Contents

[Day-1 1](#_Toc103260285)

[1.HTML ATTRIBUTES: 1](#_Toc103260286)

[2. HTML INPUT TAGS: 2](#_Toc103260287)

[CSS: 3](#_Toc103260288)

[INLINE CSS: 3](#_Toc103260289)

[INTERNAL CSS: 4](#_Toc103260290)

[EXTERNAL CSS: 5](#_Toc103260291)

[CSS SELECTORS: 6](#_Toc103260292)

[CSS ID SELECTORS: 7](#_Toc103260293)

[CLASS SELECTOR: 8](#_Toc103260294)

[UNIVERSAL SELECTOR: 9](#_Toc103260295)

[CSS VARIABLES 10](#_Toc103260296)

[Local Variables: 10](#_Toc103260297)

[Override/Global Variables: 10](#_Toc103260298)

[DAY-2 11](#_Toc103260299)

[Internal Js: 11](#_Toc103260300)

[External Js: 12](#_Toc103260301)

[Functions 13](#_Toc103260302)

[SCOPE: 13](#_Toc103260303)

[LOCAL SCOPE: 13](#_Toc103260304)

[GLOBAL SCOPE: 14](#_Toc103260305)

[ES6 STANDARDS: 14](#_Toc103260306)

[let: 14](#_Toc103260307)

[Const: 15](#_Toc103260308)

[Arrow Function: 15](#_Toc103260309)

[Promises: 16](#_Toc103260310)

[JS CLASSES: 16](#_Toc103260311)

[JS class syntax: 16](#_Toc103260312)

[Js Class Methods: 17](#_Toc103260313)

[JavaScript Event Handlers 17](#_Toc103260314)

[addEventListner 18](#_Toc103260315)

[remove: 18](#_Toc103260316)

[DAY-3 18](#_Toc103260317)

[*What are the different types of event handlers provided by JavaScript?* 19](#_Toc103260318)

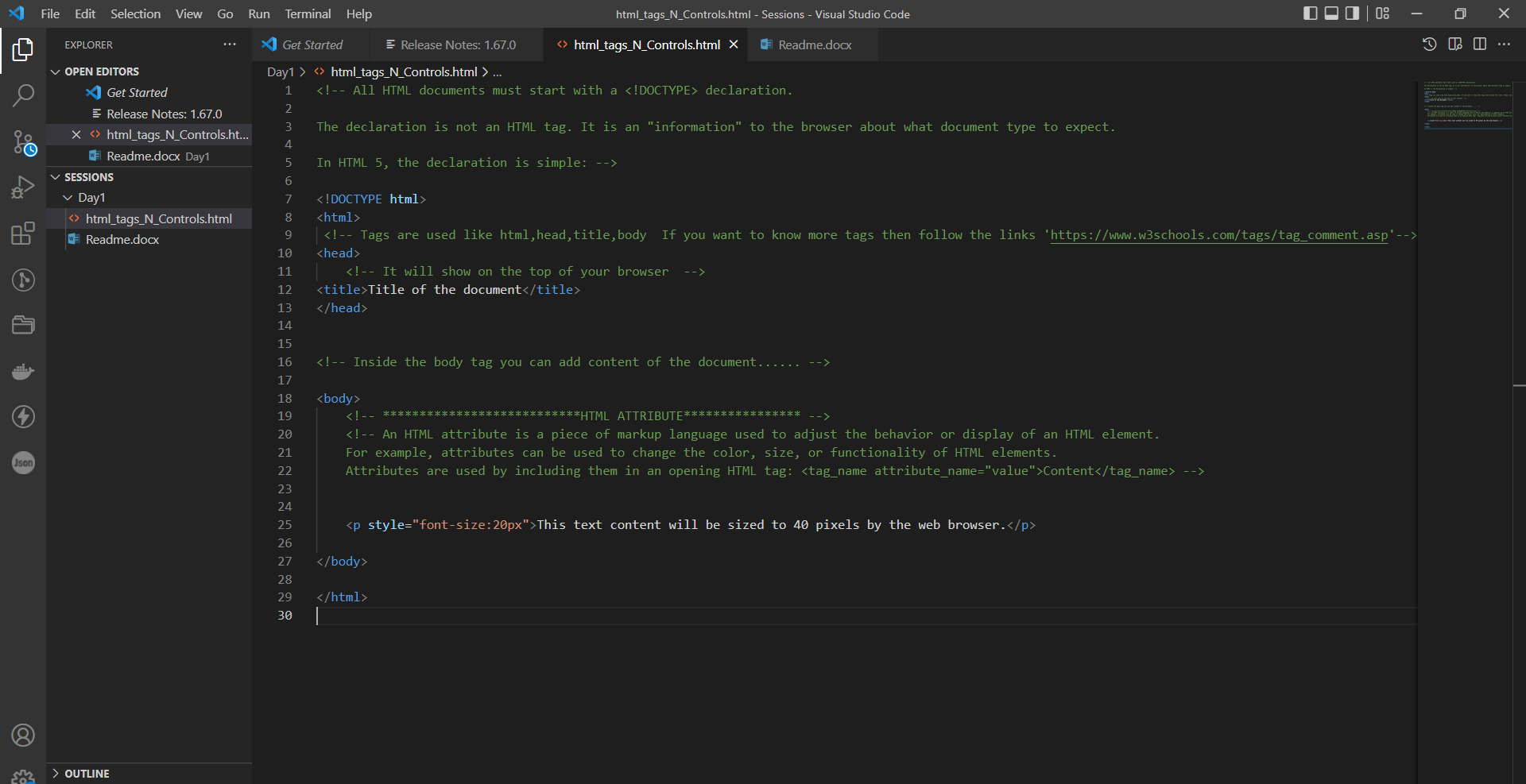
[Pass by Value: 20](#_Toc103260319)

[Pass by Reference: 22](#_Toc103260320)

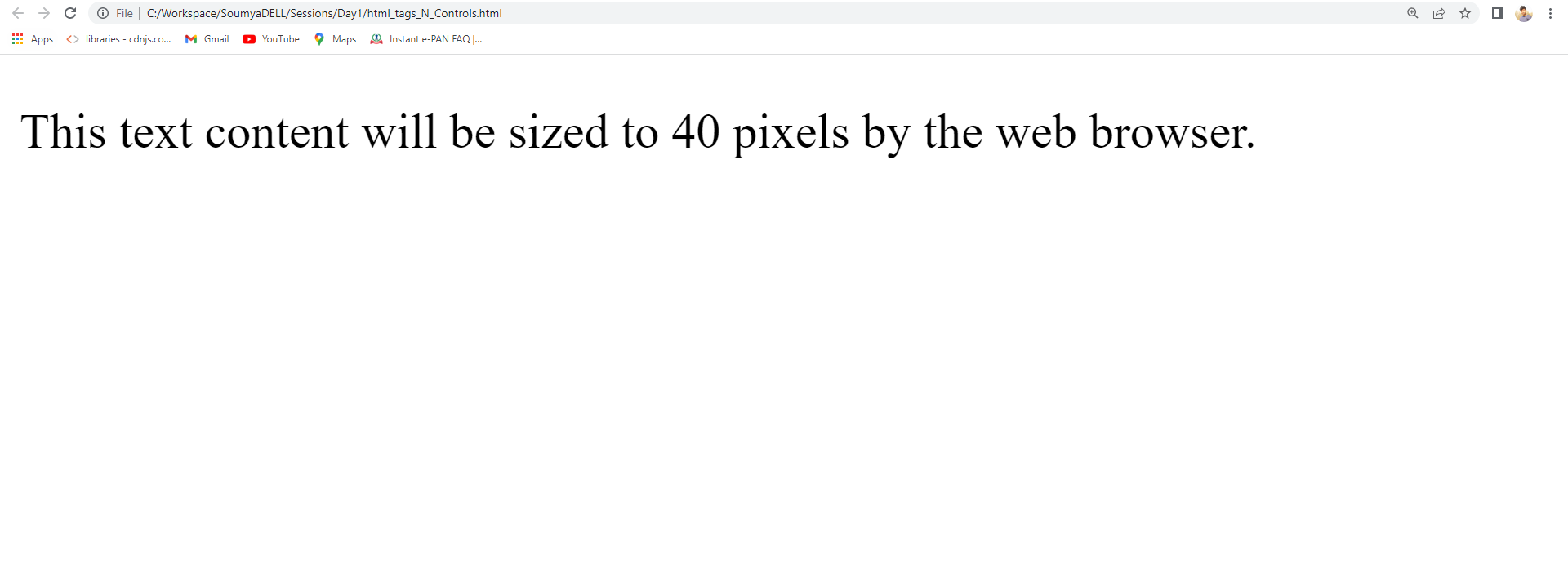
# Day-1

Code with Outputs:

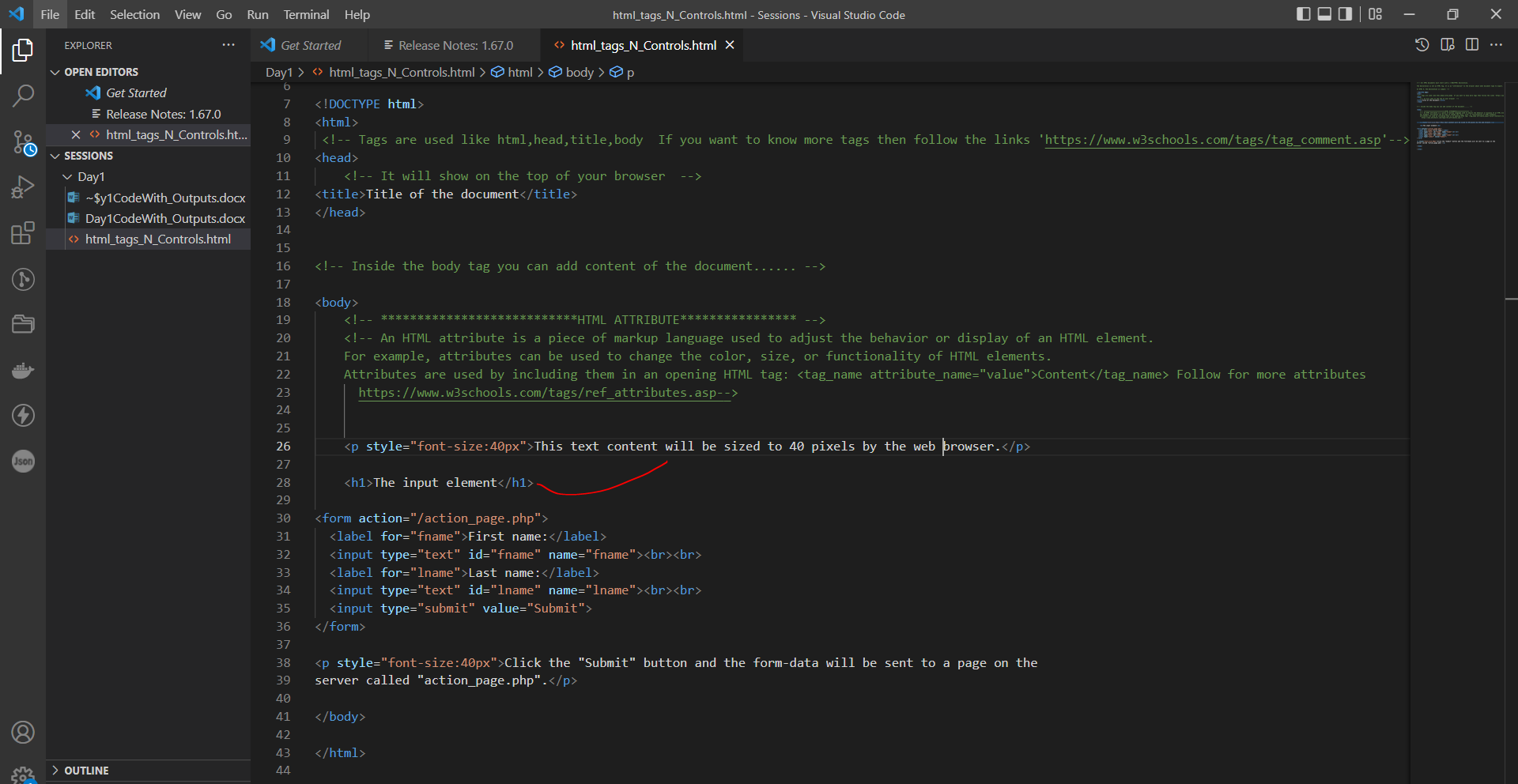
## 1.HTML ATTRIBUTES:



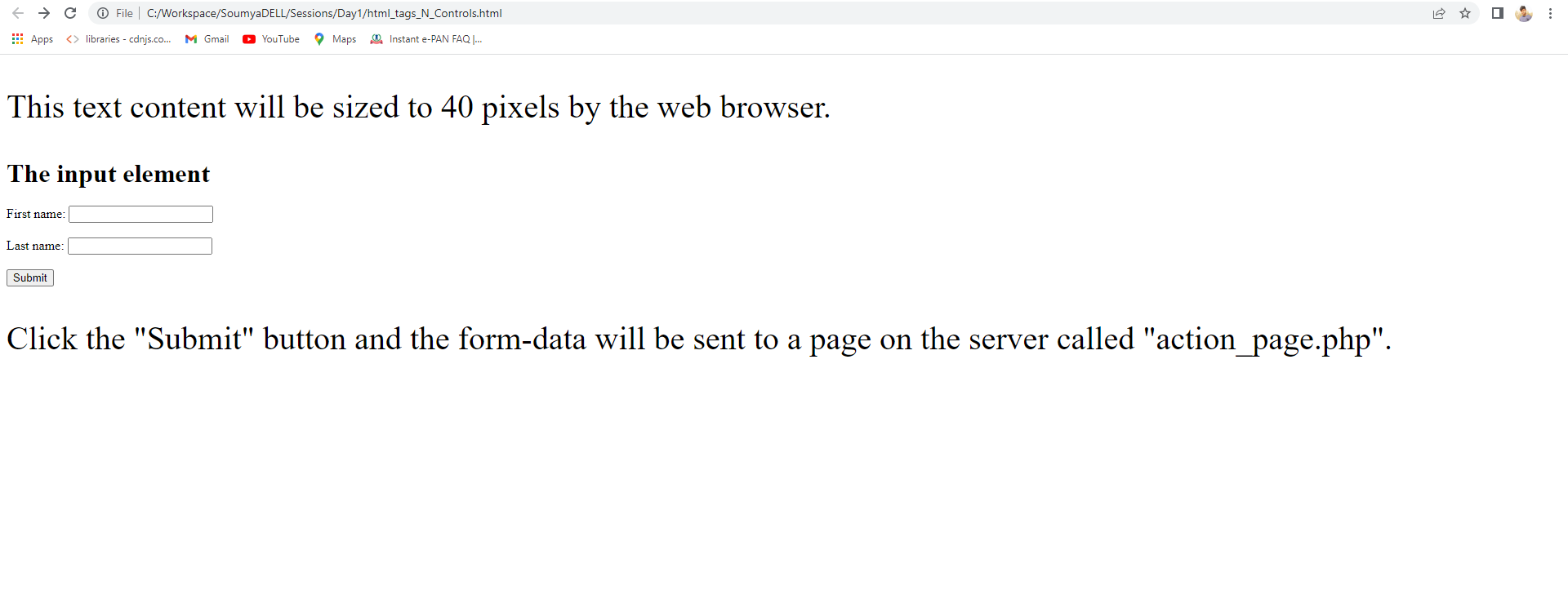
Output:



## 2. HTML INPUT TAGS:

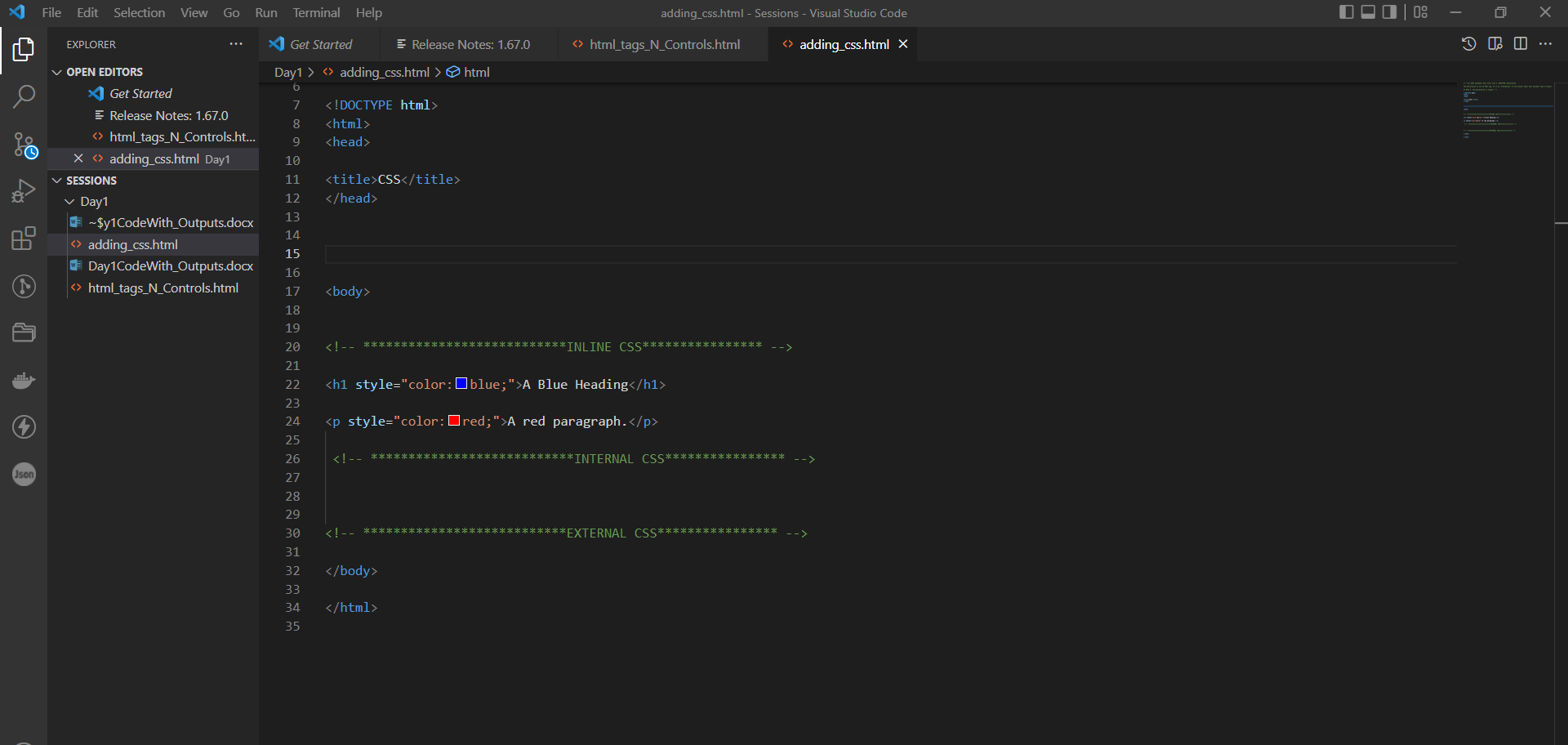


OUTPUT:

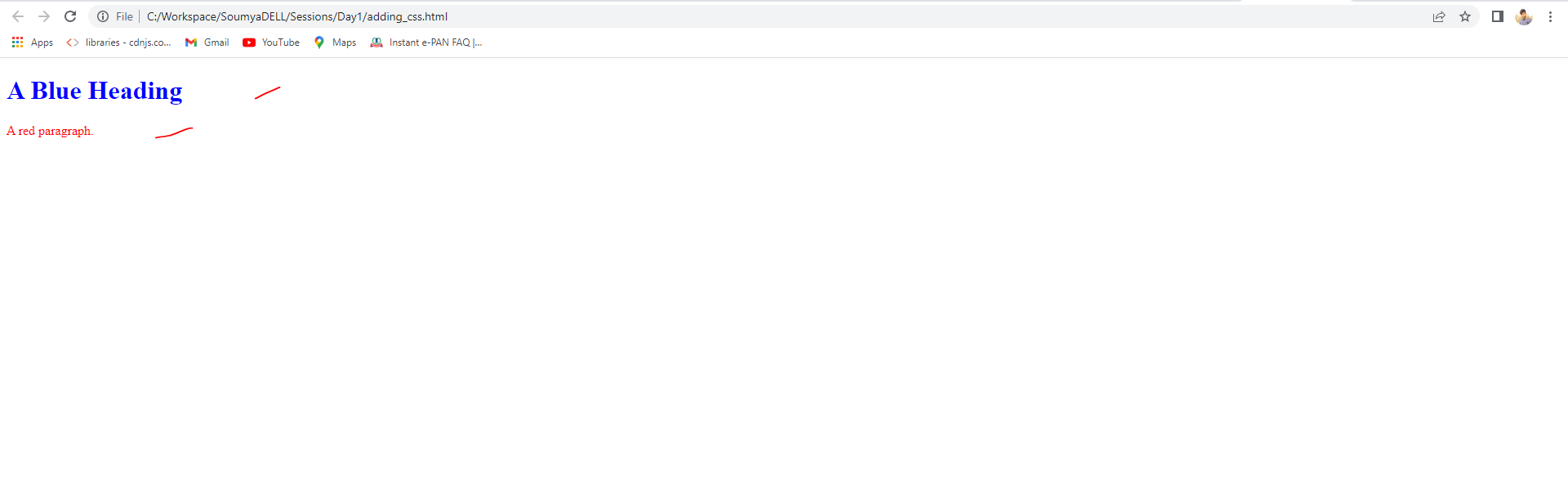


# CSS:

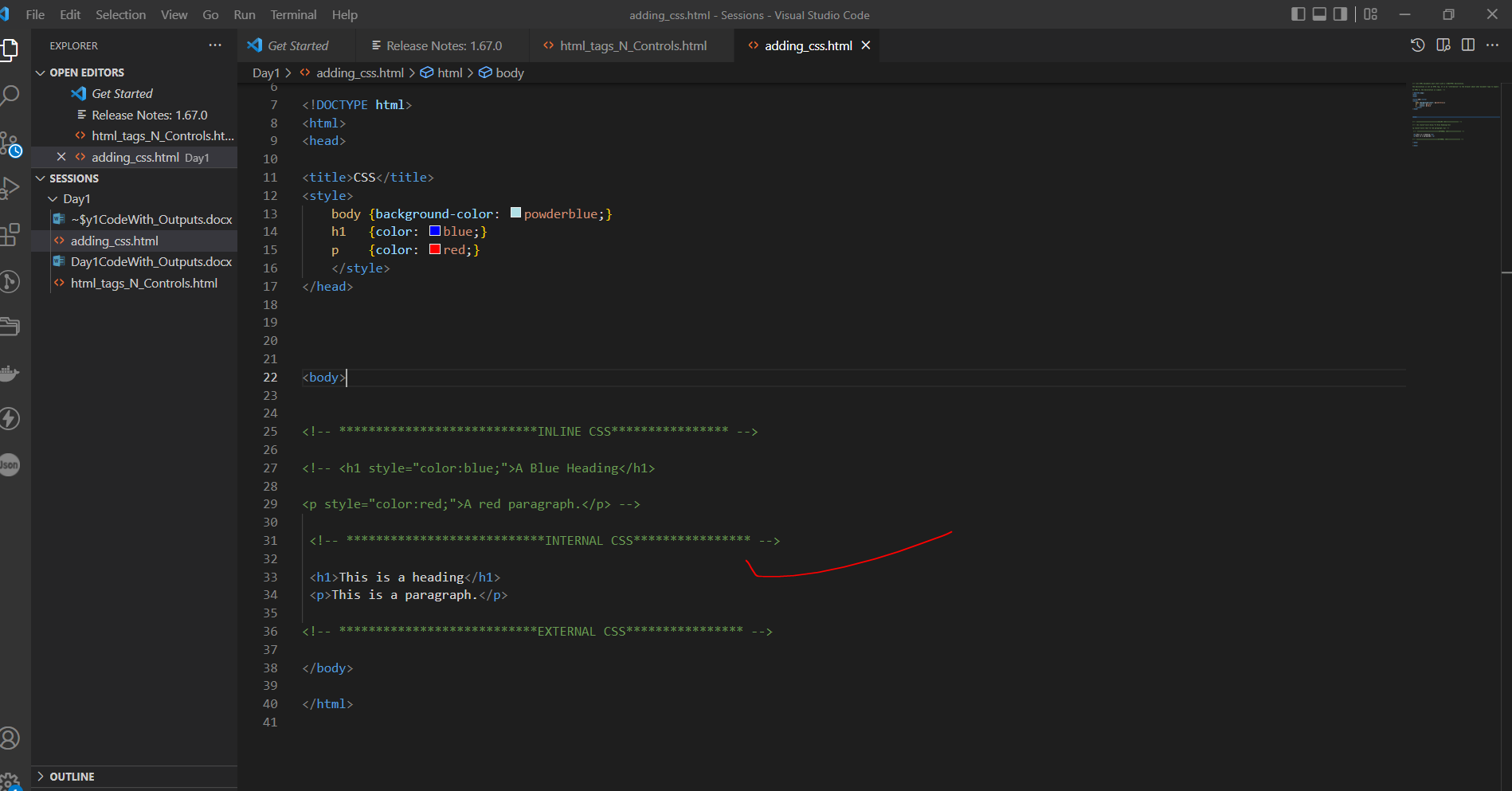
## INLINE CSS:



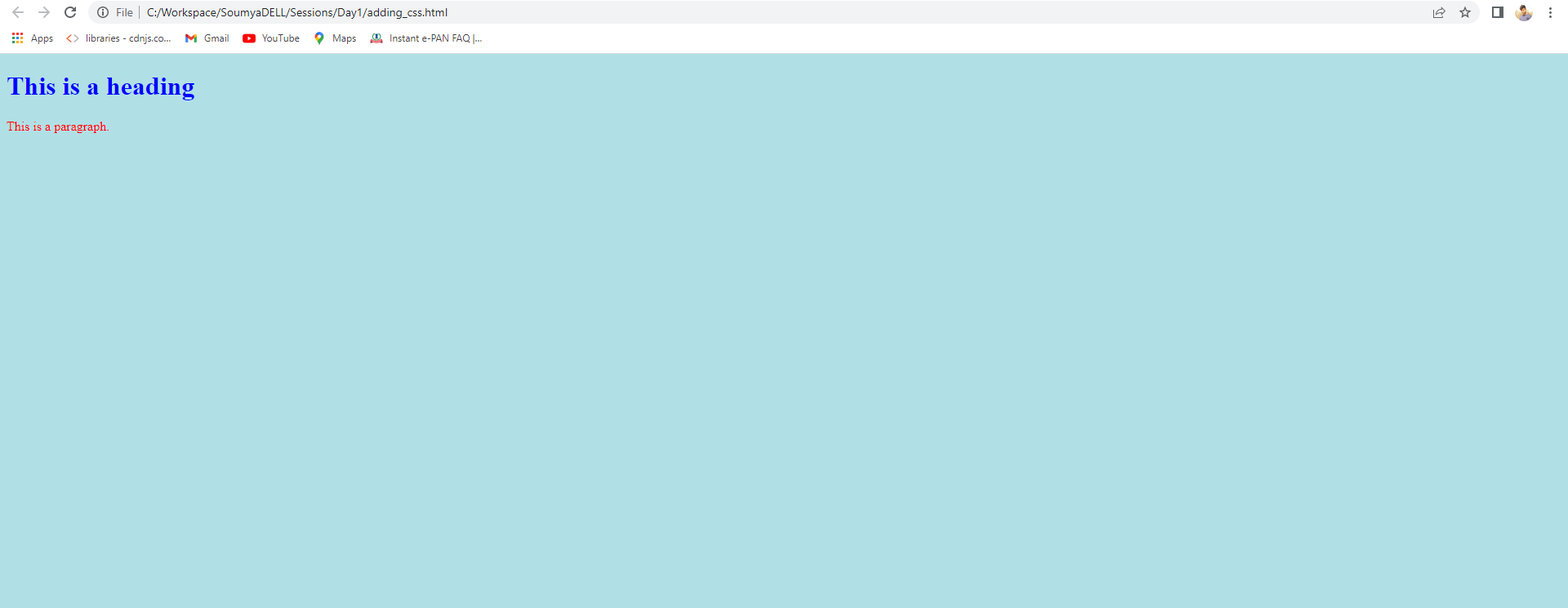
OUTPUT:



## INTERNAL CSS:

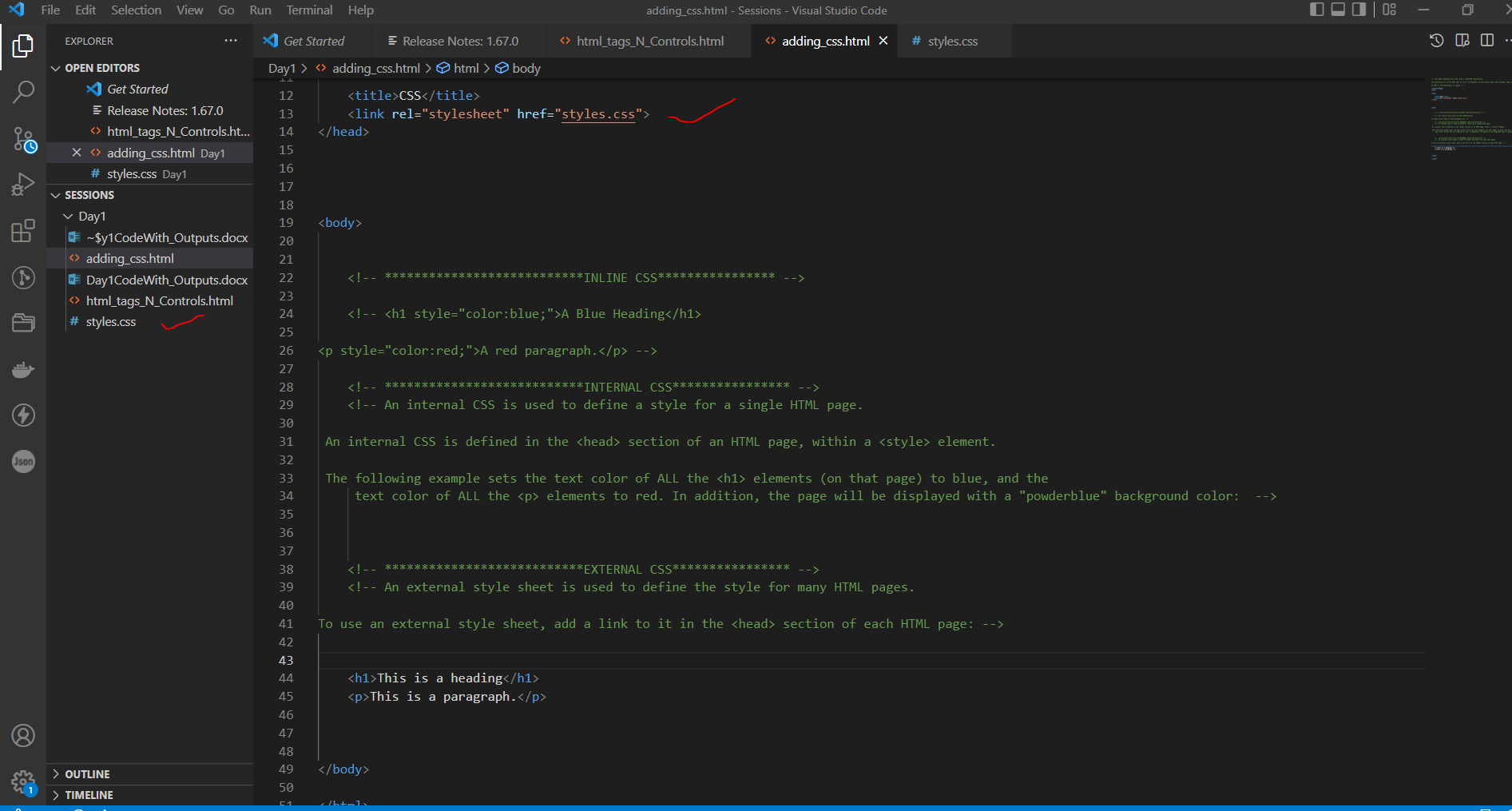


OUTPUT:

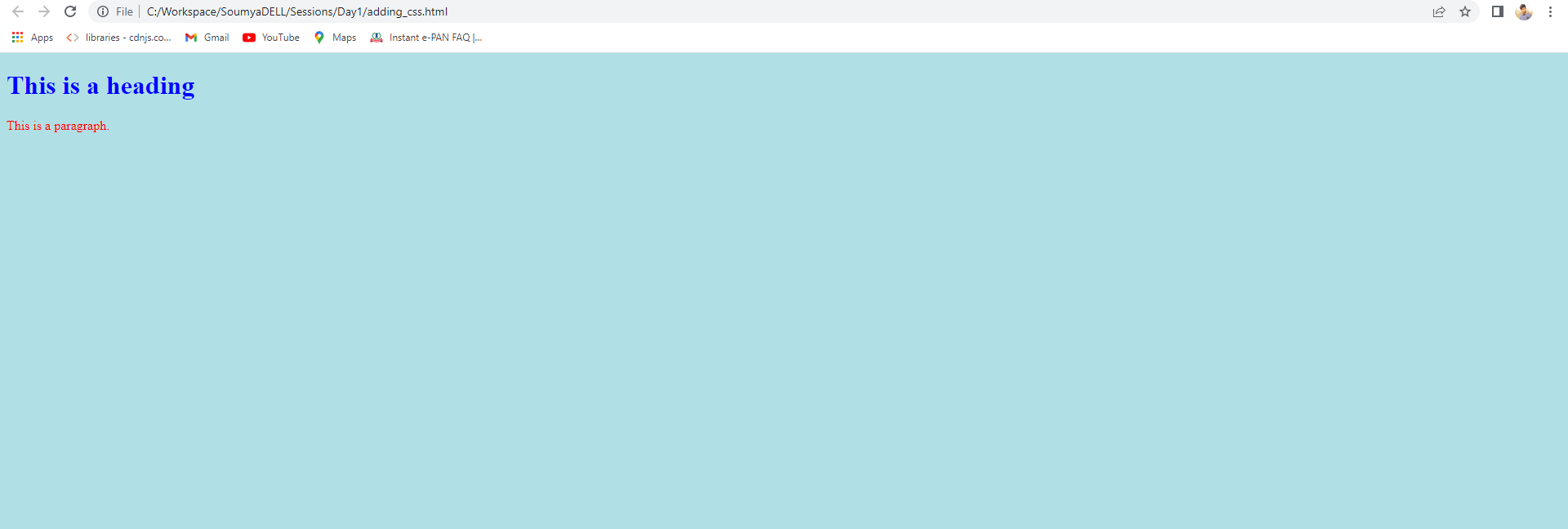


## EXTERNAL CSS:

Add new file (styles.css) with .css extension then link in html file.

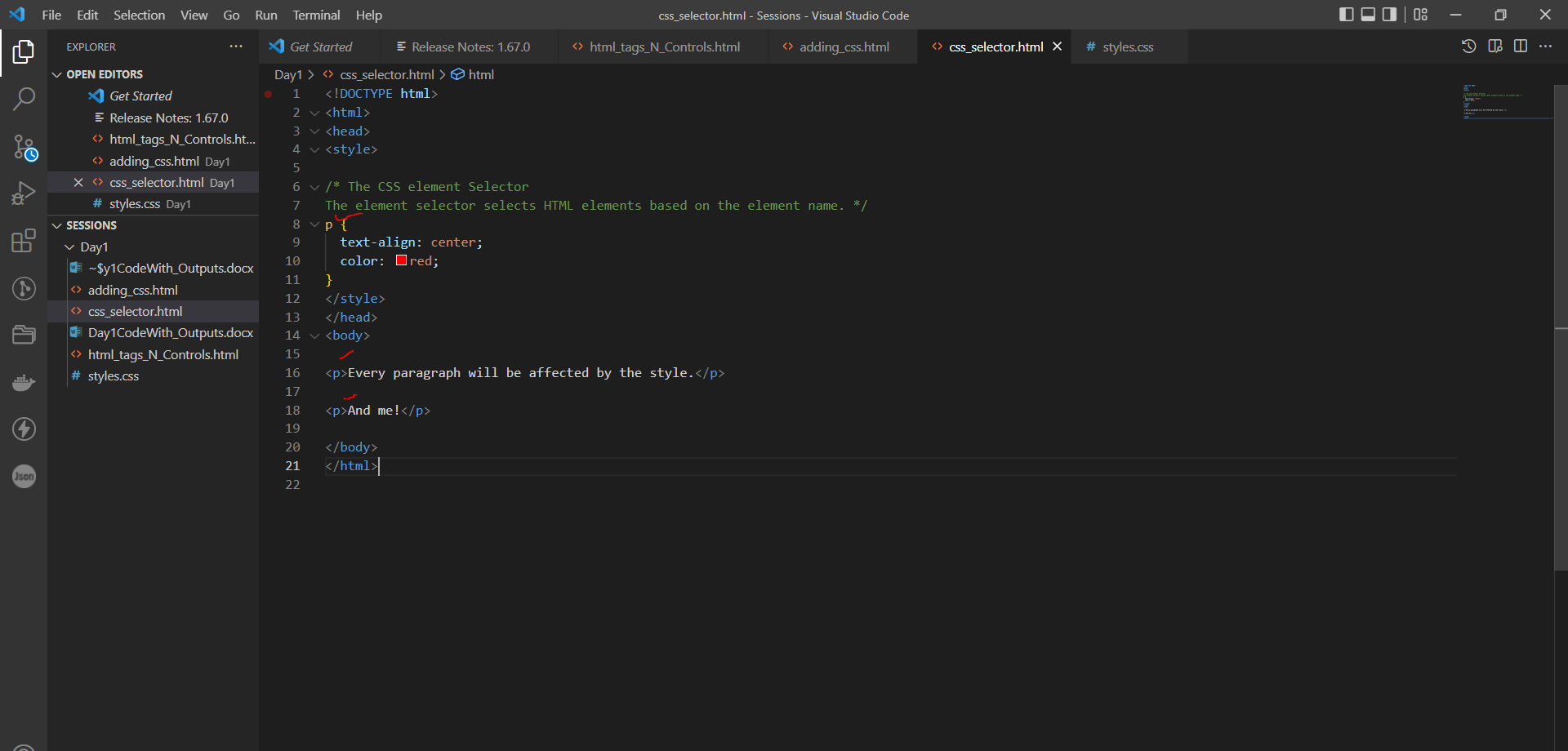


OUTPUT:

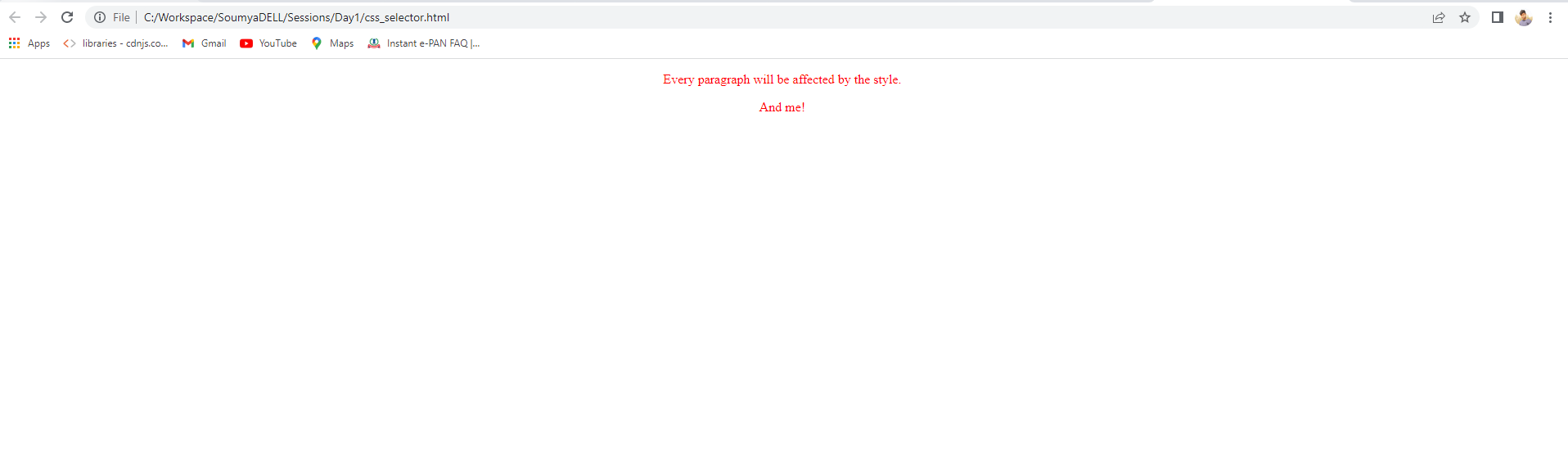


## CSS SELECTORS:

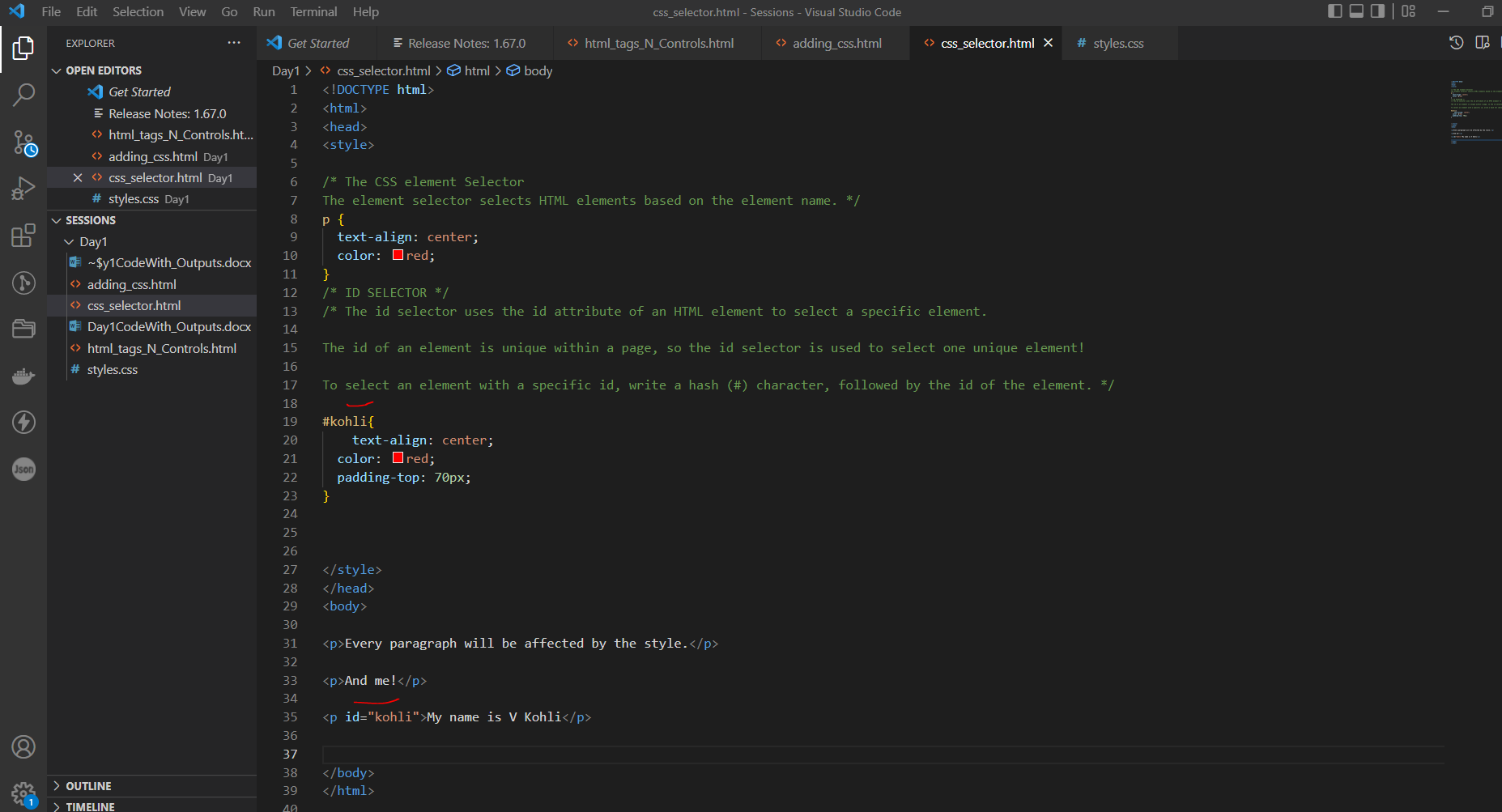
CSS ELEMENT SELECTOR:



OUTPUT:



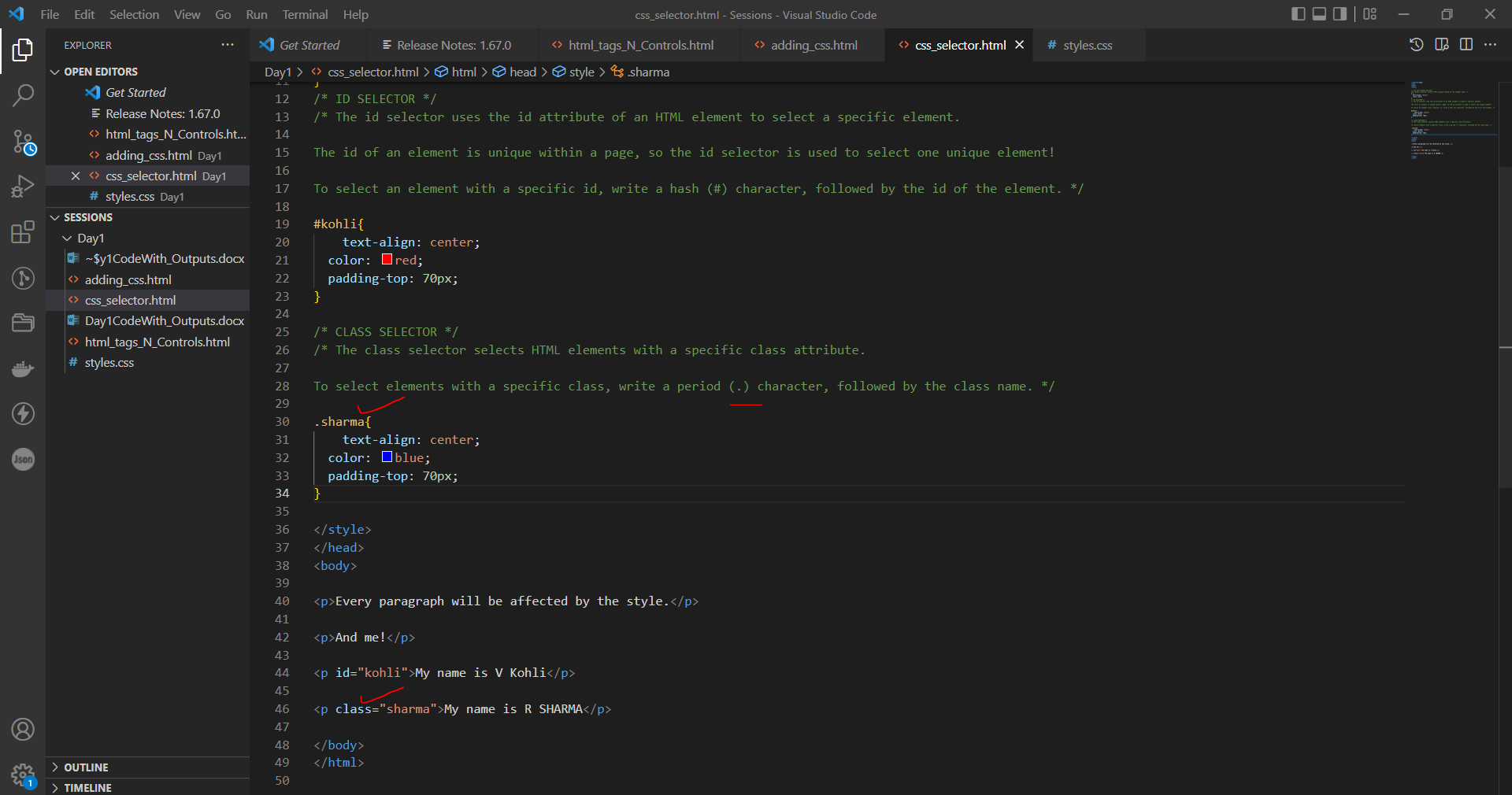
## CSS ID SELECTORS:



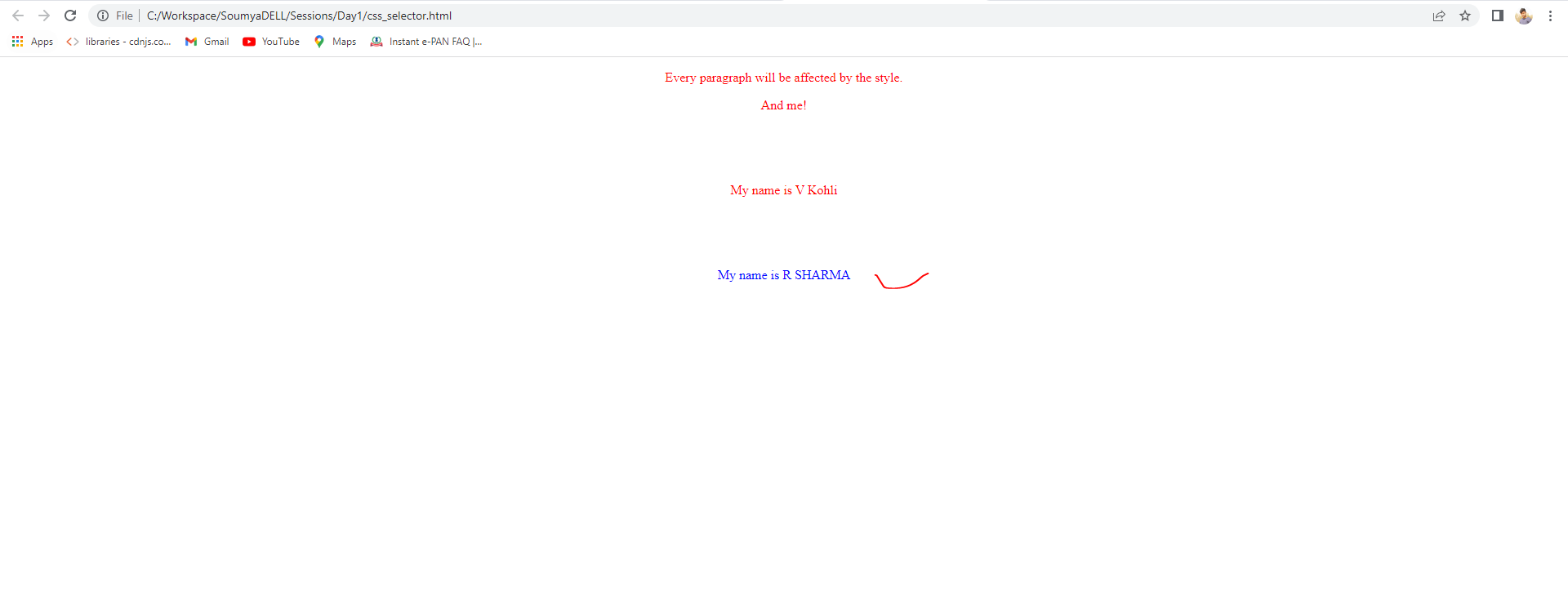
OUTPUT:



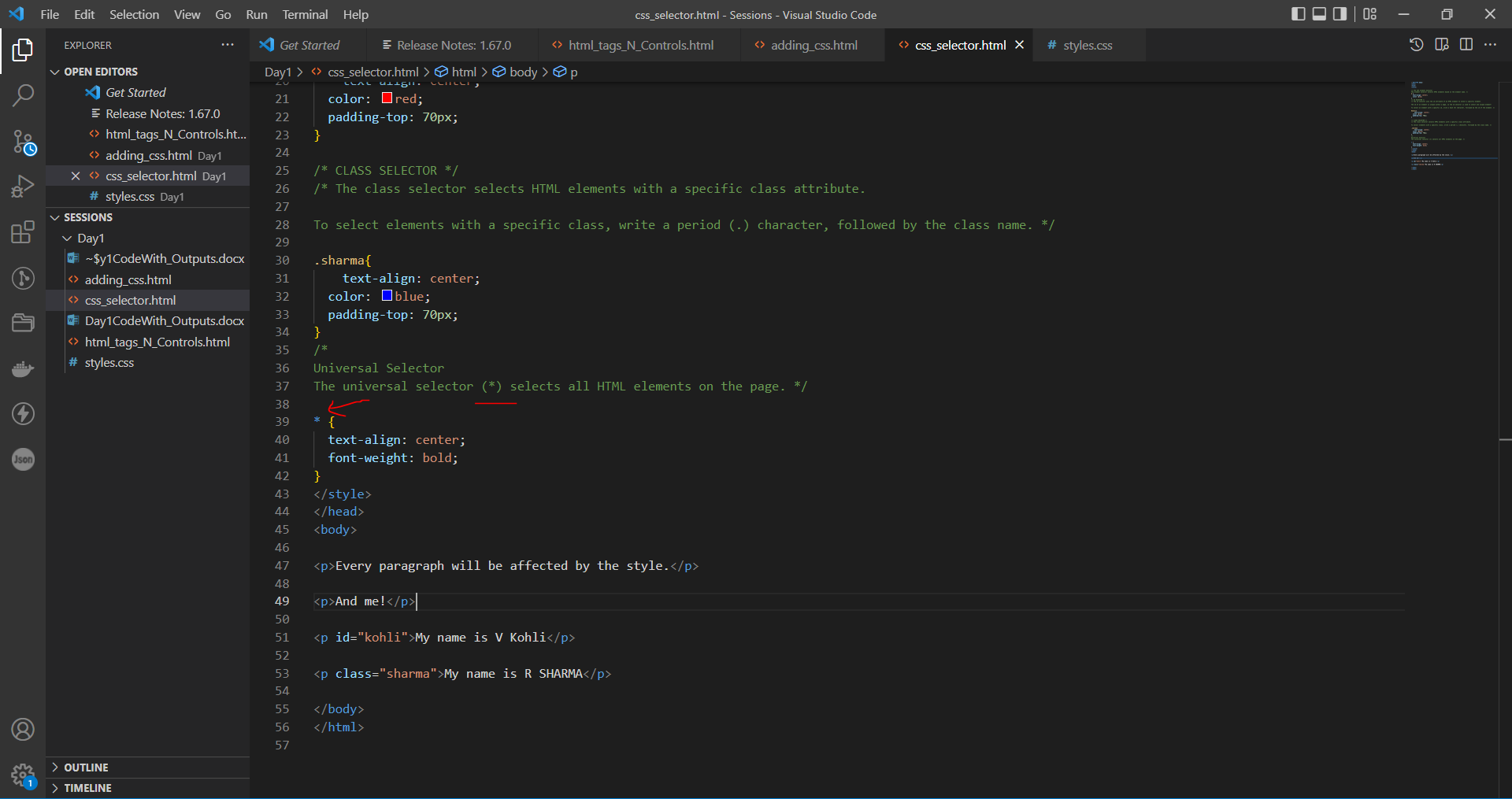
## CLASS SELECTOR:



OUTPUT:

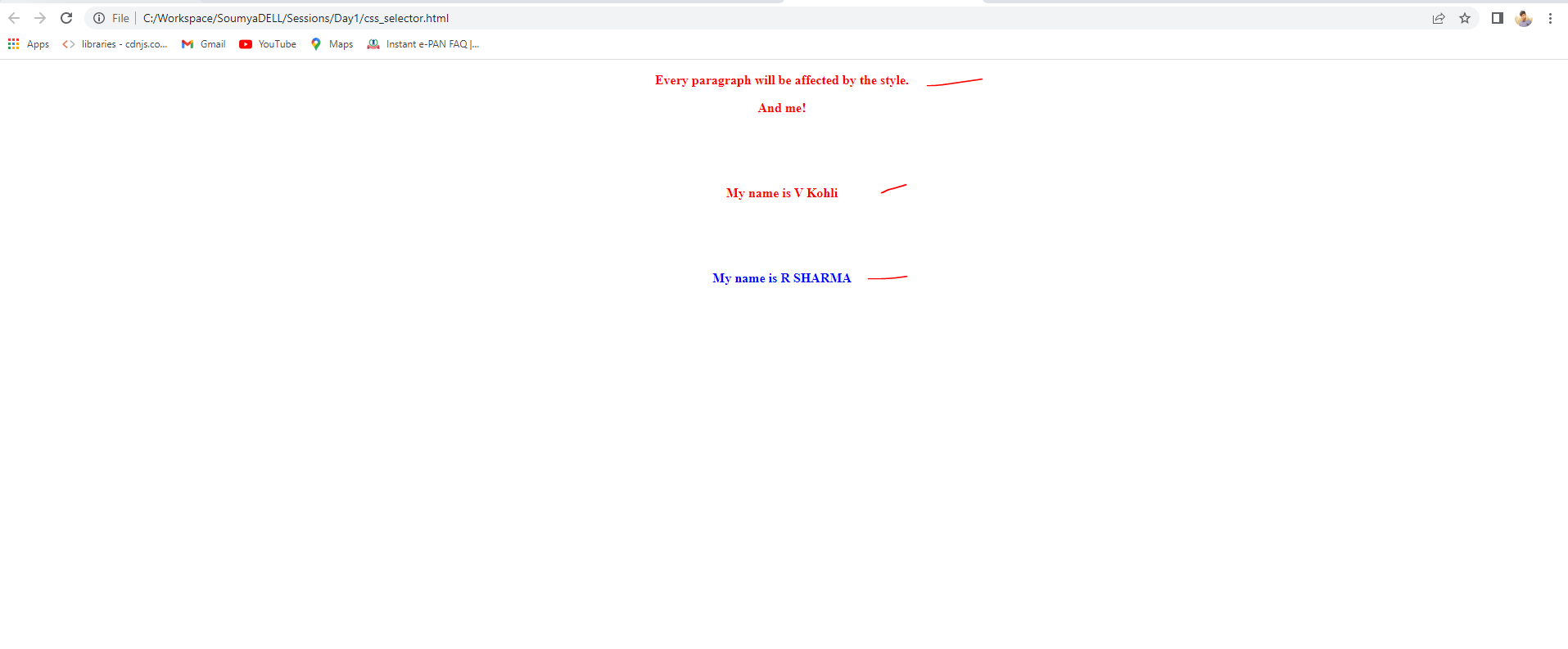


## UNIVERSAL SELECTOR:



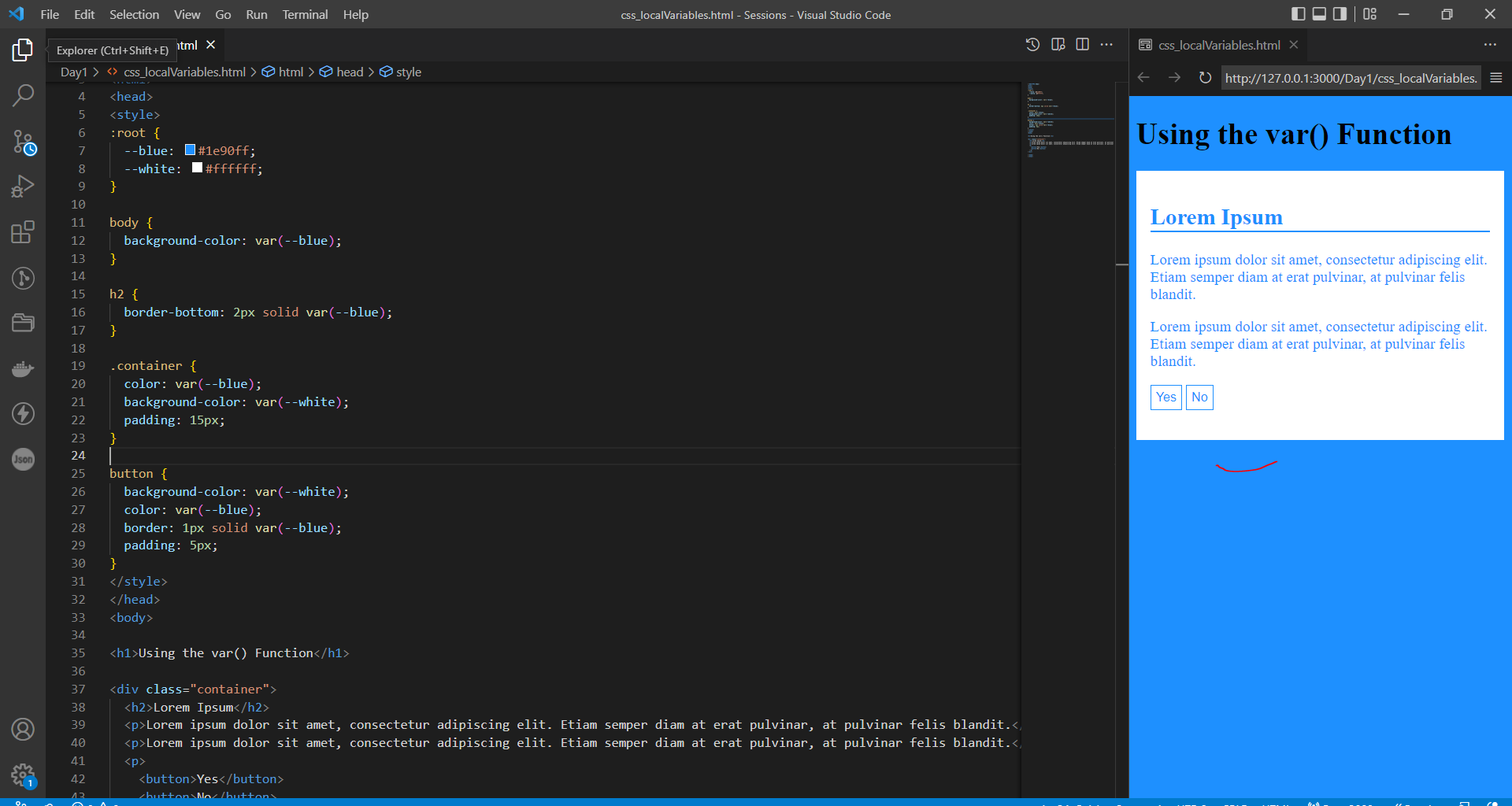
OUTPUT:

All the letters are changed to **BOLD** Find the differences with previous output images.

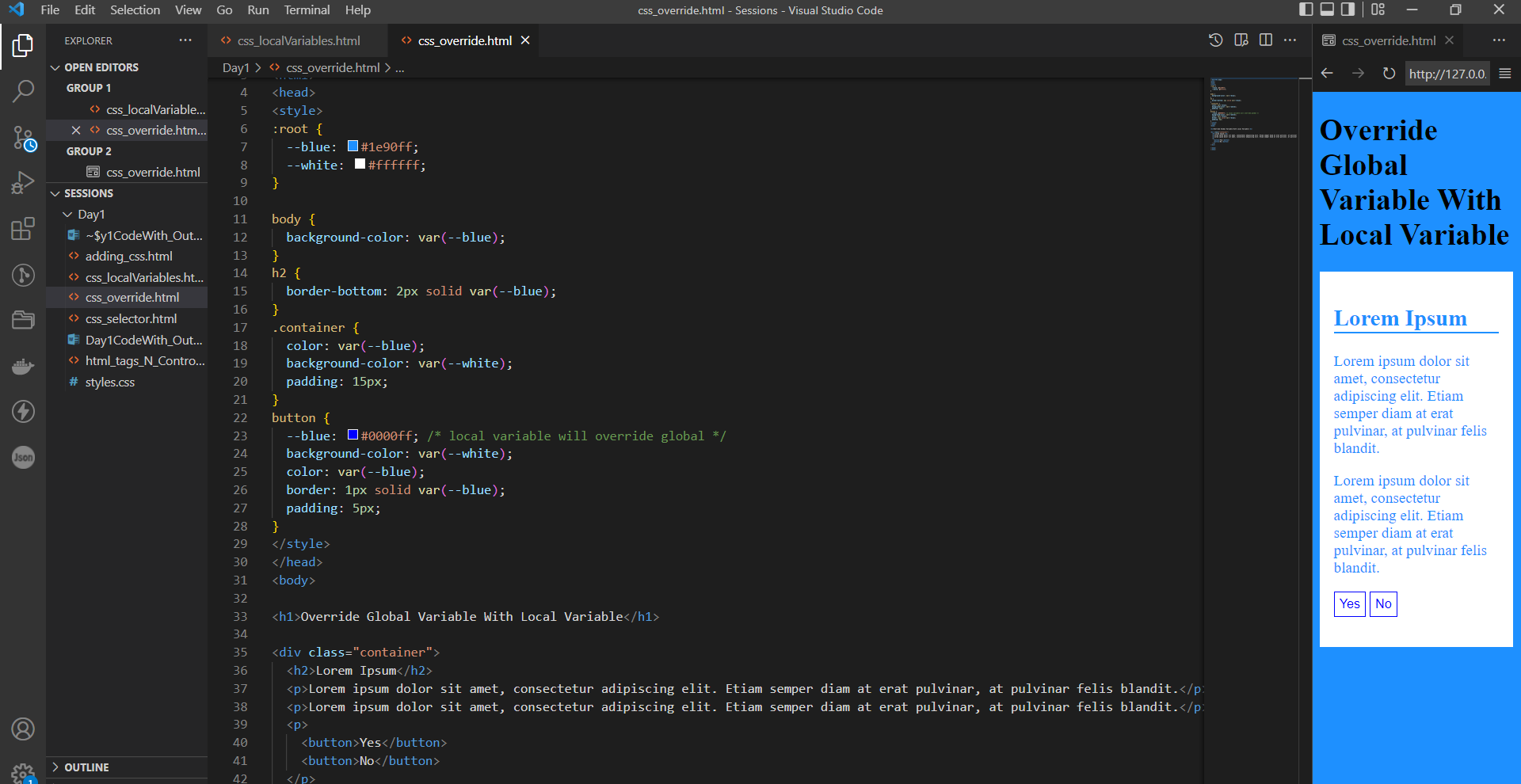


# CSS VARIABLES

## Local Variables:



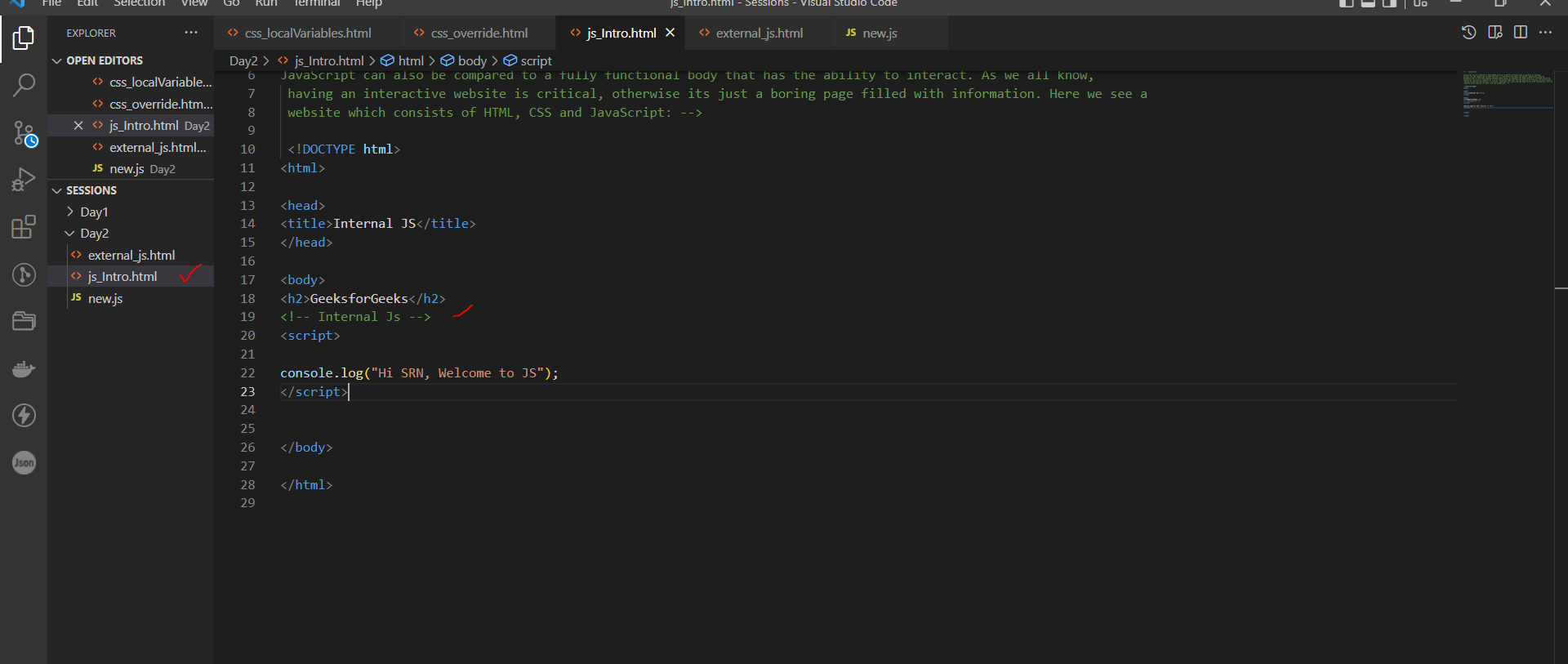
## Override/Global Variables:



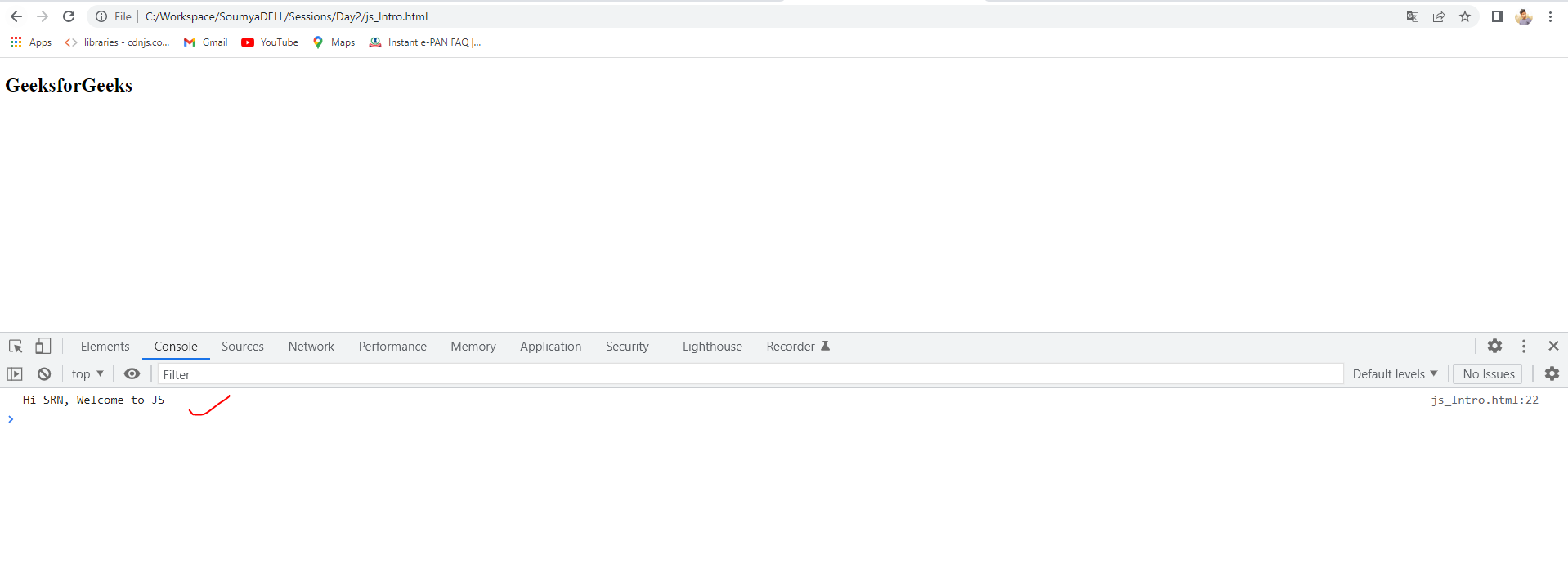
# DAY-2

**JavaScript :**

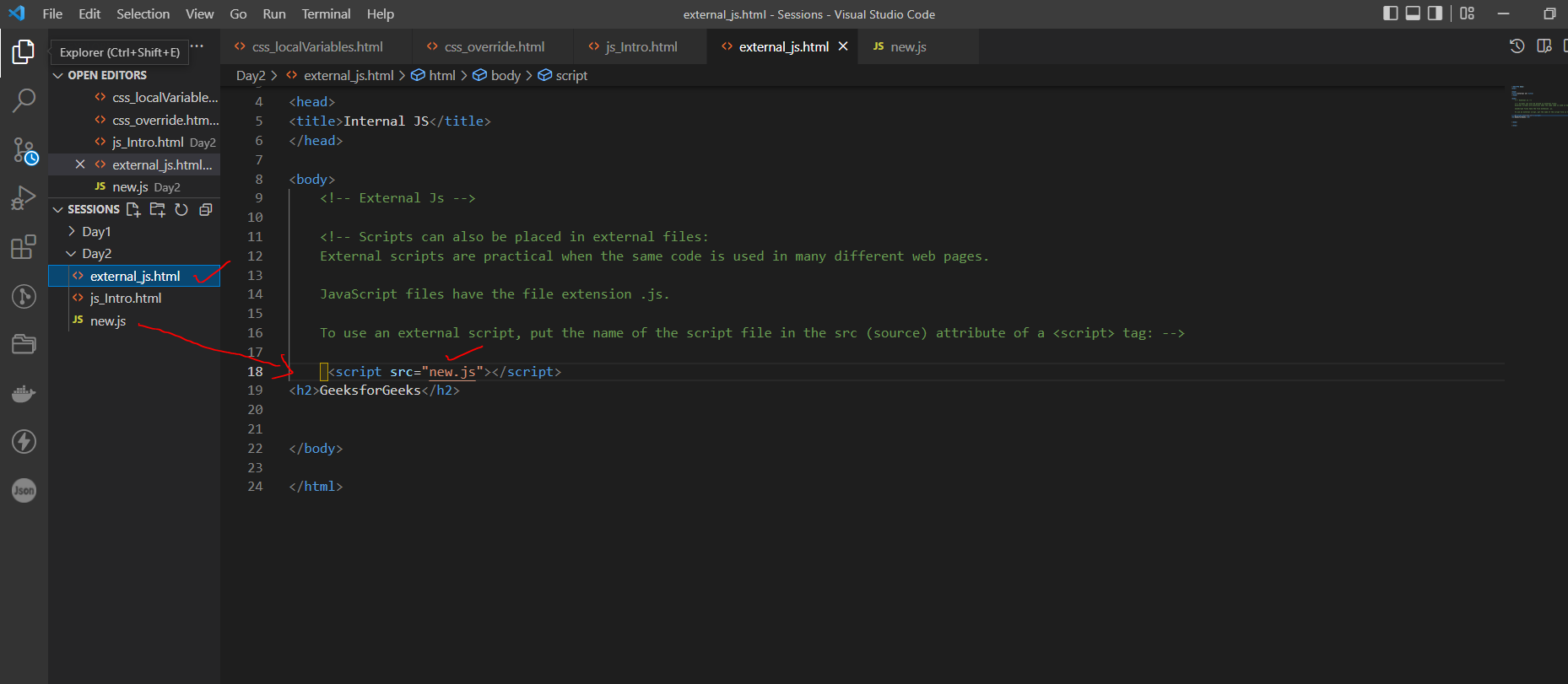
## Internal Js:



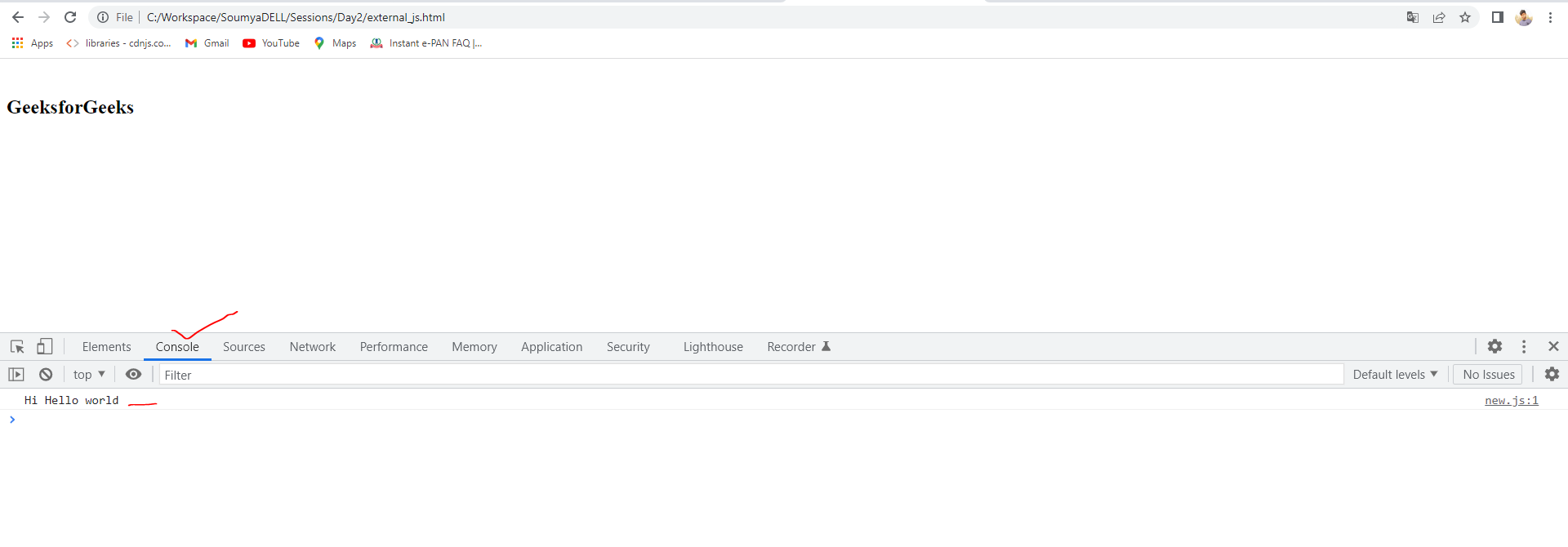
OUTPUT: check outputs in browser console.



## External Js:

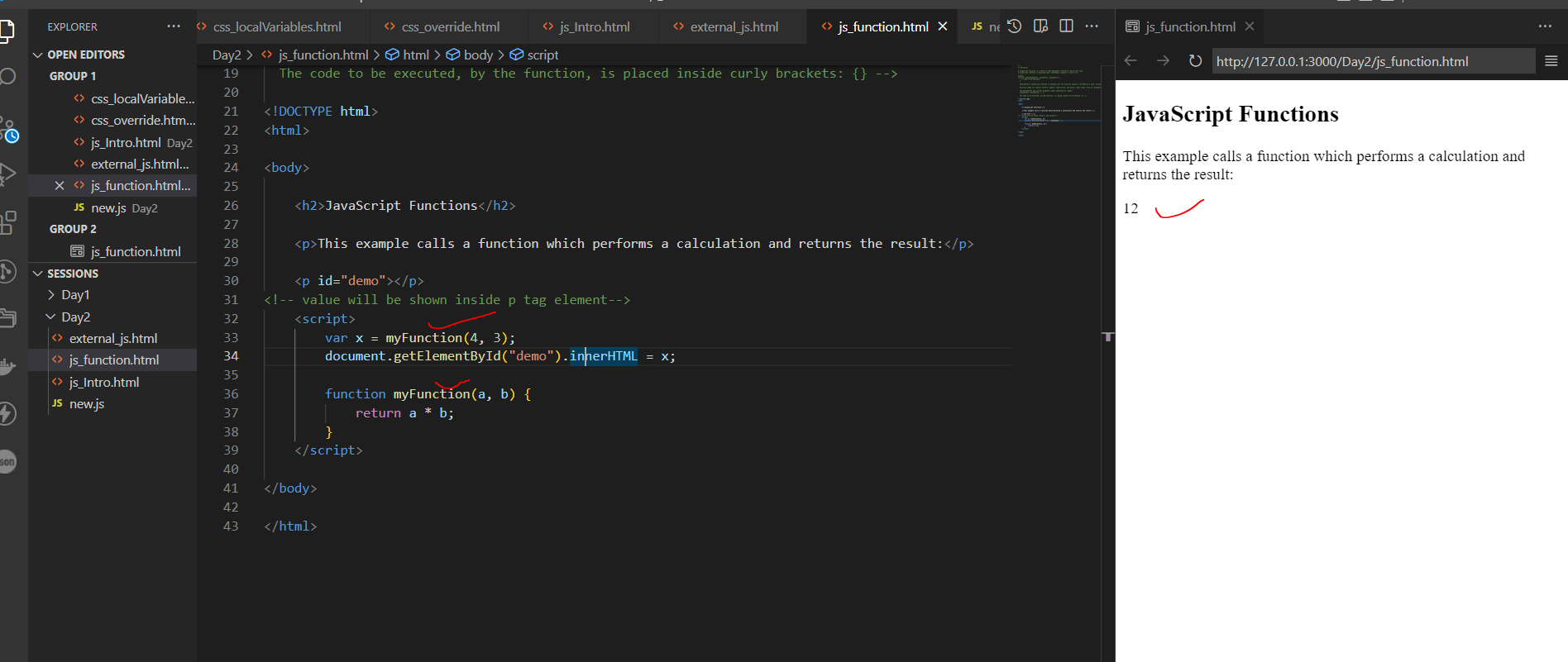


OUTPUT: check outputs in browser console.



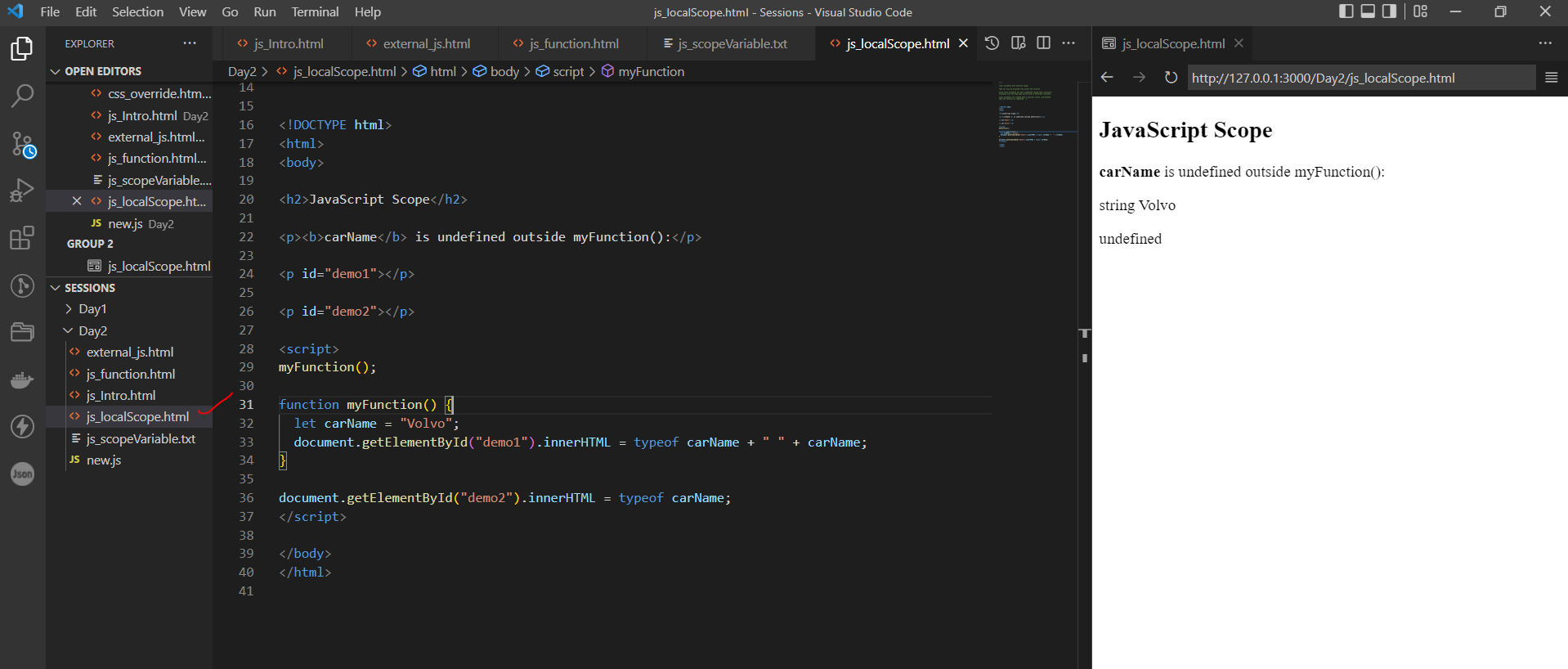
# Functions

Code with outputs:

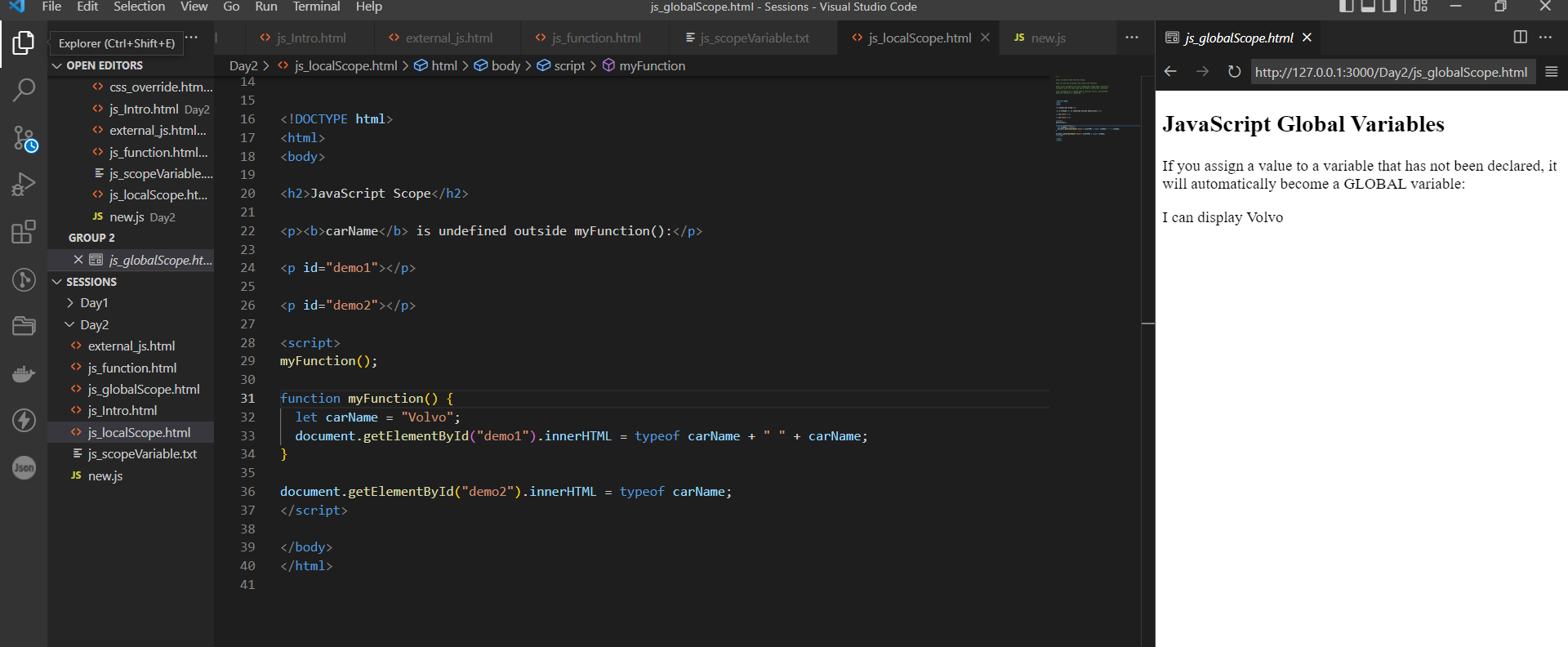


# SCOPE:

## LOCAL SCOPE:

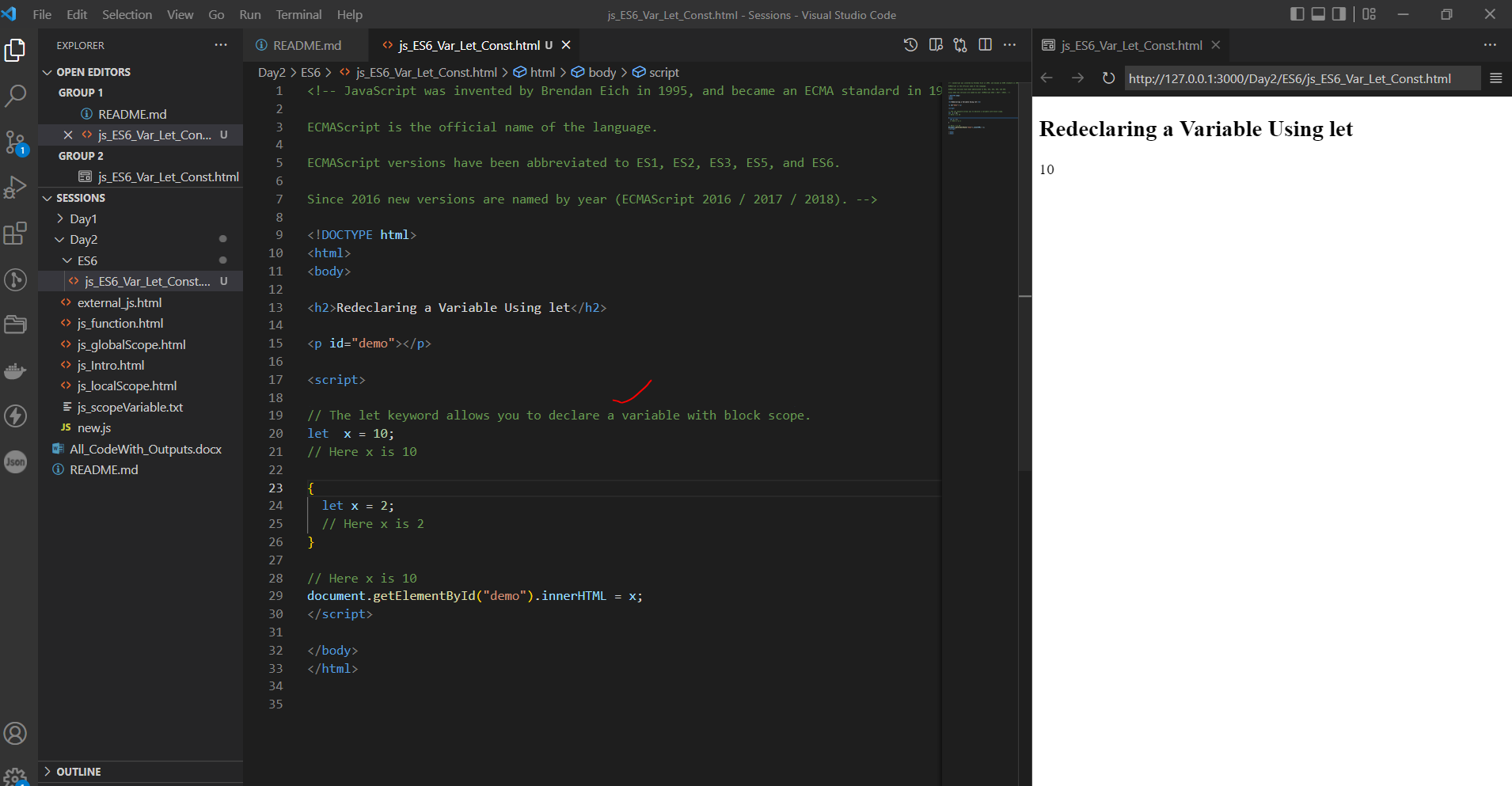


## GLOBAL SCOPE:

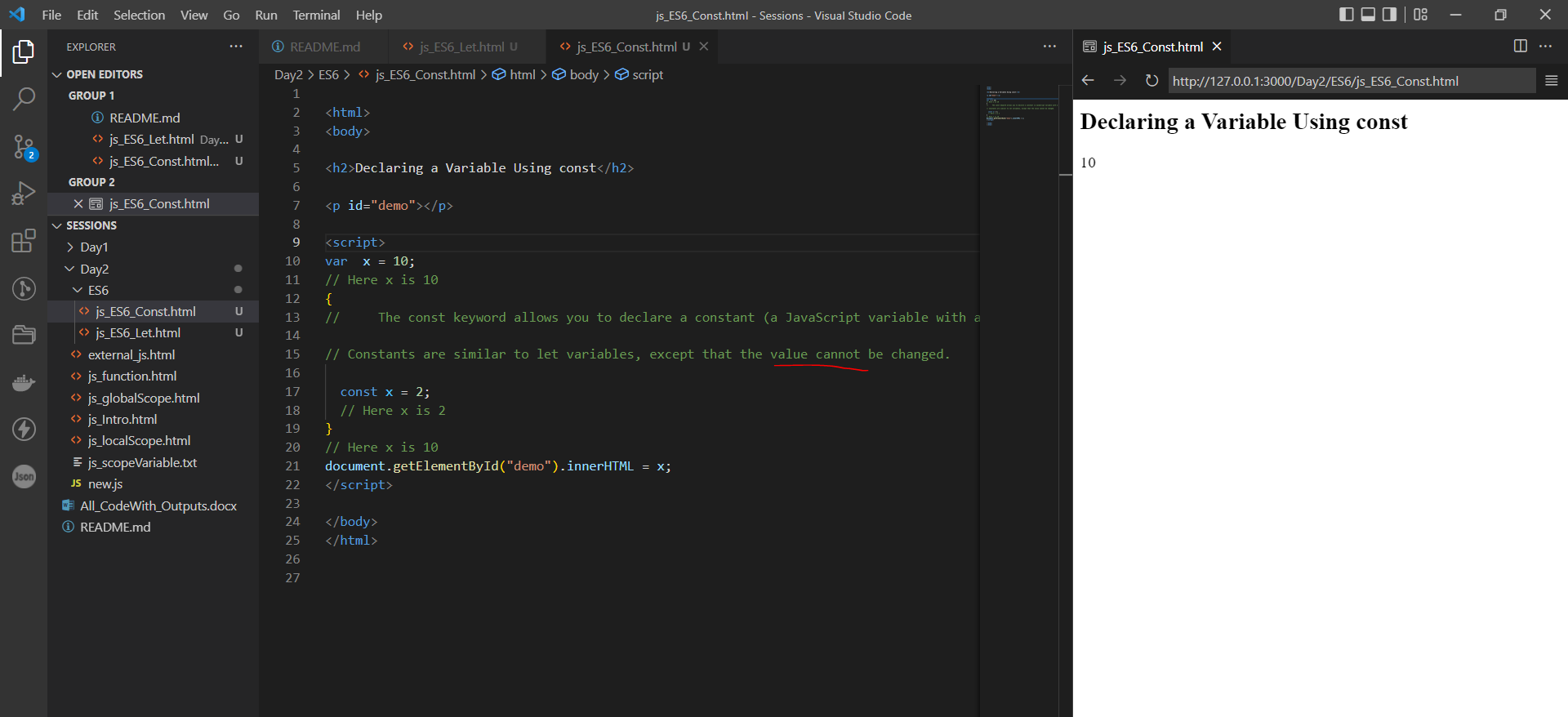


# ES6 STANDARDS:

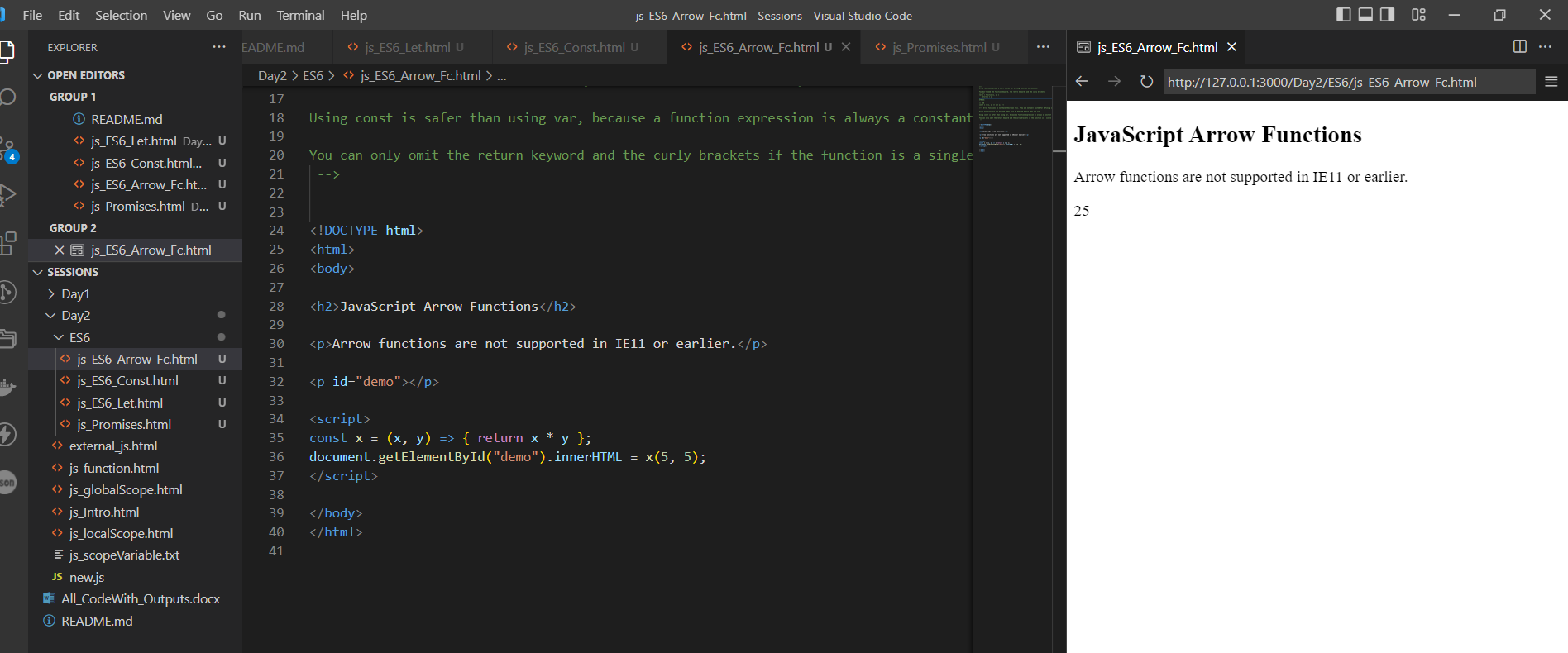
## let:



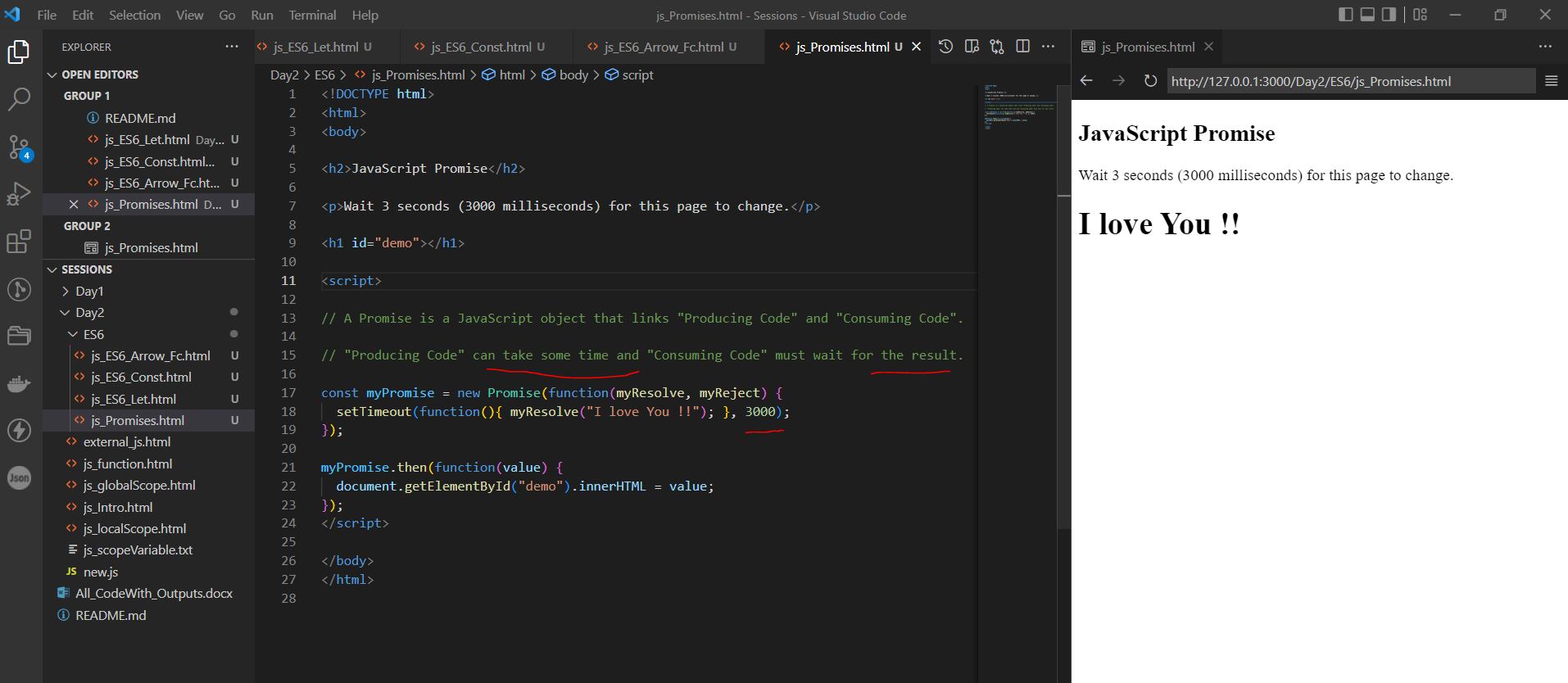
## Const:



## Arrow Function:

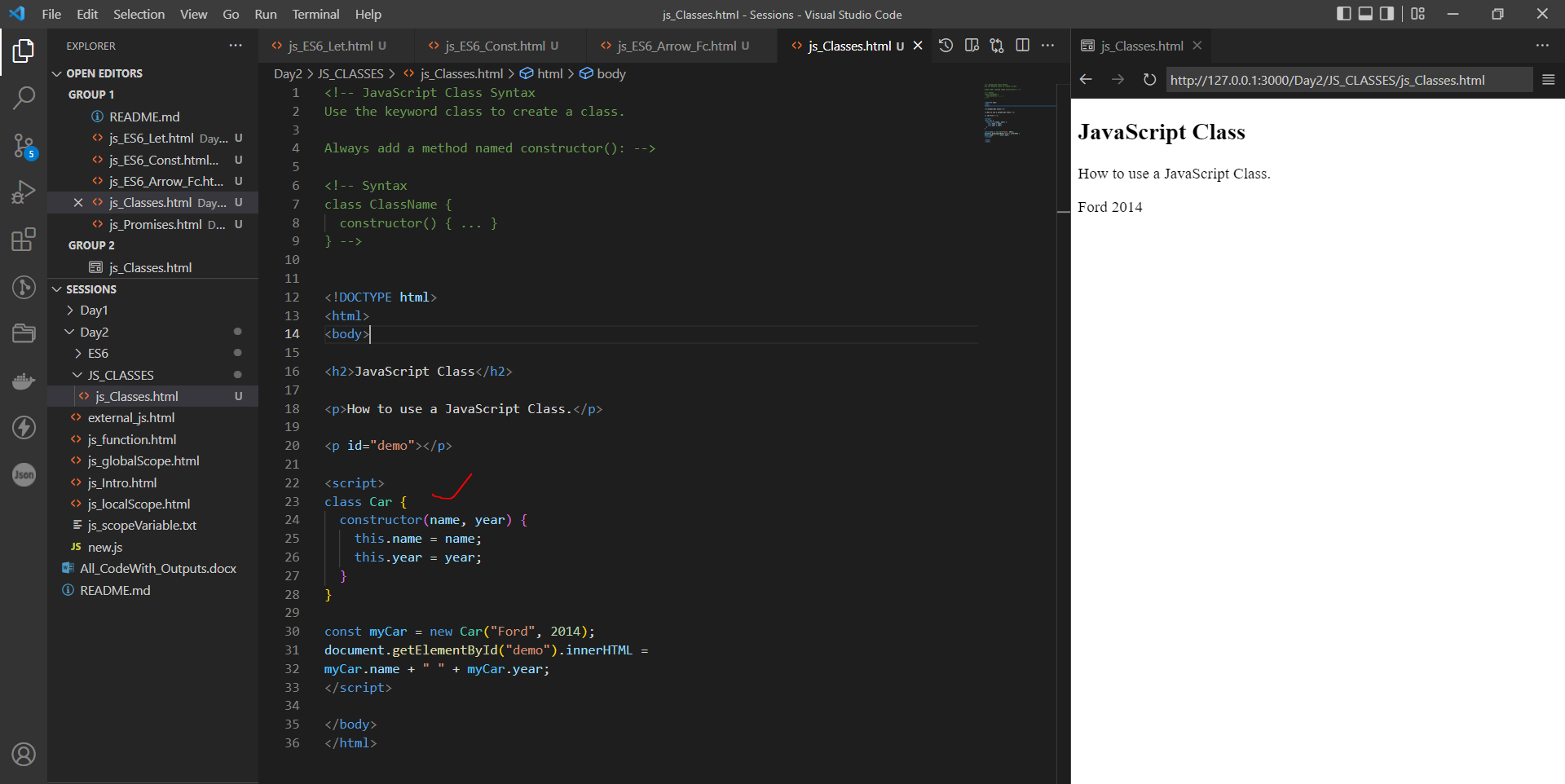


## Promises:



# JS CLASSES:

## JS class syntax:



## 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 ...

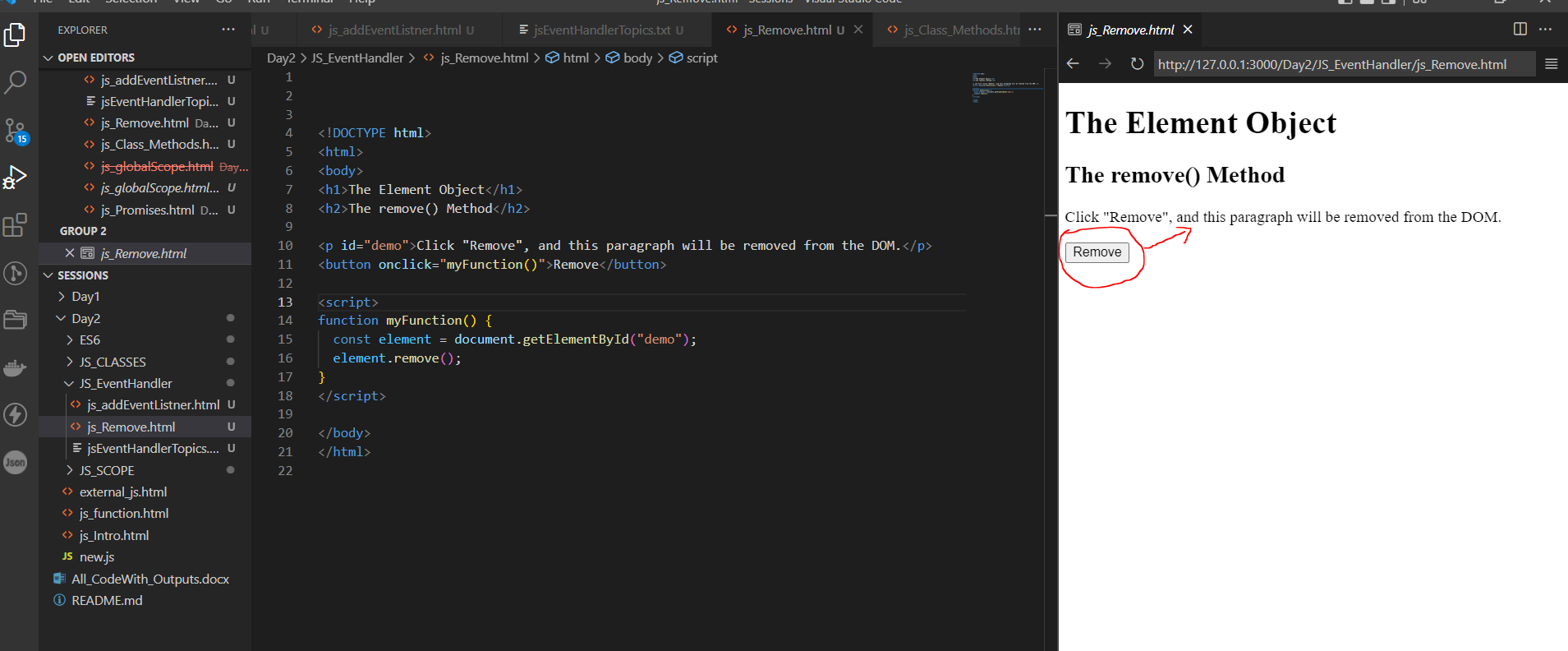
Many different methods can be used to let JavaScript work with events:

* 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 ...

addEventListner:

## remove:

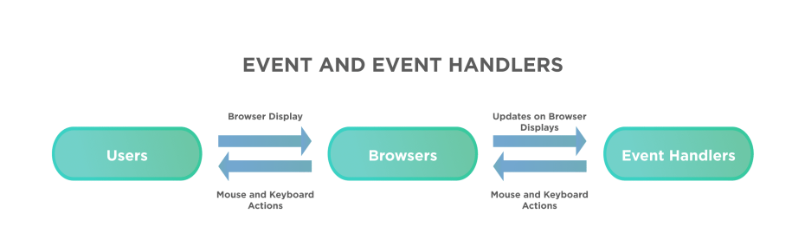
When you click this button the top paragraph letters will be removed.



# DAY-3

Why we require event handlers?

When an event, such as clicking an element or pressing a keyboard key, occurs on an HTML or DOM element, we can invoke certain functions based on these events. So, how do the HTML element knows when to execute the mentioned [***JavaScript function***](https://www.toolsqa.com/javascript/functions-in-javascript/) or JavaScript code?  The event handlers handle this. The event handlers are the properties of the HTML or DOM elements, which manages how the element should react to a specific event.

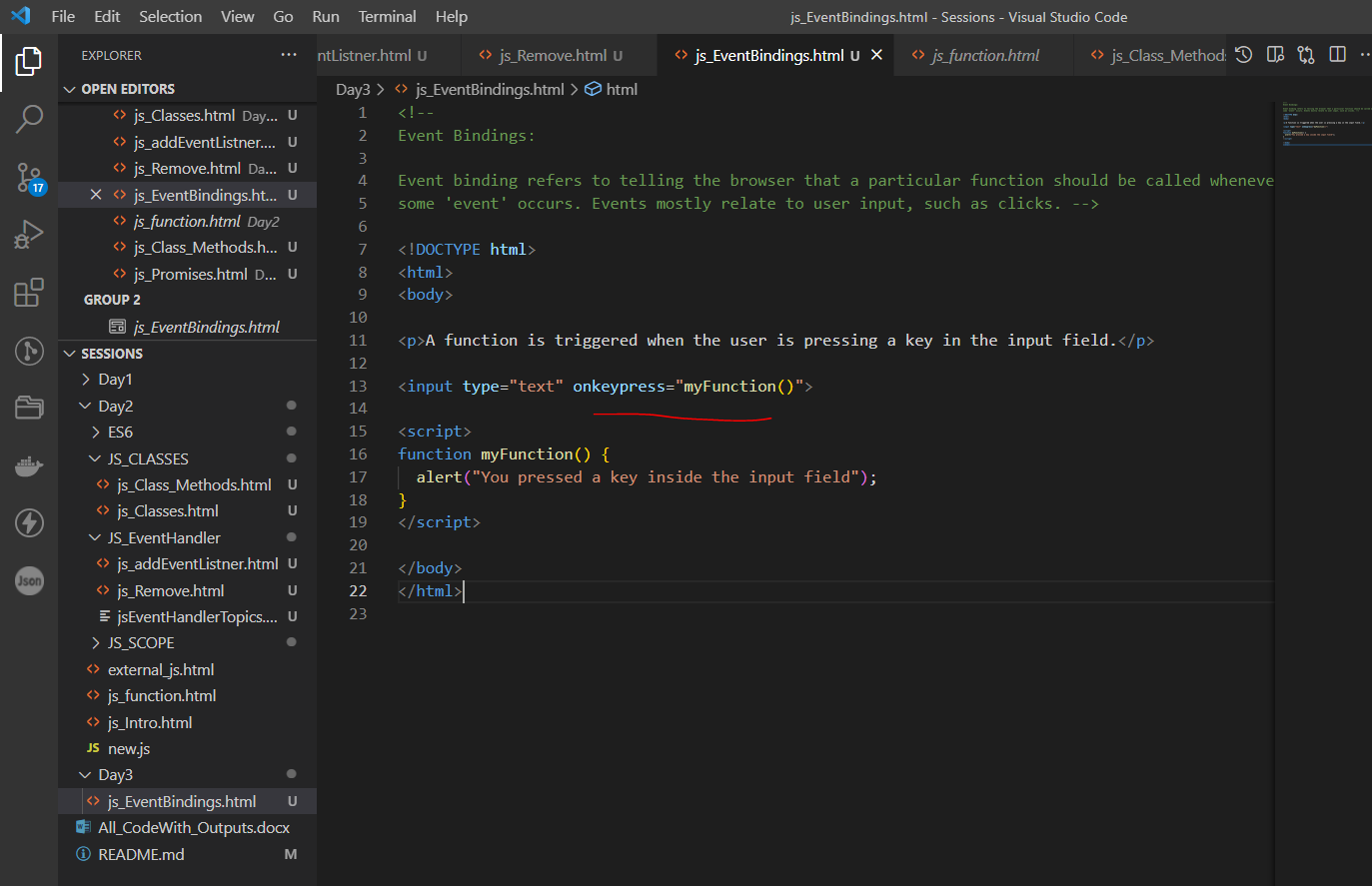


### ***What are the different types of event handlers provided by JavaScript?***

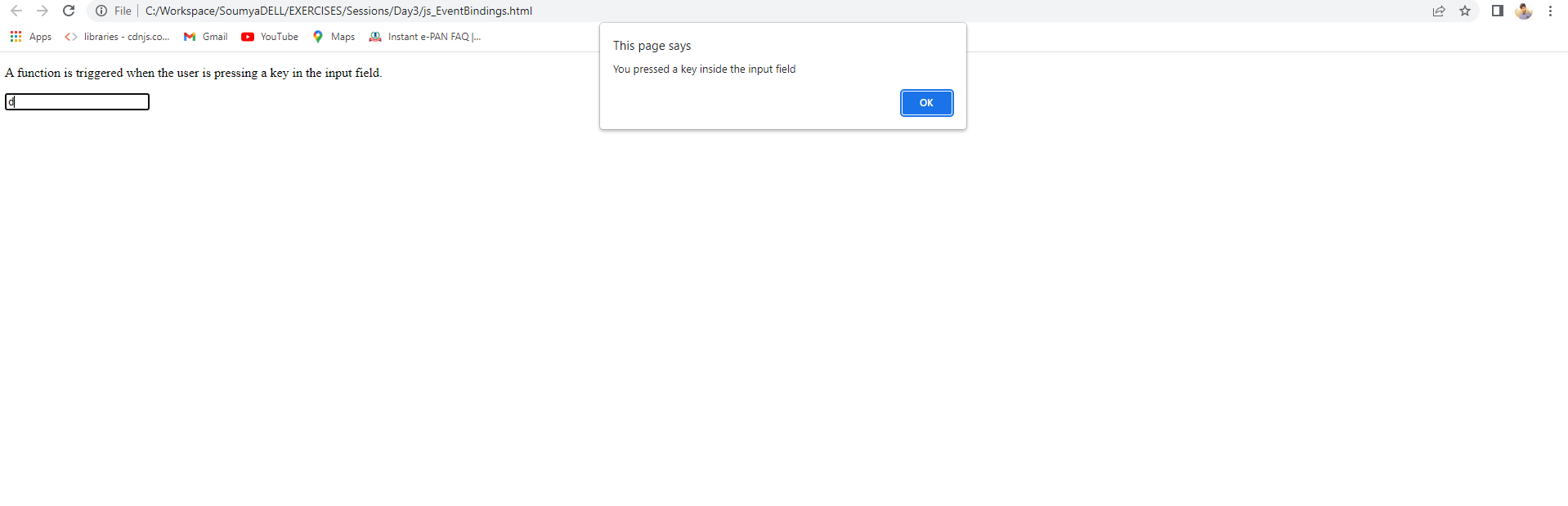
JavaScript provides various kinds of event handlers that get triggered based on specific actions on the HTML elements. Few of the event handlers are:

| **Event Handler** | **Description** |
| --- | --- |
| onclick | This event handler invokes a JavaScript code when a click action happens on an HTML element. E.g., when we click a button, a link is pushed, a checkbox checks or an image map is selected, it can trigger the onClick event handler. |
| onload | This event handler invokes a JavaScript code when a window or image finishes loading. |
| onmouseover | This event handler invokes a JavaScript code when we place the mouse over a specific link or an object. |
| onmouseout | This event handler invokes a JavaScript code when the mouse leaves a particular link or an object. |
| onkeypress | This event handler invokes a JavaScript code when the user presses a key. |
| onkeydown | This event handler invokes a JavaScript code when during the keyboard action, we press the key down. |
| onkeyup | This event handler invokes a JavaScript code when during the keyboard action, the |

Example onkeypress:



OUTPUT:



## Pass by Value:

**Pass By Value:** In Pass by value, function is called by directly passing the value of the variable as an argument. So any changes made inside the function does not affect the original value.

In Pass by value, parameters passed as an arguments create its **own copy.** So any changes made inside the function is made to the copied value not to the original value .

Let us take an example to understand better:

function Passbyvalue(a, b) {

let tmp;

tmp = b;

b = a;

a = tmp;

console.log(`Inside Pass by value

function -> a = ${a} b = ${b}`);

}

let a = 1;

let b = 2;

console.log(`Before calling Pass by value

Function -> a = ${a} b = ${b}`);

Passbyvalue(a, b);

console.log(`After calling Pass by value

Function -> a =${a} b = ${b}`);

**Output:**

Before calling Pass by value Function -> a = 1 b = 2

Inside Pass by value function -> a = 2 b = 1

After calling Pass by value Function -> a =1 b = 2

## Pass by Reference:

**Pass by Reference:** In Pass by Reference, Function is called by directly passing the reference/address of the variable as an argument. So changing the value inside the function also change the original value. In JavaScript **array and Object**follows pass by reference property.

In Pass by reference, parameters passed as an arguments does not create its own copy, it refers to the original value so changes made inside function affect the original value.

let us take an example to understand better.

function PassbyReference(obj) {

let tmp = obj.a;

obj.a = obj.b;

obj.b = tmp;

console.log(`Inside Pass By Reference

Function -> a = ${obj.a} b = ${obj.b}`);

}

let obj = {

a: 10,

b: 20

}

console.log(`Before calling Pass By Reference

Function -> a = ${obj.a} b = ${obj.b}`);

PassbyReference(obj)

console.log(`After calling Pass By Reference

Function -> a = ${obj.a} b = ${obj.b}`);

**Output:**

Before calling Pass By Reference Function -> a = 10 b = 20

Inside Pass By Reference Function -> a = 20 b = 10

After calling Pass By Reference Function -> a = 20 b = 10

**Note:** In Pass by Reference, we are mutating the original value. when we pass an object as an arguments and update that object’s reference in the function’s context, that won’t affect the object value. But if we mutate the object internally, It will affect the object .

**Example 1:** Updating the object reference in the function.

function PassbyReference(obj) {

// Changing the reference of the object

obj = {

a: 10,

b: 20,

c: "GEEKSFORGEEKS"

}

console.log(`Inside Pass by

Reference Function -> obj `);

console.log(obj);

}

let obj = {

a: 10,

b: 20

}

console.log(`Updating the object reference -> `)

console.log(`Before calling Pass By

Reference Function -> obj`);

console.log(obj);

PassbyReference(obj)

console.log(`After calling Pass By

Reference Function -> obj`);

console.log(obj);

Updating the object reference ->

Before calling PassByReference Function -> obj

{a: 10, b: 20}

Inside PassbyReference Function -> obj

{a: 10, b: 20, c: "GEEKSFORGEEKS"}

After calling PassByReference Function -> obj

{a: 10, b: 20}

**Example 2:** Mutating the original Object.

function PassbyReference(obj) {

// Mutating the origanal object

obj.c = "GEEKSFORGEEKS";

console.log(`Inside Pass by

Reference Function -> obj `);

console.log(obj);

}

let obj = {

a: 10,

b: 20

}

console.log(`Mutating the origanal object -> `)

console.log(`Before calling Pass By

Reference Function -> obj`);

console.log(obj);

PassbyReference(obj)

console.log(`After calling Pass By

Reference Function -> obj`);

console.log(obj);

**Output:**

Mutating the origanal object ->

Before calling PassByReference Function -> obj

{a: 10, b: 20}

Inside PassbyReference Function -> obj

{a: 10, b: 20, c: "GEEKSFORGEEKS"}

After calling PassByReference Function -> obj

{a: 10, b: 20, c: "GEEKSFORGEEKS"}

# JS ASYNC/AWAIT:

## Async/Await:

*"async and await make promises easier to write"*

**async** makes a function return a Promise

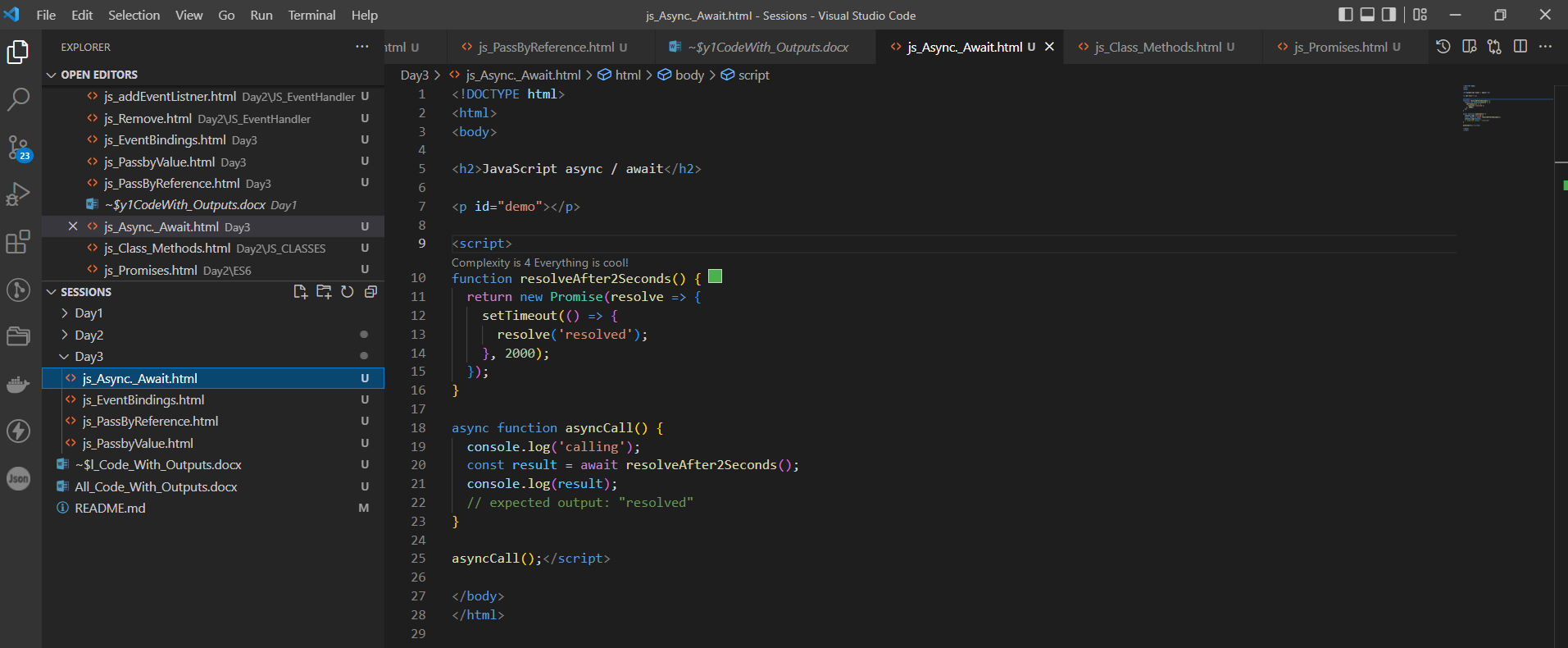
**await** makes a function wait for a Promise

Syntax:

async function name([param[, param[, ...param]]]) {

statements

}



Output:

