JS
JavaScript

Class # 2

Javascript Arrays

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

Let car="BMW"

Let car="Corrolla"

Let car="Audi"

But if you have a large amount of cars then?

To store such a large data we use arrays.

Let cars=["BMW","Corrolla","Audi",….]

Javascript Arrays Method

Array length: Return length of array e.g const array=[“item1”,”item2”]

Array.length=2

Array toString():

const array=[“item1”,”item2”];

array.toString()= item1,item2

Array Join: Join array items with specified letter or character

Const array= [“item1”,”item2”,”item3”]

array.join(“ * ”)= item1 * item2 * item3

Javascript Arrays Method

Array Push: Add item to array

Let array=[“item1”,”item2”];

array.push(“item3”);

So now array=[“item1”,”item2”,”item3”]

Array pop(): Remove last item from array

const array=[“item1”,”item2”];

array.pop() after this array=[“item1”];

Array Shift: Shift() method first element from array and shift remaining items to lower index

Const array= [“item1”,”item2”,”item3”]

array.shift();

After this array=[“item2”,”item3”];

Javascript Arrays Method

Array unshift: Add item to array at beginning of array
Let array=[“item1”,”item2”];
array.unshift(“item3”);
So now array=[“item3”,”item1”,”item2”]
Get Item of Specific index: To get specific index item use
Let array=[“item1”,”item2”,”item3”];
array[1]=item2;
Delete Keyword: To delete item from specific index
Const array=[“item1”,”item2”,”item3”];
delete array[0] will delete item1 from array and place undefined at that place

Javascript Arrays Method

Array splice: delete keyword delete item but place undefined at that place how to avoid this situation? For this we use splice() method.

Let array=[“item1”,”item2”,”item3”];

array.splice(0,1) will results array=[“item2”,”item3”];

Array Sort: Sort array items

Let fruits=[“banana”,”kiwi”,”Apple”];

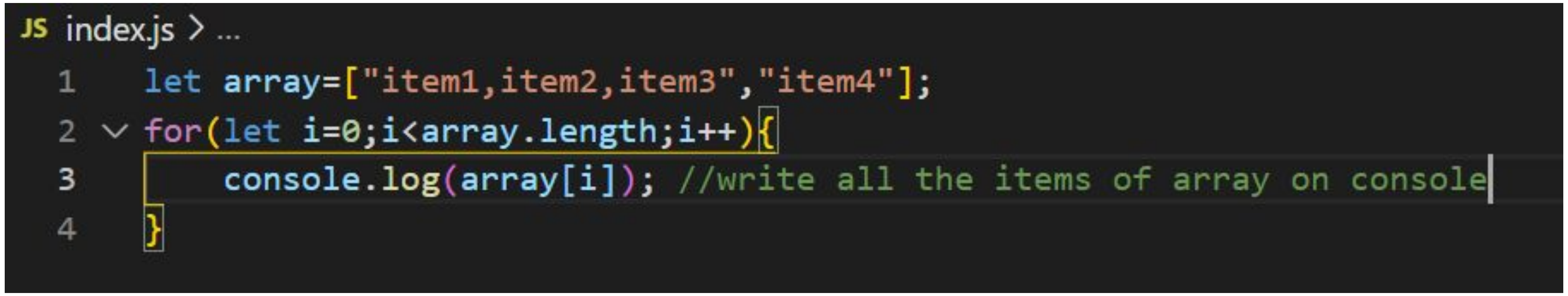
fruits.sort() will results fruits=[“Apple”,”banana”,”kiwi”];

Javascript Loops

We can iterate arrays in javascript by using loops

- For loop
- While loop
- Foreach loop

For loop:

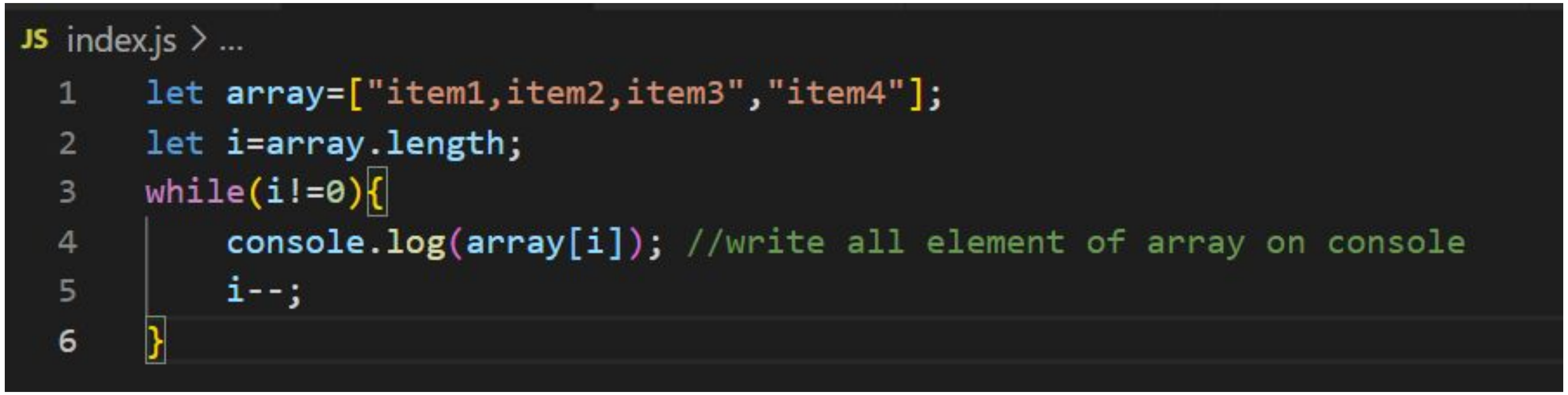


index.js > ...

```
let array=["item1,item2,item3","item4"];
for(let i=0;i<array.length;i++){
    console.log(array[i]); //write all the items of array on console
}
```

While loop:

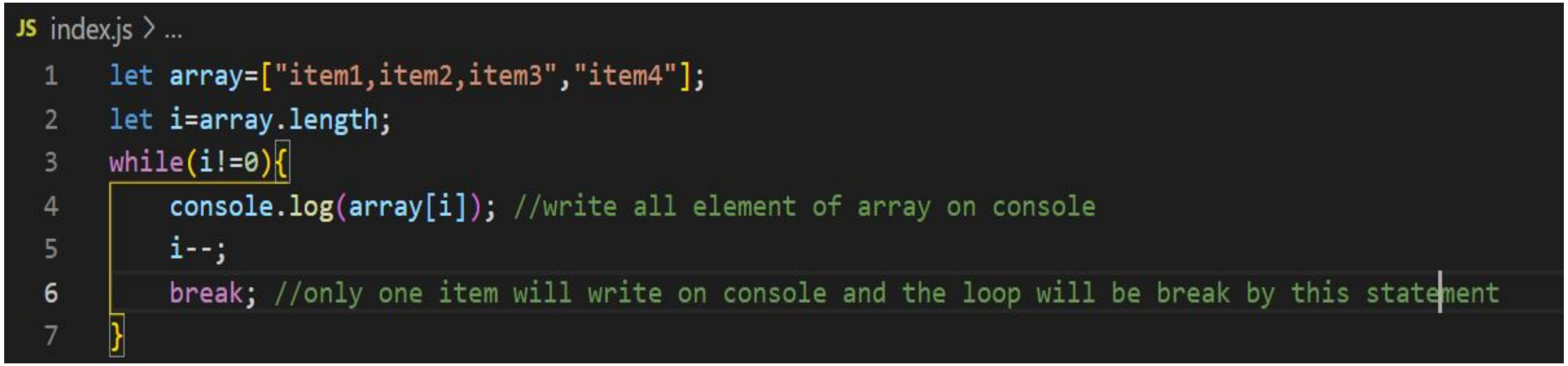
Conditional loop executes until condition not fulfills



index.js > ...

```
let array=["item1,item2,item3","item4"];
let i=array.length;
while(i!=0){
    console.log(array[i]); //write all element of array on console
    i--;
}
```

We can exist loops by using break statement.

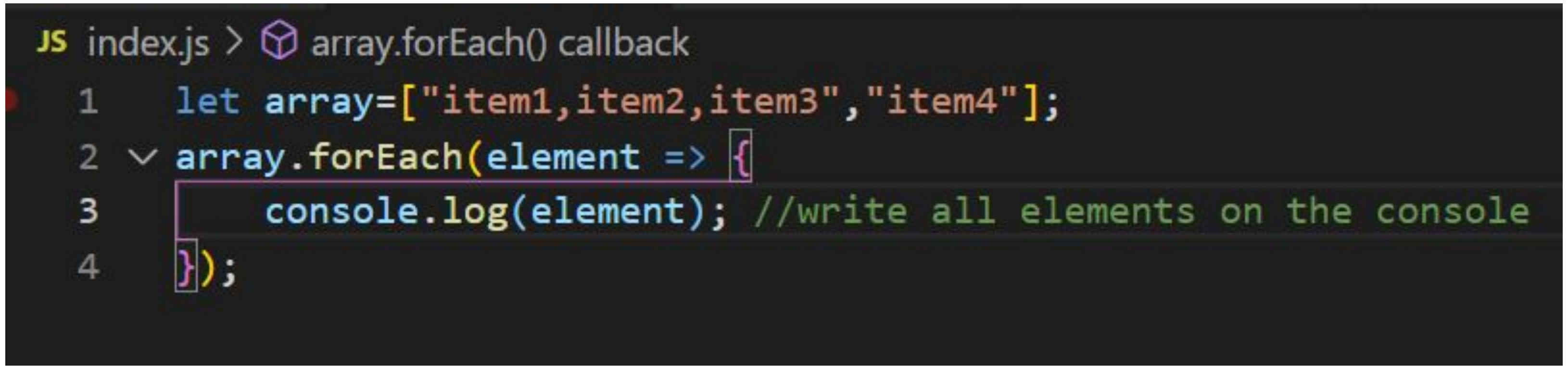


index.js > ...

```
let array=["item1,item2,item3","item4"];
let i=array.length;
while(i!=0){
    console.log(array[i]); //write all element of array on console
    i--;
    break; //only one item will write on console and the loop will be break by this statement
}
```

Foreach loop:

Foreach loop executes for each item present in the loop



index.js > array.forEach() callback

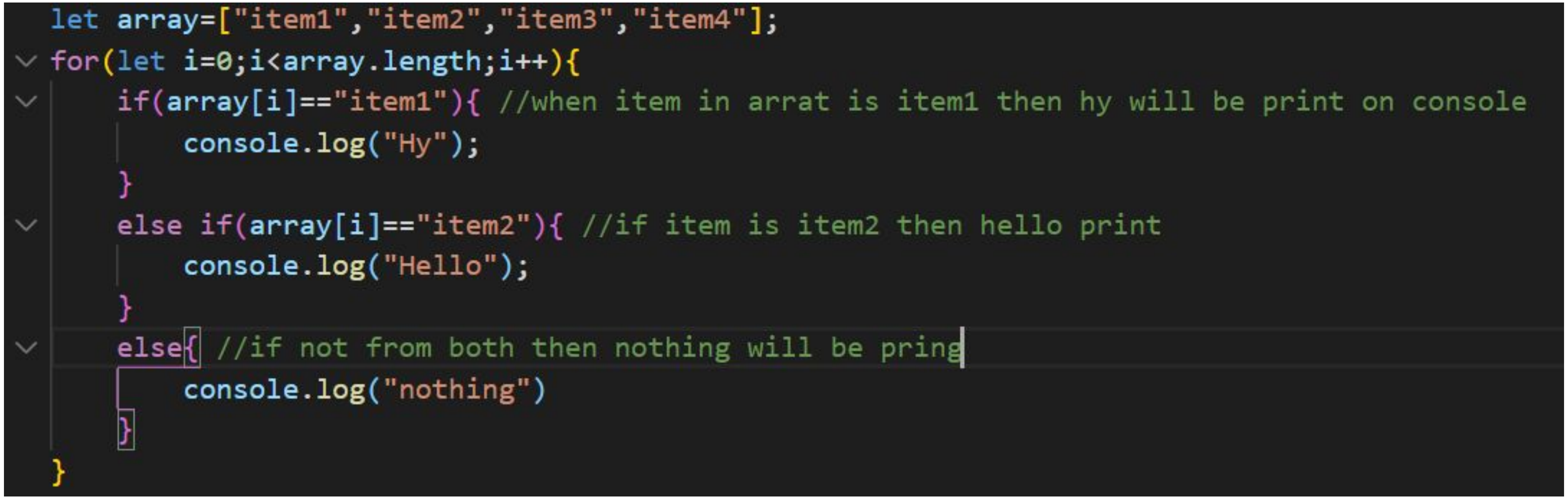
```javascript
let array=["item1,item2,item3","item4"];
array.forEach(element => {
    console.log(element); //write all elements on the console
});
```

Javascript If-else Statements

Conditional statements are used to perform different actions based on different conditions.

- Use **if** to specify a block of code to be executed, if a specified condition is true
- Use **else** to specify a block of code to be executed, if the same condition is false
- Use **else if** to specify a new condition to test, if the first condition is false
- Use **switch** to specify many alternative blocks of code to be executed

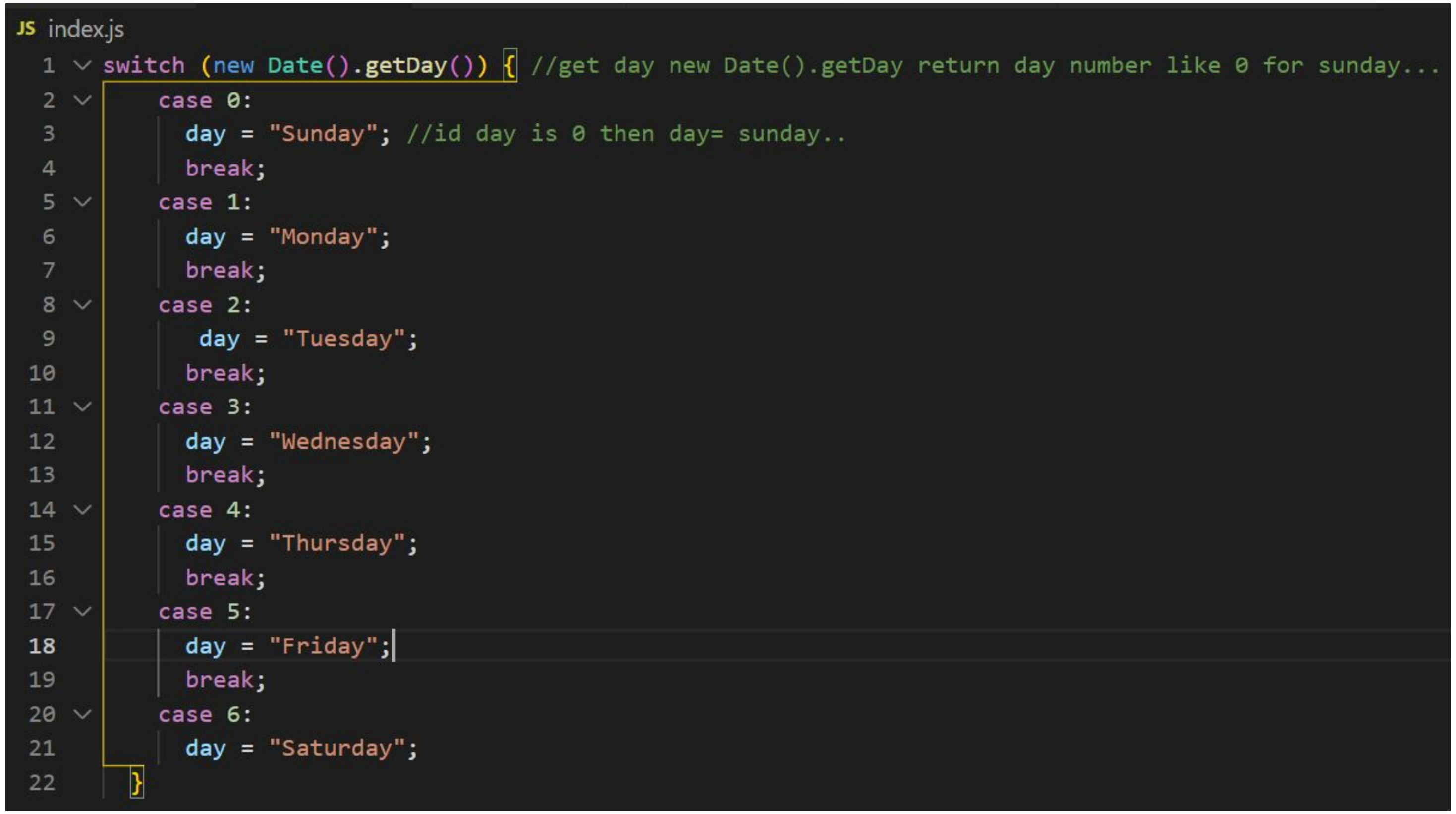
Javascript If-else Statements



```
let array=["item1","item2","item3","item4"];
for(let i=0;i<array.length;i++){
    if(array[i]=="item1"){ //when item in arrat is item1 then hy will be print on console
        console.log("Hy");
    }
    else if(array[i]=="item2"){ //if item is item2 then hello print
        console.log("Hello");
    }
    else{ //if not from both then nothing will be pring
        console.log("nothing")
    }
}
```

Javascript Switches

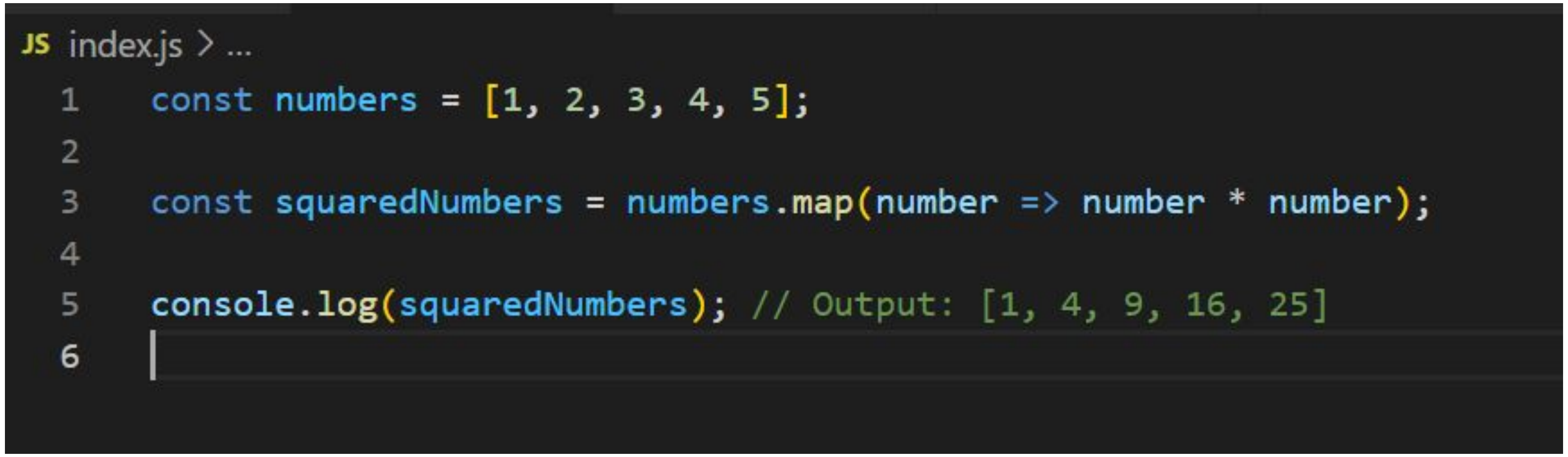
Use the switch statement to select one of many code blocks to be executed.


index.js

```javascript
switch (new Date().getDay()) { //get day new Date().getDay return day number like 0 for sunday...
    case 0:
      day = "Sunday"; //id day is 0 then day= sunday..
      break;
    case 1:
      day = "Monday";
      break;
    case 2:
       day = "Tuesday";
      break;
    case 3:
      day = "Wednesday";
      break;
    case 4:
      day = "Thursday";
      break;
    case 5:
      day = "Friday";
      break;
    case 6:
      day = "Saturday";
  }
```

Javascript Map method for array

The map() method is a built-in JavaScript array method that allows you to iterate over an array and create a new array based on the values of the original array. It executes a provided function on each element of the array and returns a new array with the results.



```javascript
const numbers = [1, 2, 3, 4, 5];

const squaredNumbers = numbers.map(number => number * number);

console.log(squaredNumbers); // Output: [1, 4, 9, 16, 25]

```

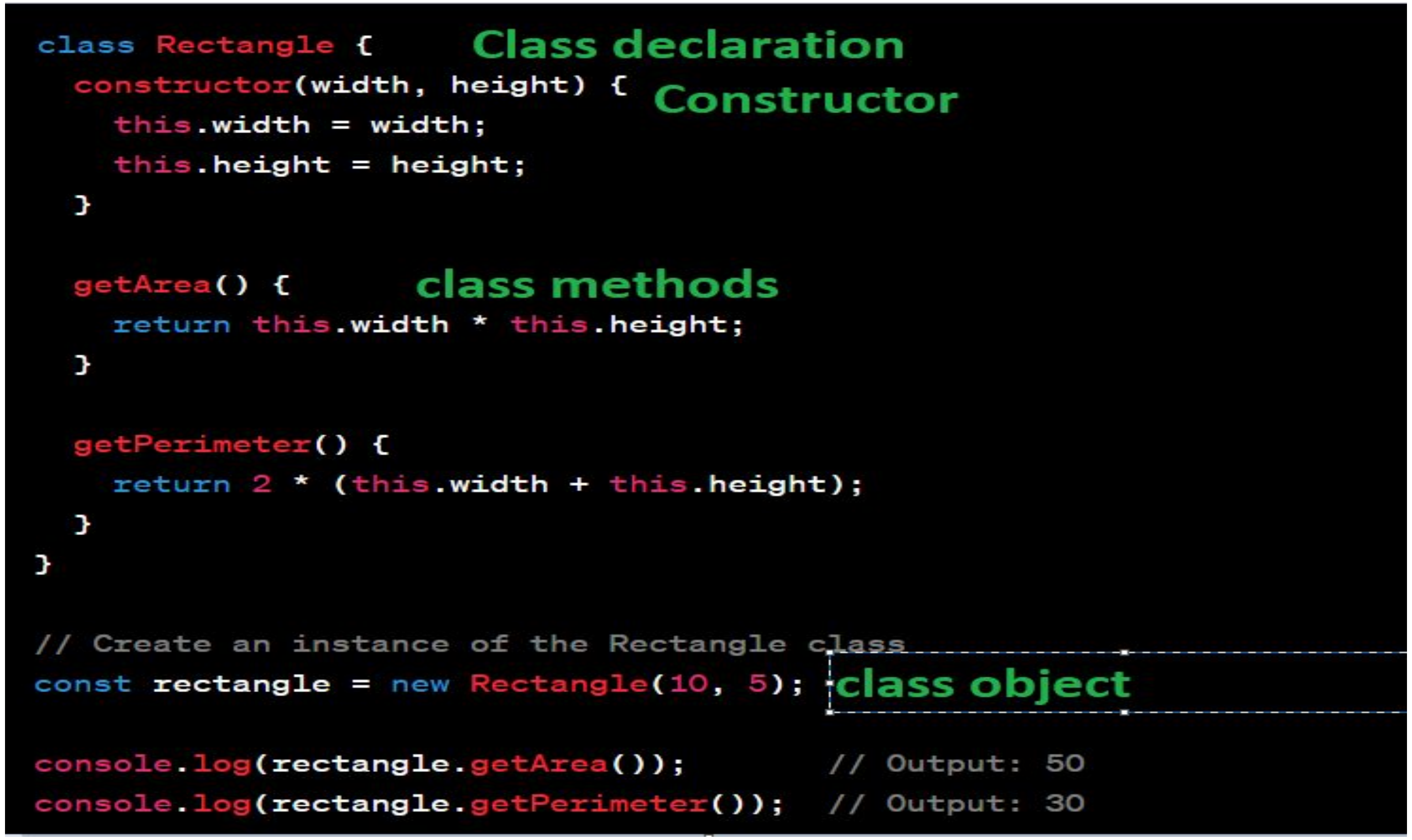
Converting one datatype to other

Method	Description
Number()	Returns a number, converted from its argument
parseFloat()	Parses a string and returns a floating point number
parseInt()	Parses a string and returns an integer

Classes in JS

In JavaScript, classes provide a way to define reusable object blueprints or templates for creating objects with similar properties and behaviors. They allow you to create objects based on a class, which serves as a blueprint for those objects. Classes are introduced in ECMAScript 2015 (ES6) and provide a more structured approach to object-oriented programming in JavaScript.

Classes in JS



```
class Rectangle {
  constructor(width, height) {
    this.width = width;
    this.height = height;
  }

  getArea() {
    return this.width * this.height;
  }

  getPerimeter() {
    return 2 * (this.width + this.height);
  }
}

// Create an instance of the Rectangle class
const rectangle = new Rectangle(10, 5);

console.log(rectangle.getArea());        // Output: 50
console.log(rectangle.getPerimeter());   // Output: 30
```

Class Terms

Constructor:

In JavaScript classes, the constructor method is a special method that is automatically called when an object is created from a class. It is used to initialize the object's properties and perform any necessary setup or configuration.

Class method:

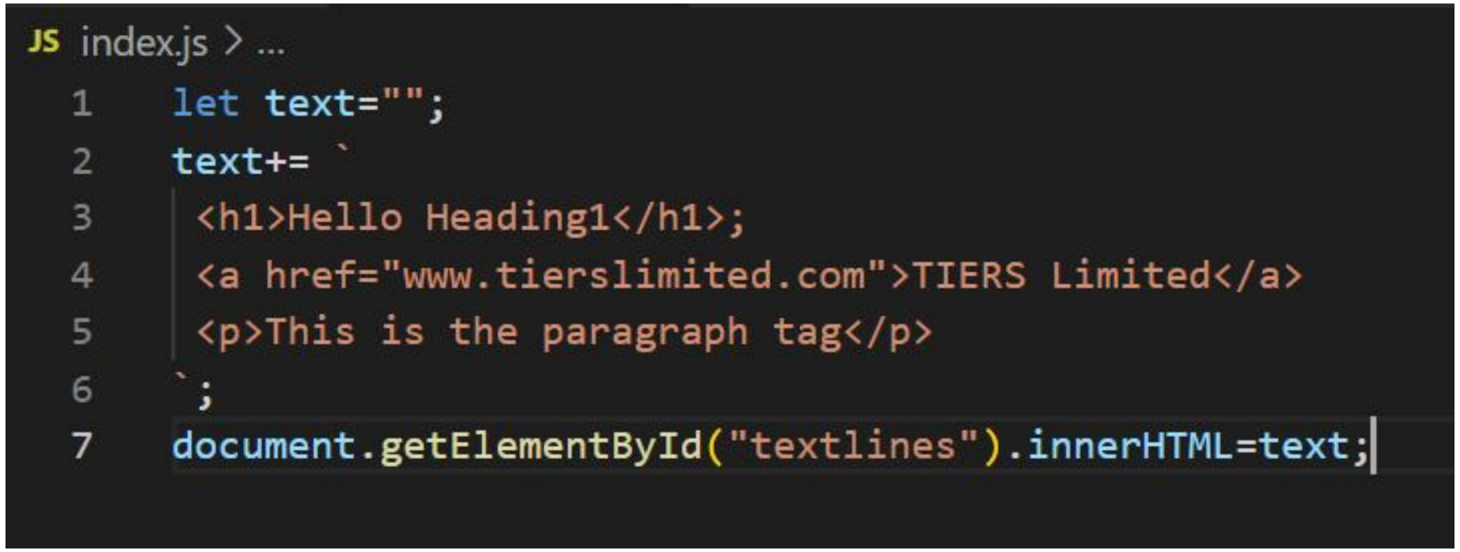
In JavaScript classes, you can define methods to add behavior to objects created from the class. Methods are functions that are associated with the class and can be called on instances of the class.

Class Object:

In JavaScript, objects created from a class are instances of that class. When you create a new instance of a class, you are creating a new object that inherits the properties and methods defined in the class.

How to write HTML in JS?

To write HTML in Javascript we can use backticks (``). Here is the code snippet



index.js > ...

```
let text="";
text+= `
  <h1>Hello Heading1</h1>;
  <a href="www.tierslimited.com">TIERS Limited</a>
  <p>This is the paragraph tag</p>
`;
document.getElementById("textlines").innerHTML=text;
```

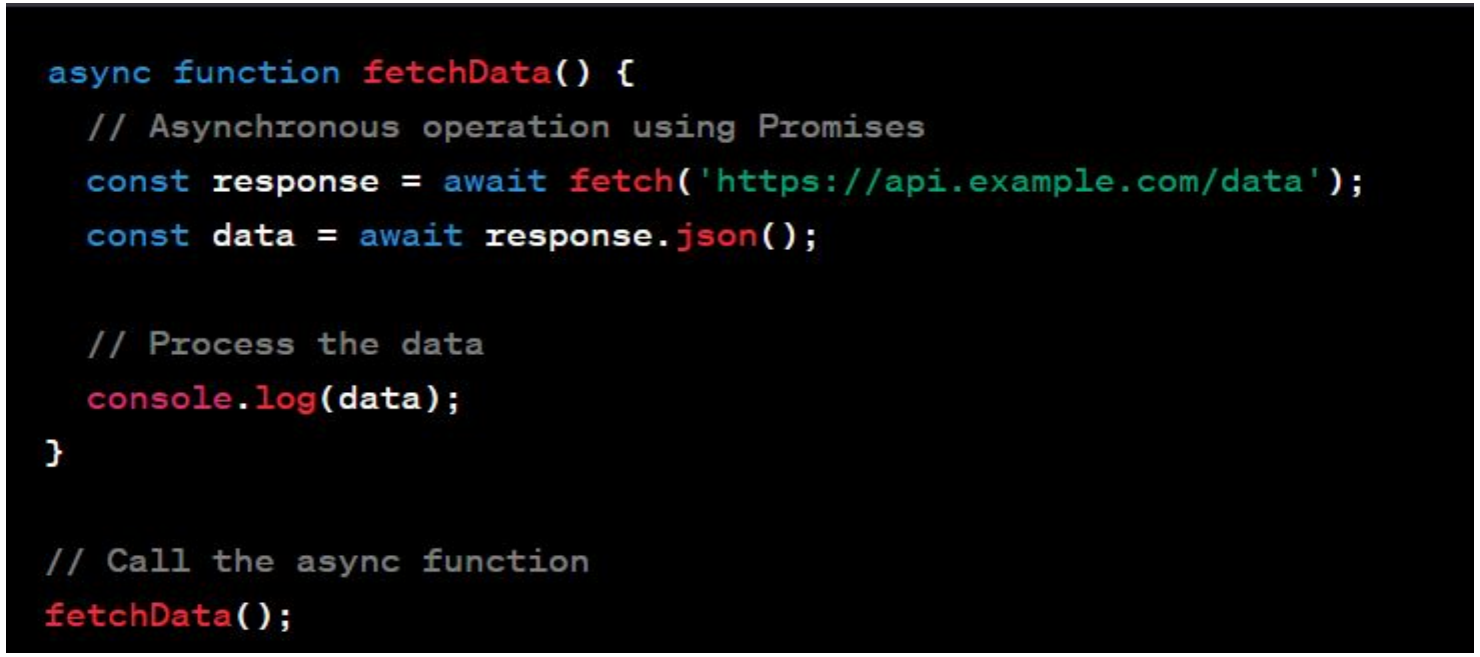
We can use this in dynamic website where we have to show data dynamically.

Javascript async functions

In JavaScript, async functions provide a convenient way to write asynchronous code that looks and behaves more like synchronous code. They allow you to work with Promises in a more intuitive manner and simplify the process of handling asynchronous operations.

To define an async function, you use the async keyword before the function declaration.

Here is an example of fetching api data using async function



```
async function fetchData() {
  // Asynchronous operation using Promises
  const response = await fetch('https://api.example.com/data');
  const data = await response.json();

  // Process the data
  console.log(data);
}

// Call the async function
fetchData();
```

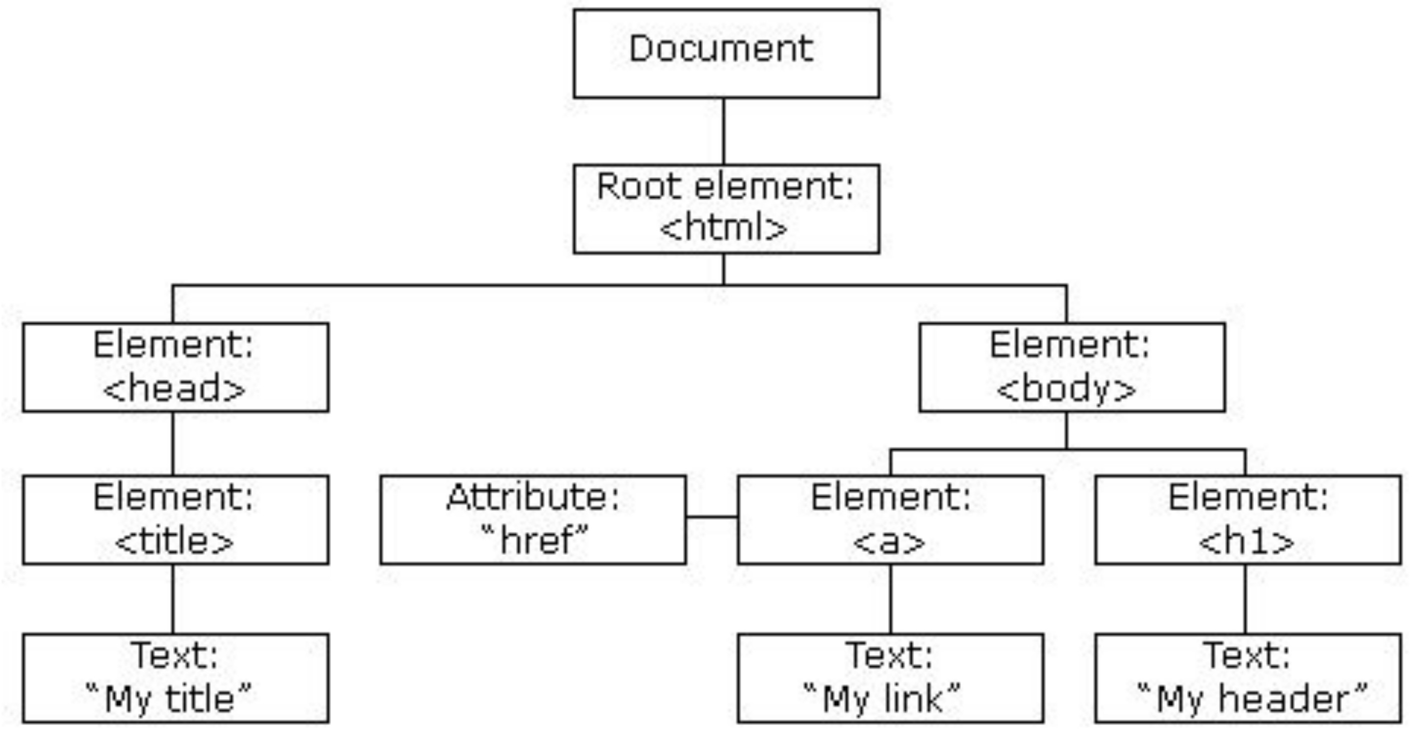
DOM (Document Object Model)

Introduction of DOM:

In JavaScript, the Document Object Model (DOM) is a programming interface that represents the structure of an HTML or XML document as a tree-like structure. It provides methods and properties to access, manipulate, and interact with the elements and content of a web page.

The DOM represents an HTML or XML document as a hierarchical tree structure, where each element, attribute, and text node in the document is represented as an object. These objects can be accessed and manipulated using JavaScript to dynamically modify the content, structure, and styling of a web page. We can get the element by using document.getElementById() etc methods.

DOM (Document Object Model)



DOM (Document Object Model)

Using the DOM, you can perform various operations, such as:

- Accessing and modifying HTML elements and their attributes.
- Manipulating the content and structure of the document, such as adding, removing, or modifying elements.
- Responding to user interactions and events, such as clicking a button or submitting a form.
- Changing the styling and appearance of elements, such as modifying CSS properties.
- Dynamically loading and manipulating data from external sources, such as making AJAX requests and updating the page content.

JavaScript provides built-in objects and methods to interact with the DOM, such as document for accessing the document object, getElementById() for selecting elements by their ID, querySelector() for selecting elements using CSS selectors, and many more.

LocalStorage using Javascript

In JavaScript, the localStorage object provides a way to store data in the web browser's local storage. Local storage allows you to persistently store data on the client-side, meaning the data will still be available even if the user closes the browser or navigates away from the page.

The localStorage object provides a simple key-value storage mechanism. Here's an example of how to use localStorage to store and retrieve data:

To store data in localStorage, you can use the setItem() method, which takes a key and a value as parameters:

```
localStorage.setItem('username', ‘abc’);
```

To get this value we use

```
localStorage.getItem(‘username’);
```

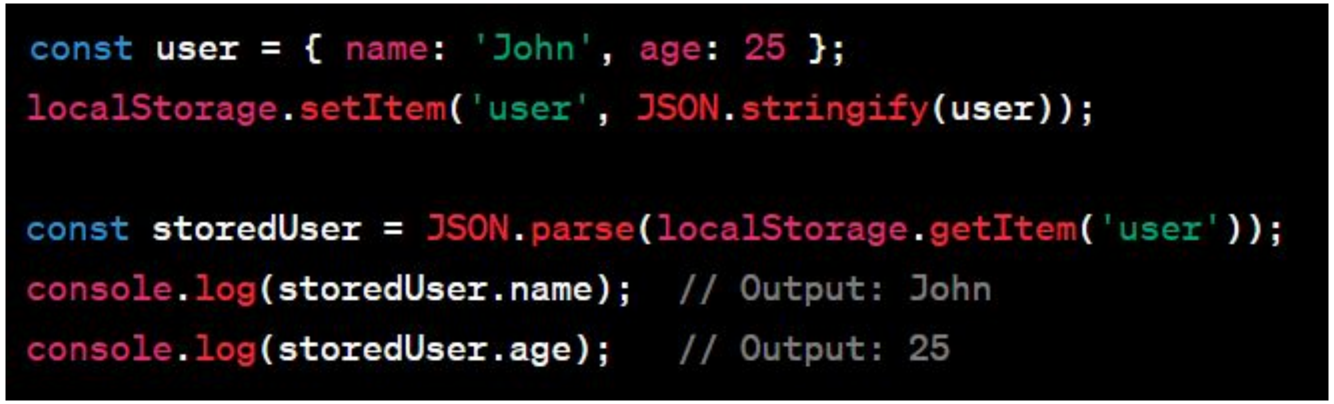
It will return abc.

To remove item from localstorage we can use

```
localStorage.removeItem('username');
```

LocalStorage using Javascript

you can use localStorage to store other JavaScript data types, such as objects or arrays. To do so, you need to convert the data to a string using JSON.stringify() before storing it, and then parse it back to its original format using JSON.parse() when retrieving it from localStorage.



```javascript
const user = { name: 'John', age: 25 };
localStorage.setItem('user', JSON.stringify(user));

const storedUser = JSON.parse(localStorage.getItem('user'));
console.log(storedUser.name);  // Output: John
console.log(storedUser.age);   // Output: 25
```

Class # 2 Task

The task # 2 is same as we have done in previous class. Now we have to store the aggregate of each student on localStorage and there is a new page in html where user can view the name and aggregate of all the students who calculate their aggregate.