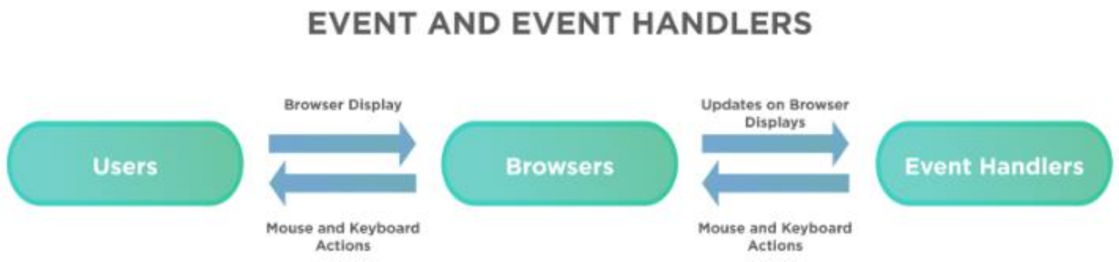
Contents

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.


EVENT AND EVENT HANDLERS
Browser Display
Updates on Browser Displays
Users
Browsers
Event Handlers
Mouse and Keyboard Actions
Mouse and Keyboard Actions

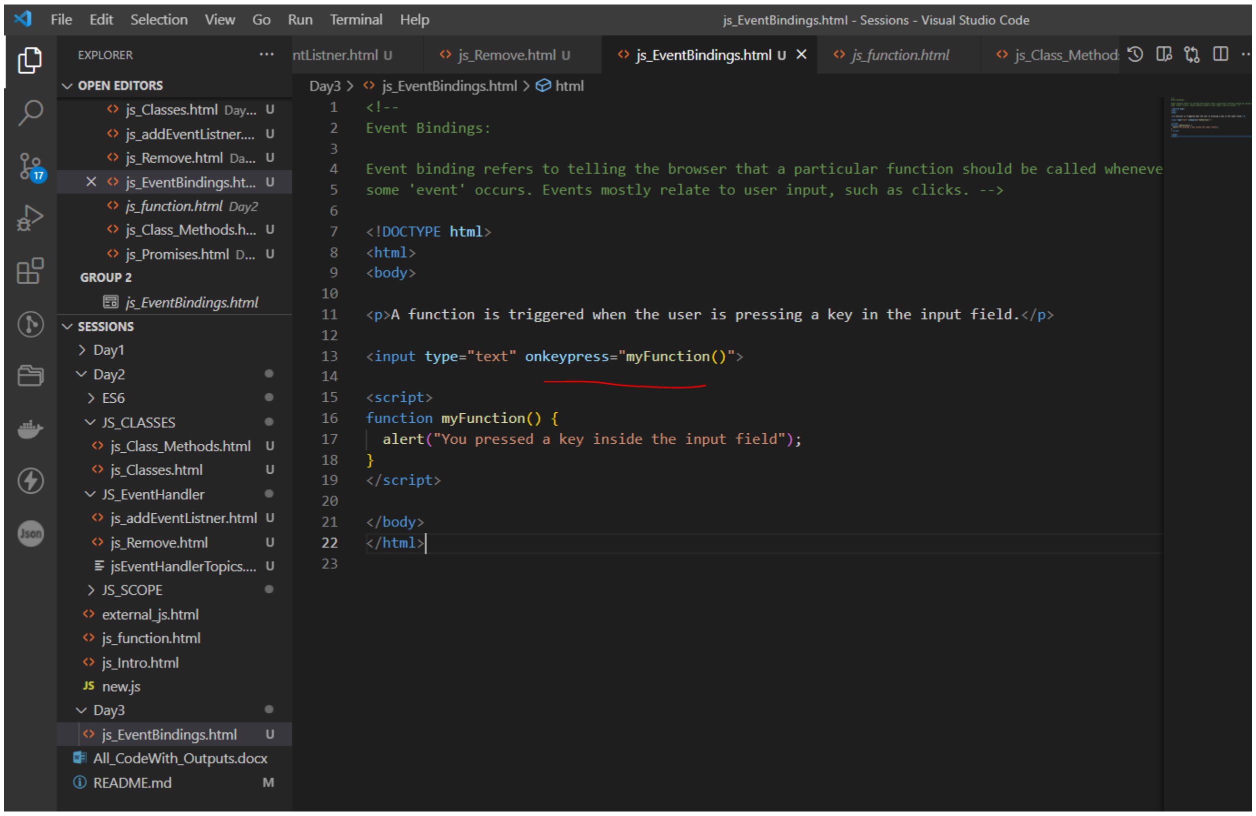
What are the different types of event handlers provided by JavaScript?

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.

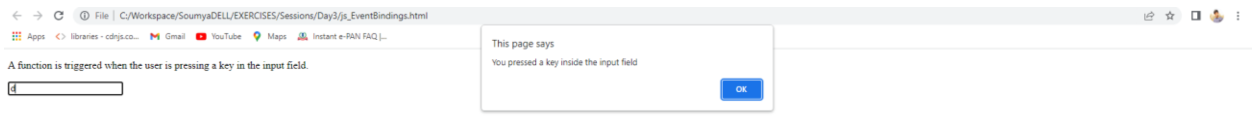
Event Handler	Description
onkeypress	This event handler invokes a JavaScript code when the user presses a key.
onkeydown	This event handler invokes a JavaScript code when during the keyboard action, we press the key down.
onkeyup	This event handler invokes a JavaScript code when during the keyboard action, the

Example onkeypress:



```html
<!--
Event Bindings:

Event binding refers to telling the browser that a particular function should be called wheneve
some 'event' occurs. Events mostly relate to user input, such as clicks. -->

<!DOCTYPE html>
<html>
<body>

<p>A function is triggered when the user is pressing a key in the input field.</p>

<input type="text" onkeypress="myFunction()">

<script>
function myFunction() {
  alert("You pressed a key inside the input field");
}
</script>

</body>
</html>
```

OUTPUT:


C:/Workspace/SoumyaDELL/EXERCISES/Sessions/Day3/js_EventBindings.html

A function is triggered when the user is pressing a key in the input field.

d

This page says

You pressed a key inside the input field

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:

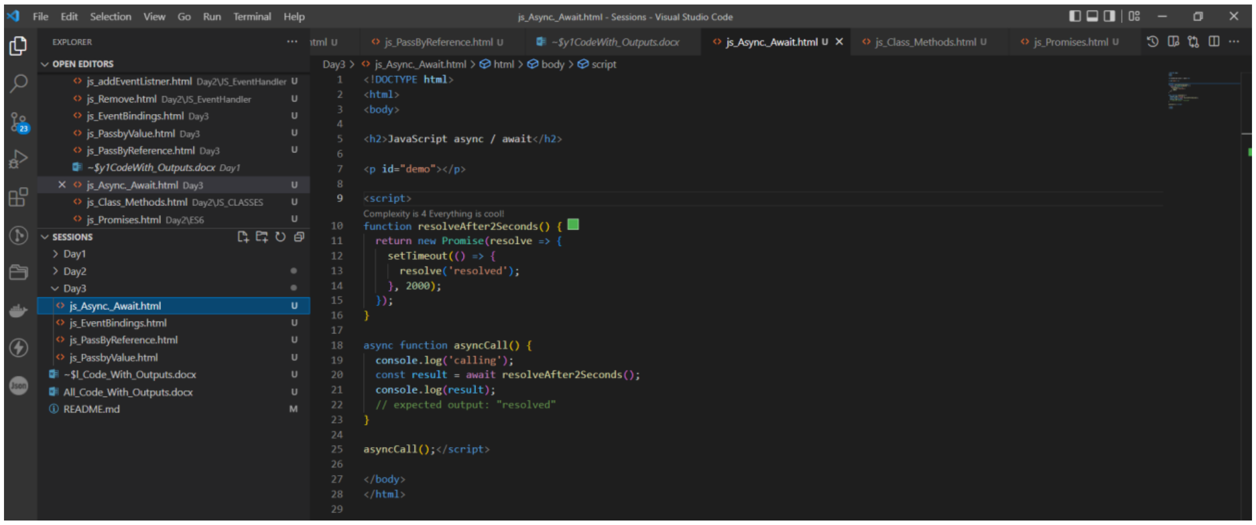
"async and await make promises easier to write"

async makes a function return a Promise

await makes a function wait for a Promise

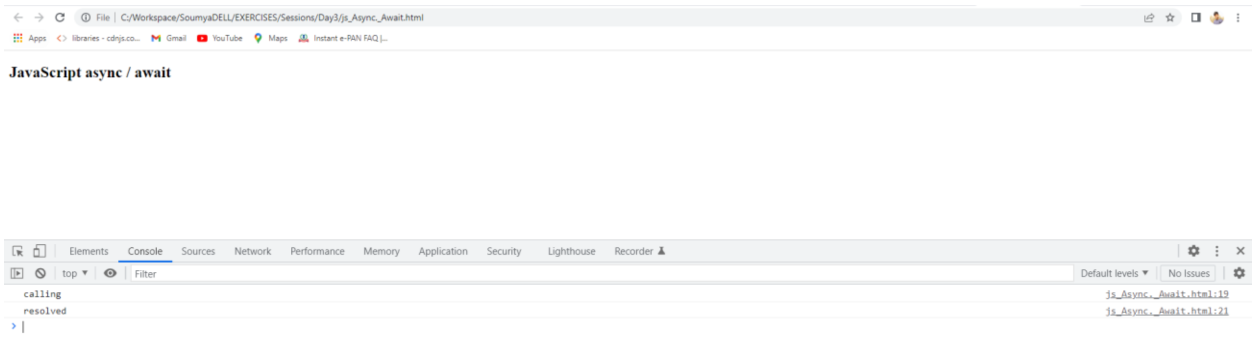
Syntax:

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



```html
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript async / await</h2>

<p id="demo"></p>

<script>
function resolveAfter2Seconds() {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve('resolved');
    }, 2000);
  });
}

async function asyncCall() {
  console.log('calling');
  const result = await resolveAfter2Seconds();
  console.log(result);
  // expected output: "resolved"
}

asyncCall();</script>

</body>
</html>
```

Output:

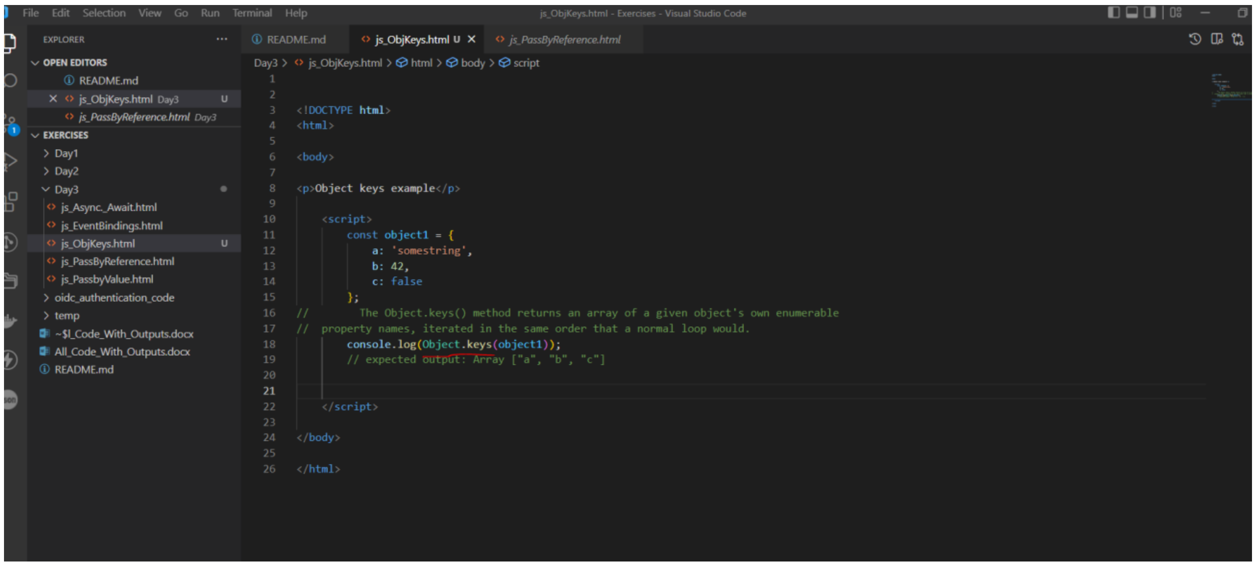


JavaScript async / await

```
calling                                   js_Async_Await.html:19
resolved                                  js_Async_Await.html:21
```

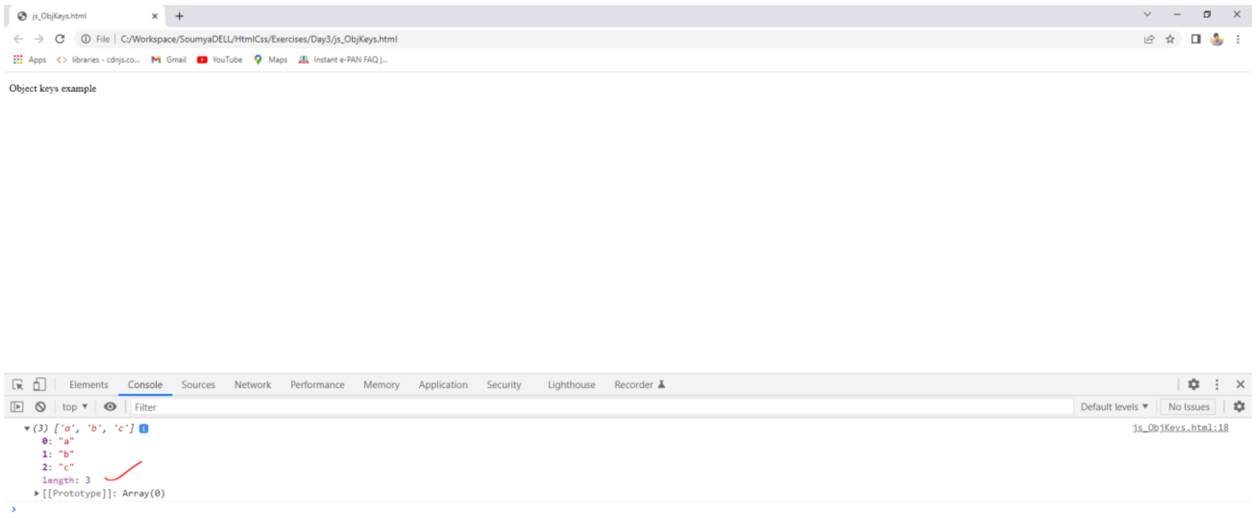
Object Keys:

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.



```


<!DOCTYPE html>
<html>

<body>

<p>Object keys example</p>

    <script>
        const object1 = {
            a: 'somestring',
            b: 42,
            c: false
        };
//        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.
        console.log(Object.keys(object1));
        // expected output: Array ["a", "b", "c"]


    </script>

</body>

</html>
```

Output:



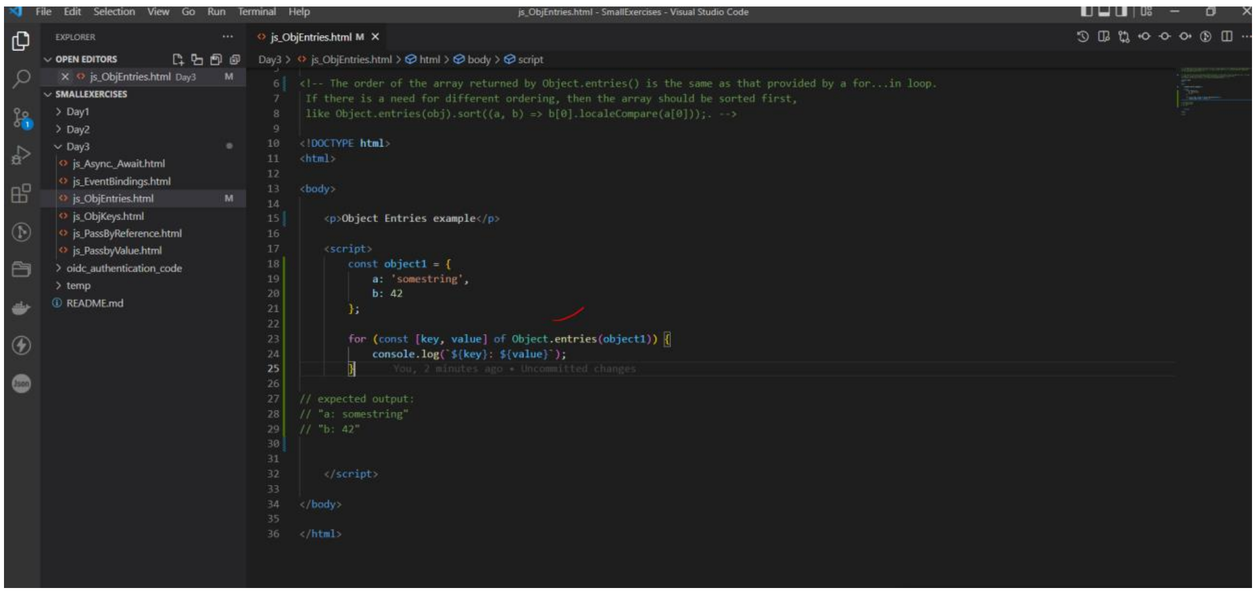
Object keys example

```
▼ (3) ['a', 'b', 'c']
    0: "a"
    1: "b"
    2: "c"
    length: 3
  ▶ [[Prototype]]: Array(0)
```

Object Entries:

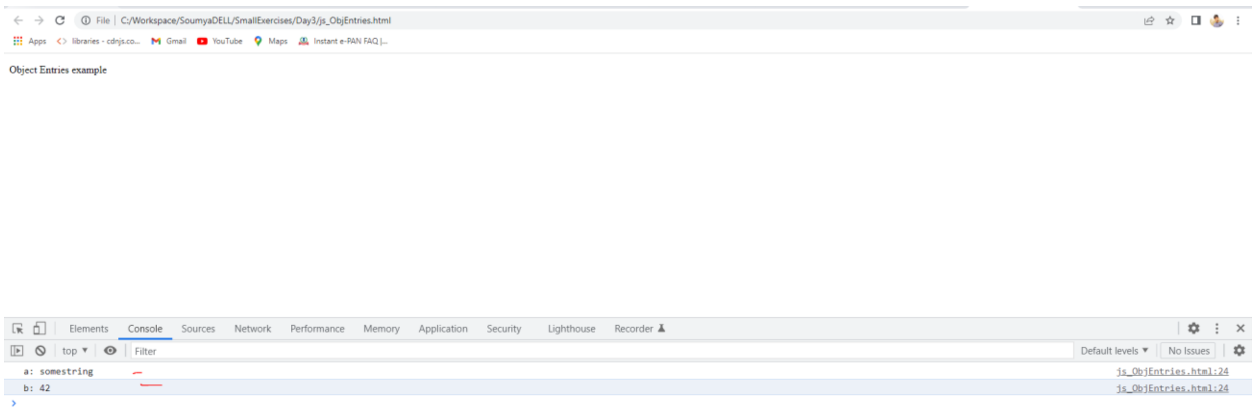
The **`Object.entries()`** method returns an array of a given object's own enumerable string-keyed property `[key, value]` pairs. This is the same as iterating with a [`for...in`]() loop, except that a `for...in` loop enumerates properties in the prototype chain as well.

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]));`.



```html
<!-- 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]));. -->

<!DOCTYPE html>
<html>

<body>

    <p>Object Entries example</p>

    <script>
        const object1 = {
            a: 'somestring',
            b: 42
        };

        for (const [key, value] of Object.entries(object1)) {
            console.log(`${key}: ${value}`);
        }

// expected output:
// "a: somestring"
// "b: 42"


    </script>

</body>

</html>
```

Output:



Object Entries example

```
a: somestring                                    js_ObjEntries.html:24
b: 42                                            js_ObjEntries.html:24
```

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

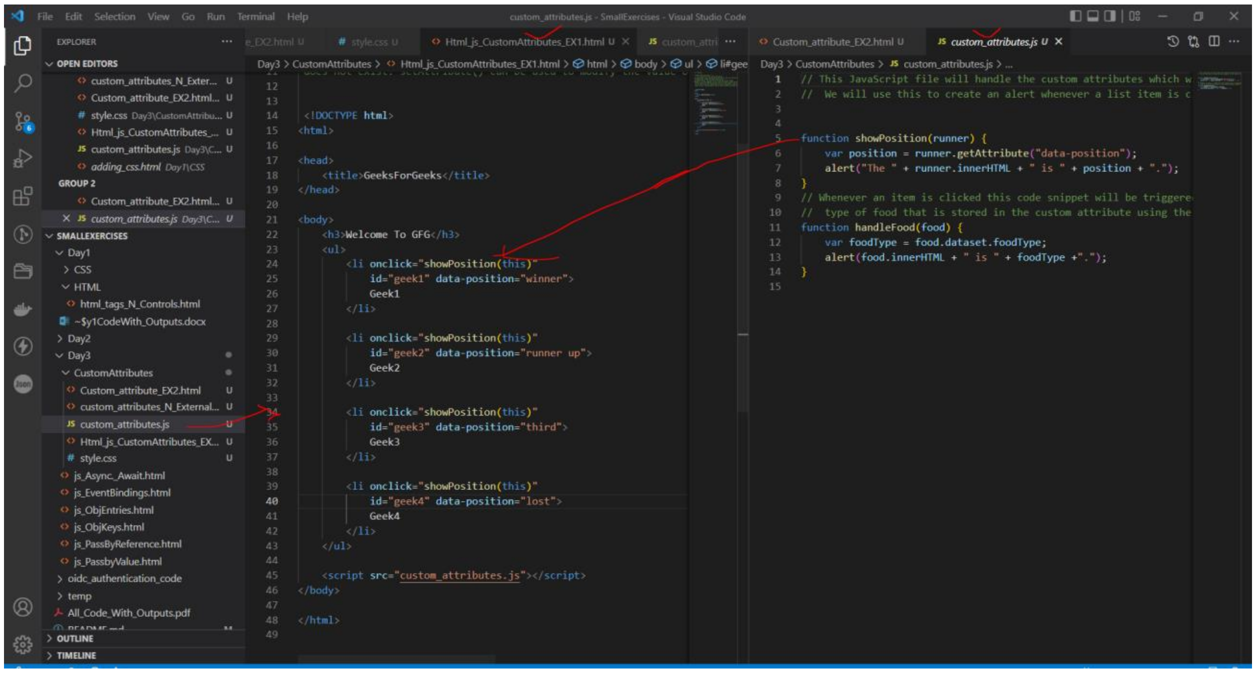
An array of the given object's own enumerable string-keyed property `[key, value]` pairs.

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:



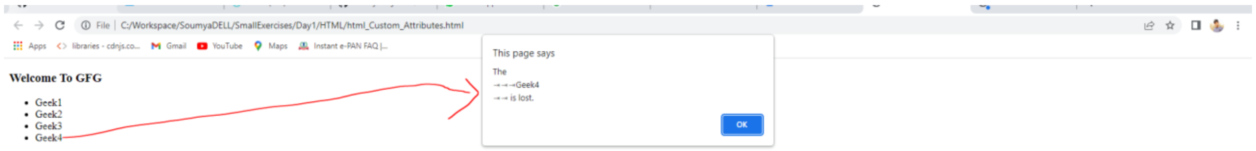
Html_js_CustomAttributes_EX1.html

```html
<!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>
```

custom_attributes.js

```javascript
// This JavaScript file will handle the custom attributes which w
//  We will use this to create an alert whenever a list item is c


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 triggere
//  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:



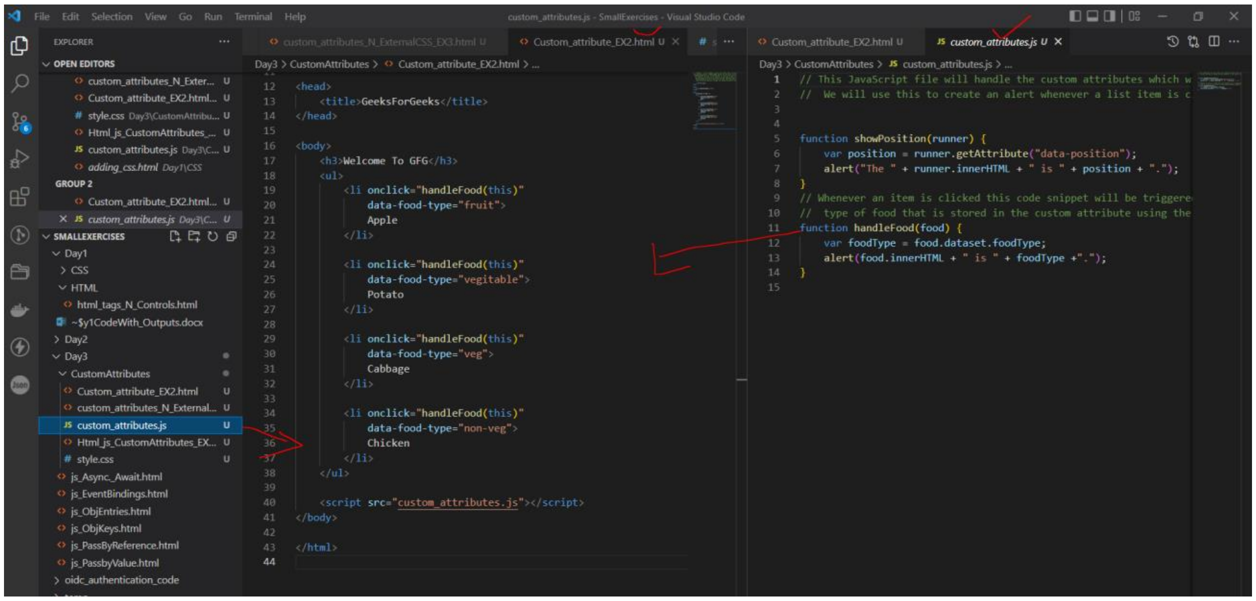
C:/Workspace/SoumyaDELL/SmallExercises/Day1/HTML/html_Custom_Attributes.html

Welcome To GFG

- Geek1
- Geek2
- Geek3
- Geek4

```
This page says

The
Geek4
is lost.

OK
```

EXAMPLE-2



custom_attributes.js - SmallExercises - Visual Studio Code

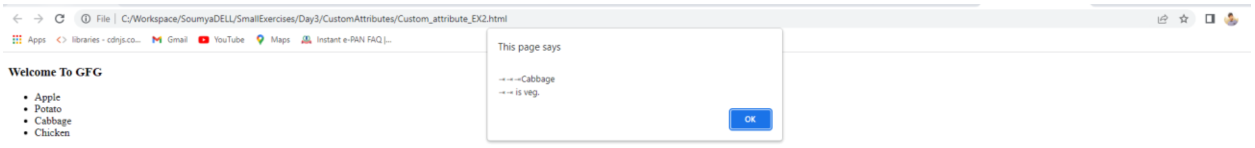
Day3 > CustomAttributes > Custom_attribute_EX2.html > ...

```html
<head>
    <title>GeeksForGeeks</title>
</head>

<body>
    <h3>Welcome To GFG</h3>
    <ul>
        <li onclick="handleFood(this)"
            data-food-type="fruit">
            Apple
        </li>

        <li onclick="handleFood(this)"
            data-food-type="vegitable">
            Potato
        </li>

        <li onclick="handleFood(this)"
            data-food-type="veg">
            Cabbage
        </li>

        <li onclick="handleFood(this)"
            data-food-type="non-veg">
            Chicken
        </li>
    </ul>

    <script src="custom_attributes.js"></script>
</body>

</html>
```

Day3 > CustomAttributes > custom_attributes.js > ...

```javascript
// This JavaScript file will handle the custom attributes which w
//  We will use this to create an alert whenever a list item is c


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 triggere
//  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:



C:/Workspace/SoumyaDELL/SmallExercises/Day3/CustomAttributes/Custom_attribute_EX2.html

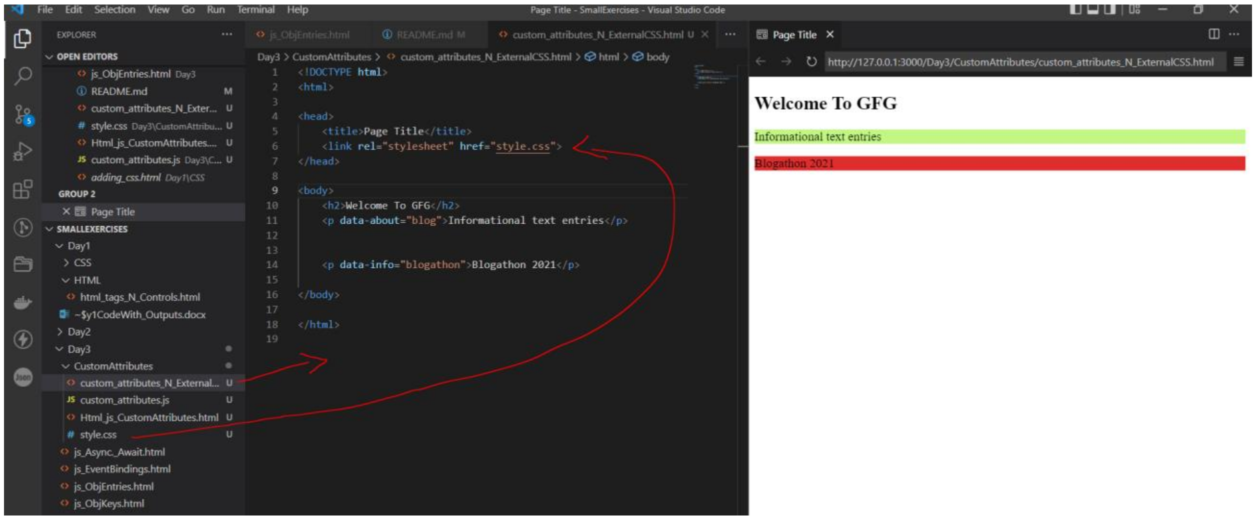
Welcome To GFG

- Apple
- Potato
- Cabbage
- Chicken

```
This page says

→→→Cabbage
→→ is veg.
```

Html & ExternalCss:

EXAMPLE:3



```html
<!DOCTYPE html>
<html>

<head>
    <title>Page Title</title>
    <link rel="stylesheet" href="style.css">
</head>

<body>
    <h2>Welcome To GFG</h2>
    <p data-about="blog">Informational text entries</p>


    <p data-info="blogathon">Blogathon 2021</p>

</body>

</html>
```

Welcome To GFG

Informational text entries

Blogathon 2021