

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

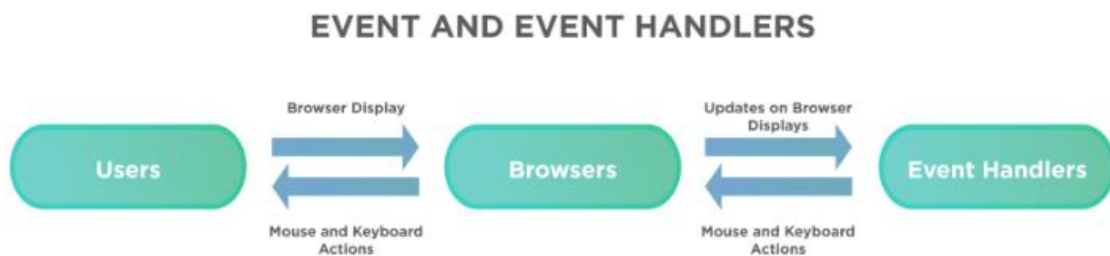
remove:.....	21
DAY-3	22
<i>What are the different types of event handlers provided by JavaScript?.....</i>	<i>22</i>
Pass by Value:.....	24
Pass by Reference:.....	25
JS ASYNC/AWAIT:	29
Async/Await:	29
Object Keys:	30
Object Entries:.....	32
HTML_JS_CUSTOM ATTRIBUTES:	34
EXAMPLE-1:	34
EXAMPLE-2	35
Html & 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-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* 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.

Event Handler	Description
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:

The screenshot shows the Visual Studio Code editor with a file named `js_EventBindings.html` open. The editor displays the following HTML and JavaScript code:

```

1 <!--
2 Event Bindings:
3
4 Event binding refers to telling the browser that a particular function should be called whenever
5 some 'event' occurs. Events mostly relate to user input, such as clicks. -->
6
7 <!DOCTYPE html>
8 <html>
9 <body>
10
11 <p>A function is triggered when the user is pressing a key in the input field.</p>
12
13 <input type="text" onkeypress="myFunction()">
14
15 <script>
16 function myFunction() {
17   alert("You pressed a key inside the input field");
18 }
19 </script>
20
21 </body>
22 </html>
23

```

The `onkeypress="myFunction()"` attribute in the `<input>` tag is highlighted with a red underline. The `myFunction()` function is also highlighted with a red underline. The Explorer sidebar on the left shows the project structure, including files like `js_EventBindings.html`, `js_function.html`, and `js_Class_Methods.html`.

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 original 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 original 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 original 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"

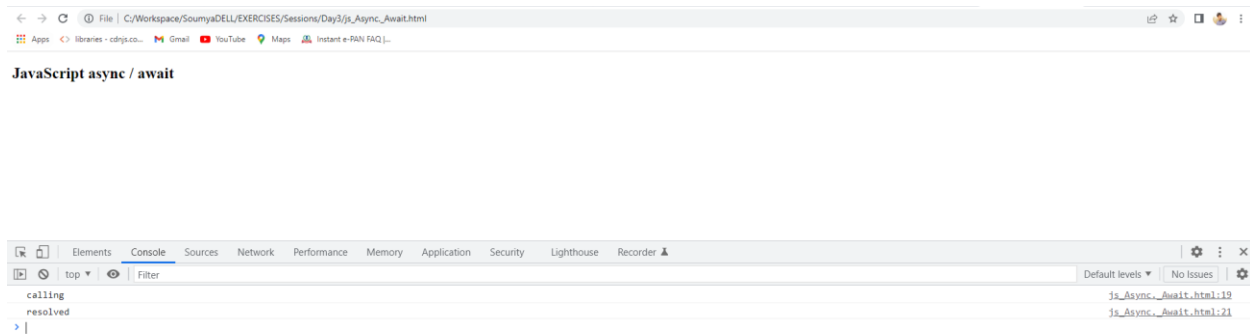
async makes a function return a Promise

await makes a function wait for a Promise

Syntax:

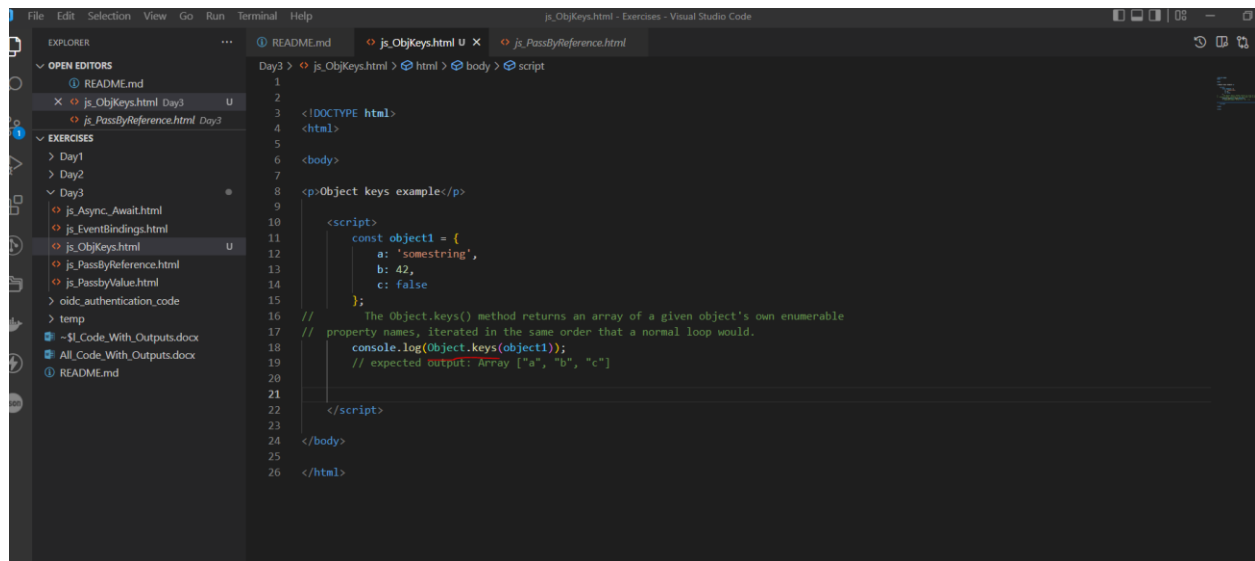
```
async function name([param[, param[, ...param]]]) {  
  statements  
}
```

Output:



Object Keys:

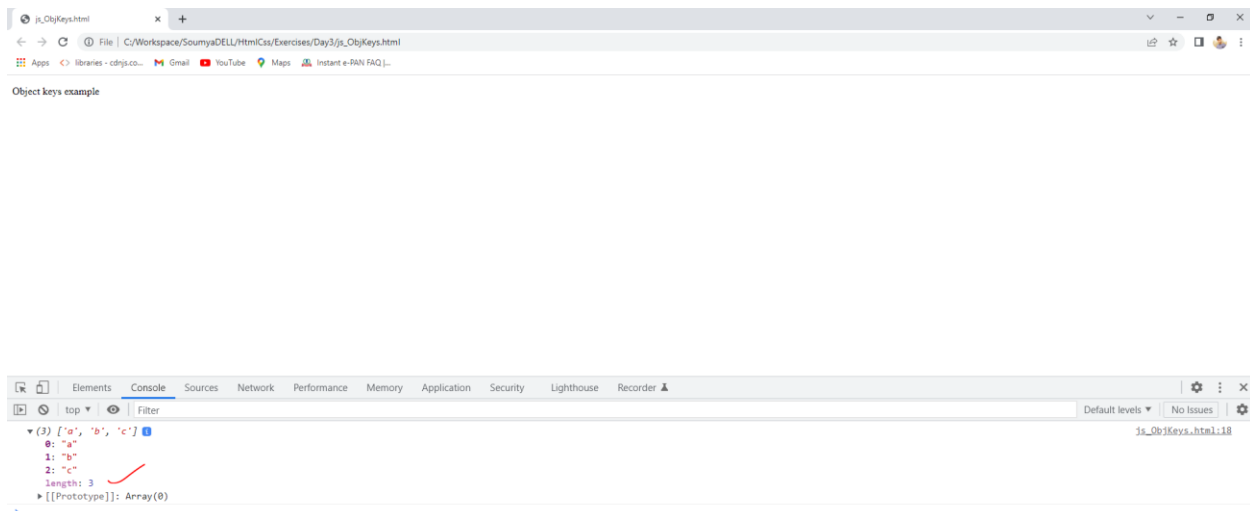
The `Object.keys()` method returns an array of a given object's own enumerable property *names*, iterated in the same order that a normal loop would.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays a project structure with folders for 'OPEN EDITORS' and 'EXERCISES'. The 'EXERCISES' folder contains subfolders for 'Day1', 'Day2', and 'Day3'. Under 'Day3', there are files for 'js_Async_Await.html', 'js_EventBindings.html', 'js_ObjKeys.html' (which is selected and marked with a 'U'), 'js_PassByReference.html', 'js_PassByValue.html', 'oidc_authentication_code', and 'temp'. Below these are document files: '-\$!_Code_With_Outputs.docx', 'All_Code_With_Outputs.docx', and 'README.md'. The main editor area shows the content of 'js_ObjKeys.html'. It is an HTML document with a DOCTYPE declaration, html, head, and body tags. Inside the body, there is a paragraph with the text 'Object keys example' and a script block. The script block contains a JavaScript object definition and a comment explaining the Object.keys() method. The code is as follows:

```
1
2
3 <!DOCTYPE html>
4 <html>
5
6 <body>
7
8 <p>Object keys example</p>
9
10 <script>
11     const object1 = {
12         a: 'somestring',
13         b: 42,
14         c: false
15     };
16     // The Object.keys() method returns an array of a given object's own enumerable
17     // property names, iterated in the same order that a normal loop would.
18     console.log(Object.keys(object1));
19     // expected output: Array ["a", "b", "c"]
20
21 </script>
22
23
24 </body>
25
26 </html>
```

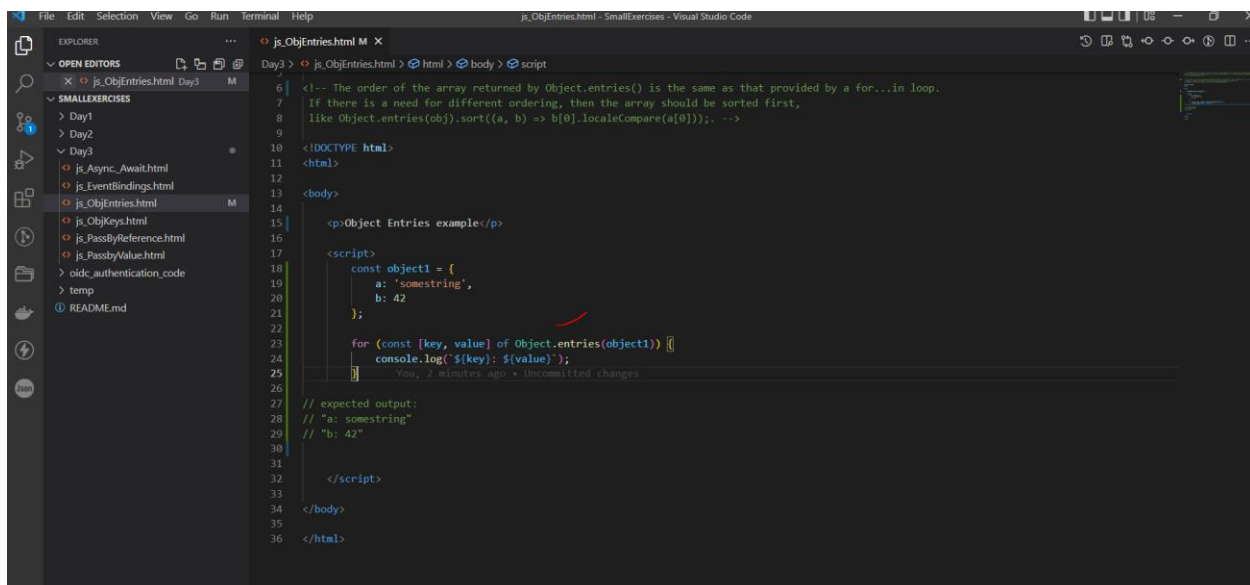
Output:



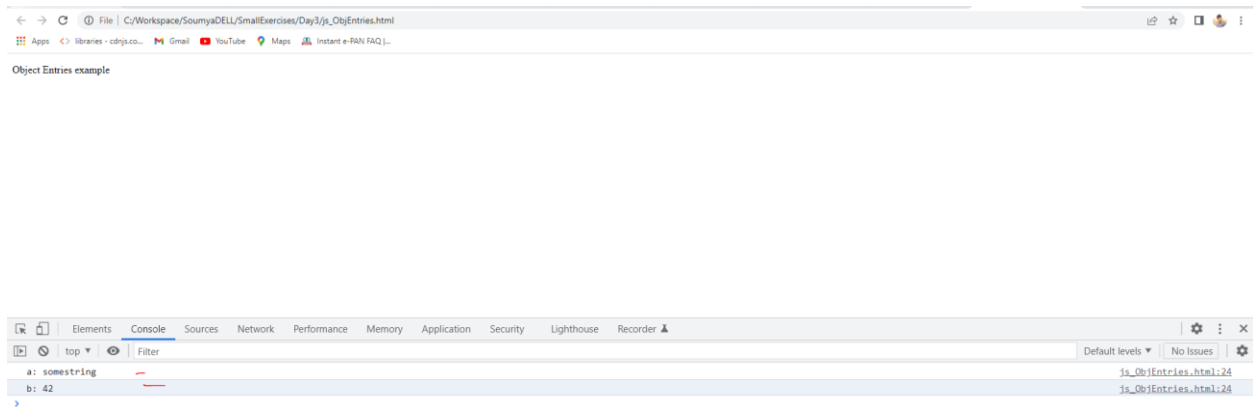
Object Entries:

The `Object.entries()` method returns an array of a given object's own enumerable string-keyed property `[key, value]` pairs. This is the same as iterating with a [for...in](#) loop, except that a `for...in` loop enumerates properties in the prototype chain as well.

The order of the array returned by `Object.entries()` is the same as that provided by a [for...in](#) loop. If there is a need for different ordering, then the array should be sorted first, like `Object.entries(obj).sort((a, b) => b[0].localeCompare(a[0]))`.



Output:



[Syntax](#)

`Object.entries(obj)`

Copy to Clipboard

[Parameters](#)

`obj`

The object whose own enumerable string-keyed property `[key, value]` pairs are to be returned.

[Return value](#)

An array of the given object's own enumerable string-keyed property `[key, value]` pairs.

[Description](#)

`Object.entries()` returns an array whose elements are arrays corresponding to the enumerable string-keyed property `[key, value]` pairs found directly upon `object`. The ordering of the properties is the same as that given by looping over the property values of the object manually.

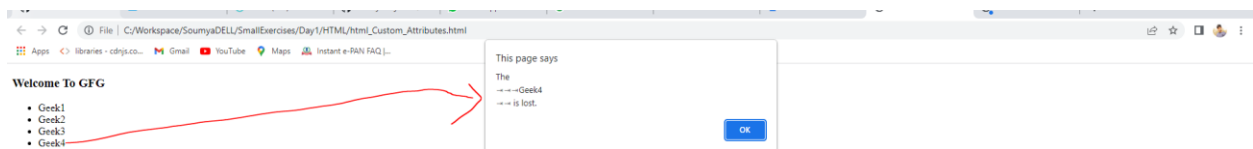
HTML_JS_CUSTOM ATTRIBUTES:

EXAMPLE-1:

```
<!-- HTML File -->
<!DOCTYPE html>
<html>
<head>
<title>GeeksForGeeks</title>
</head>
<body>
<h3>Welcome To GFG</h3>
<ul>
<li onclick="showPosition(this)" id="geek1" data-position="winner">
Geek1
</li>
<li onclick="showPosition(this)" id="geek2" data-position="runner up">
Geek2
</li>
<li onclick="showPosition(this)" id="geek3" data-position="third">
Geek3
</li>
<li onclick="showPosition(this)" id="geek4" data-position="lost">
Geek4
</li>
</ul>
<script src="custom_attributes.js"></script>
</body>
</html>
```

```
// JavaScript File
// This JavaScript file will handle the custom attributes which we
// We will use this to create an alert whenever a list item is clicked
function showPosition(runner) {
var position = runner.getAttribute("data-position");
alert("The " + runner.innerHTML + " is " + position + ".");
}
// Whenever an item is clicked this code snippet will be triggered
// type of food that is stored in the custom attribute using the
function handleFood(food) {
var foodType = food.dataset.foodType;
alert(food.innerHTML + " is " + foodType + ".");
}
```

Output:

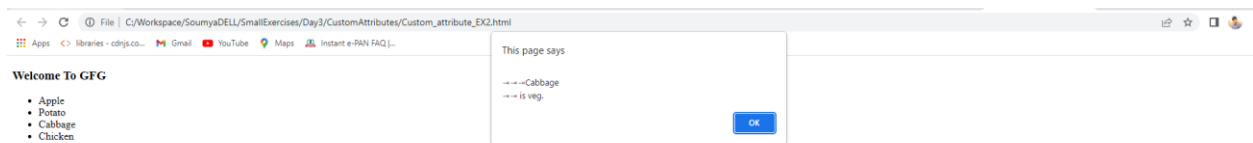


EXAMPLE-2

```
Day3 > CustomAttributes > Custom_attribute_EX2.html > ...
12 <head>
13 <title>GeeksForGeeks</title>
14 </head>
15
16 <body>
17 <h3>Welcome To GFG</h3>
18
19 <ul>
20 <li onclick="handleFood(this)"
21   data-food-type="fruit">
22   Apple
23 </li>
24
25 <li onclick="handleFood(this)"
26   data-food-type="vegetable">
27   Potato
28 </li>
29
30 <li onclick="handleFood(this)"
31   data-food-type="veg">
32   Cabbage
33 </li>
34
35 <li onclick="handleFood(this)"
36   data-food-type="non-veg">
37   Chicken
38 </li>
39 </ul>
40
41 <script src="custom_attributes.js"></script>
42
43 </body>
44 </html>
```

```
Day3 > CustomAttributes > custom_attributes.js > ...
1 // This JavaScript file will handle the custom attributes which w
2 // We will use this to create an alert whenever a list item is c
3
4
5 function showPosition(runner) {
6   var position = runner.getAttribute("data-position");
7   alert("The " + runner.innerHTML + " is " + position + ".");
8 }
9 // Whenever an item is clicked this code snippet will be trigger
10 // type of food that is stored in the custom attribute using the
11 function handleFood(food) {
12   var foodType = food.dataset.foodType;
13   alert(food.innerHTML + " is " + foodType + ".");
14 }
15
```

Output:



Html & ExternalCss:

EXAMPLE:3

