GLOBAL SCOPE:

The screenshot shows a Visual Studio Code editor with a file named `js_globalScope.html`. The code defines a function `myFunction()` that declares a variable `carName` with the value "Volvo". Outside the function, the code attempts to access `carName` and displays its value in the browser. The browser preview on the right shows the text "I can display Volvo".

```
14 <!DOCTYPE html>
15 <html>
16 <body>
17 <h2>JavaScript Scope</h2>
18
19 <p><b>carName</b> is undefined outside myFunction():</p>
20
21 <p id="demo1"></p>
22
23 <p id="demo2"></p>
24
25 <script>
26 myFunction();
27
28 function myFunction() {
29   let carName = "Volvo";
30   document.getElementById("demo1").innerHTML = typeof carName + " " + carName;
31 }
32
33 document.getElementById("demo2").innerHTML = typeof carName;
34 </script>
35 </body>
36 </html>
```

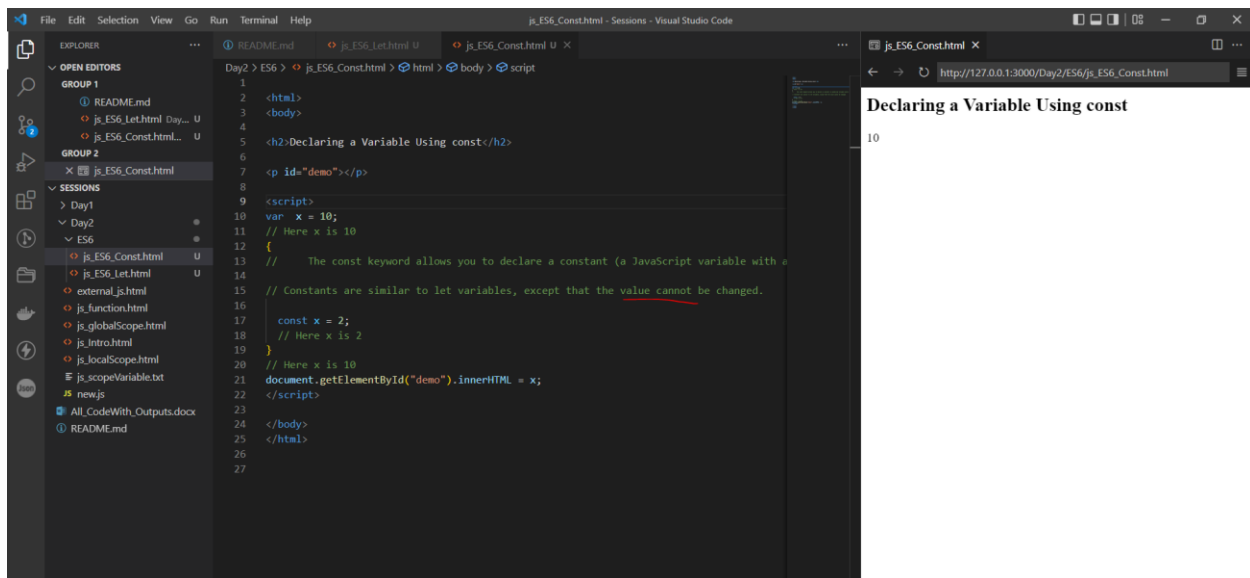
ES6 STANDARDS:

let:

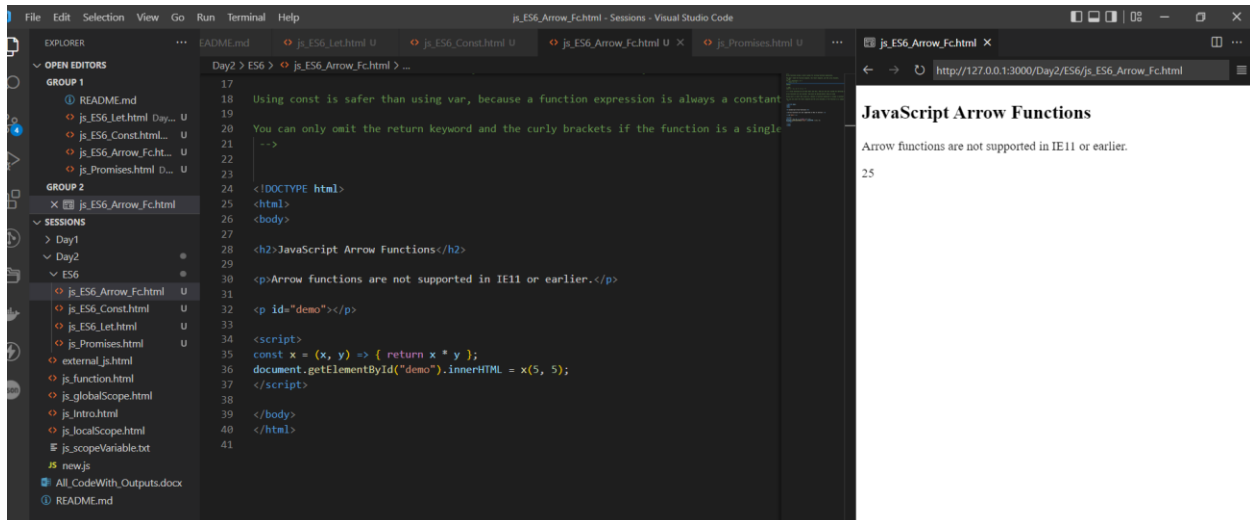
The screenshot shows a Visual Studio Code editor with a file named `js_ES6_Var_Let_Const.html`. The code demonstrates the use of the `let` keyword for variable declaration and redeclaration within a block scope. The browser preview on the right shows the text "Redeclaring a Variable Using let".

```
1 <!-- JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1996. -->
2
3 ECMAScript is the official name of the language.
4
5 ECMAScript versions have been abbreviated to ES1, ES2, ES3, ES5, and ES6.
6
7 Since 2016 new versions are named by year (ECMAScript 2016 / 2017 / 2018). -->
8
9 <!DOCTYPE html>
10 <html>
11 <body>
12
13 <h2>Redeclaring a Variable Using let</h2>
14
15 <p id="demo"></p>
16
17 <script>
18 // The let keyword allows you to declare a variable with block scope.
19 let x = 10;
20 // Here x is 10
21
22 {
23   let x = 2;
24   // Here x is 2
25 }
26
27 // Here x is 10
28 document.getElementById("demo").innerHTML = x;
29 </script>
30 </body>
31 </html>
```

Const:



Arrow Function:



Promises:

The screenshot shows a Visual Studio Code editor with a file explorer on the left and a code editor in the center. The file explorer shows a project structure with files like `README.md`, `js_ES6_Let.html`, `js_ES6_Const.html`, `js_ES6_Arrow_Fc.html`, and `js_Promises.html`. The code editor displays the content of `js_Promises.html`, which includes HTML and JavaScript code. The JavaScript code defines a Promise that resolves after 3000 milliseconds with the value "I love You !!". The browser output on the right shows the rendered HTML, which includes the text "JavaScript Promise" and "I love You !!".

```
Day2 > ES6 > js_Promises.html > html > body > script
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <h2>JavaScript Promise</h2>
6
7 <p>Wait 3 seconds (3000 milliseconds) for this page to change.</p>
8
9 <h1 id="demo"></h1>
10
11 <script>
12
13 // A Promise is a JavaScript object that links "Producing Code" and "Consuming Code".
14
15 // "Producing Code" can take some time and "Consuming Code" must wait for the result.
16
17 const myPromise = new Promise(function(myResolve, myReject) {
18   | setTimeout(function() { myResolve("I love You !!"); }, 3000);
19   | });
20
21 myPromise.then(function(value) {
22   | document.getElementById("demo").innerHTML = value;
23   | });
24 </script>
25
26 </body>
27 </html>
28
```

JavaScript Promise

Wait 3 seconds (3000 milliseconds) for this page to change.

I love You !!

JS CLASSES:

JS class syntax:

The screenshot shows a Visual Studio Code editor with a file explorer on the left and a code editor in the center. The file explorer shows a project structure with files like `README.md`, `js_ES6_Let.html`, `js_ES6_Const.html`, `js_ES6_Arrow_Fc.html`, and `js_Classes.html`. The code editor displays the content of `js_Classes.html`, which includes HTML and JavaScript code. The JavaScript code defines a class `Car` with a constructor that takes `name` and `year` as arguments. The browser output on the right shows the rendered HTML, which includes the text "JavaScript Class" and "How to use a JavaScript Class.".

```
Day2 > JS_CLASSES > js_Classes.html > html > body
1 <!-- JavaScript Class Syntax
2 Use the keyword class to create a class.
3
4 Always add a method named constructor(): -->
5
6 <!-- Syntax
7 class ClassName {
8   | constructor() { ... }
9   | -->
10
11
12 <!DOCTYPE html>
13 <html>
14 <body>
15
16 <h2>JavaScript Class</h2>
17
18 <p>How to use a JavaScript Class.</p>
19
20 <p id="demo"></p>
21
22 <script>
23 class Car {
24   | constructor(name, year) {
25   |   this.name = name;
26   |   this.year = year;
27   | }
28
29
30 const myCar = new Car("Ford", 2014);
31 document.getElementById("demo").innerHTML =
32   myCar.name + " " + myCar.year;
33 </script>
34
35 </body>
36 </html>

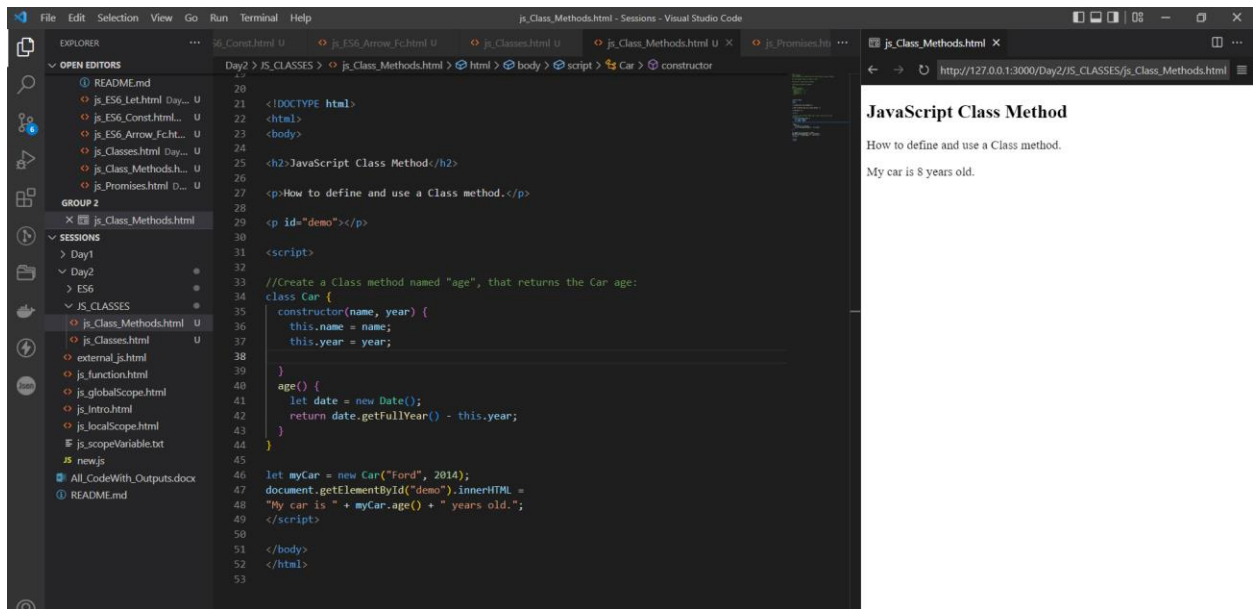
```

JavaScript Class

How to use a JavaScript Class.

Ford 2014

Js Class Methods:



JavaScript Event Handlers

Event handlers can be used to handle and verify user input, user actions, and browser actions:

- Things that should be done every time a page loads
- Things that should be done when the page is closed
- Action that should be performed when a user clicks a button
- Content that should be verified when a user inputs data
- And more ...

Many different methods can be used to let JavaScript work with events:

- HTML event attributes can execute JavaScript code directly
- HTML event attributes can call JavaScript functions
- You can assign your own event handler functions to HTML elements
- You can prevent events from being sent or being handled
- And more ...

addEventListener:

remove:

When you click this button the top paragraph letters will be removed.

