React.js

* HTML
* CSS
* Javascript

HTML: It is mainly to display the content

CSS: It is to style the HTML

Javascript: It is to add effects to the web page by changing HTML & CSS at runtime

Software requirement

1. VS Code
2. Live server plugin: To auto-reload the changes and give a live preview

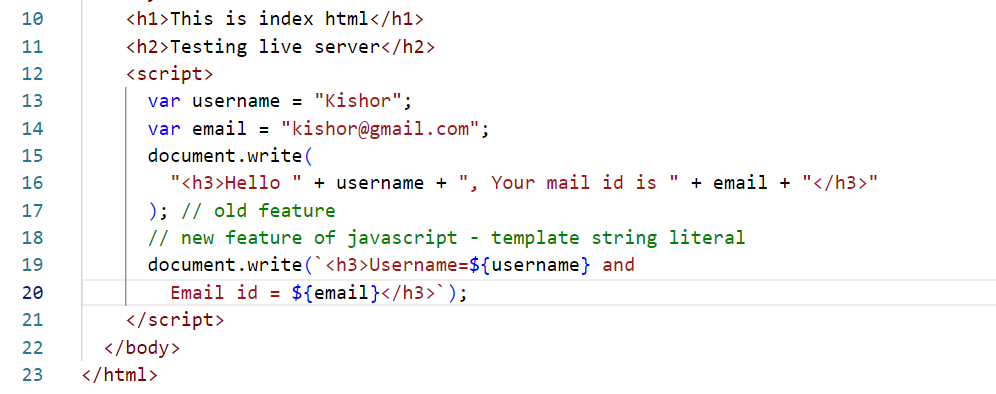
New Features of ES6 or Javascript

Template String literal:

It avoids you to break the string i.e., using + operator to concatenate string and the expression

var name = “…”;  
var email = “…”;

`<h3>Your name is ${name}, email id ${email}</h3>`



Classes & Constructors to create object

ES6 introduced class & constructor keyword to create the objects, earlier developers had to use functions as a constructor to create the object.

Old approach

function Employee(id, name, salary) {  
 this.id = id;  
 this.name = name;  
 this.salary = salary;  
}

var obj = new Employee(123, “Raj”, 45000);

obj.id, obj.name, obj.salary will access the properties

Suppose you want to add a function to the object, then you must use prototype.

Employee.prototype.display = function() {   
 document.write(….);  
}

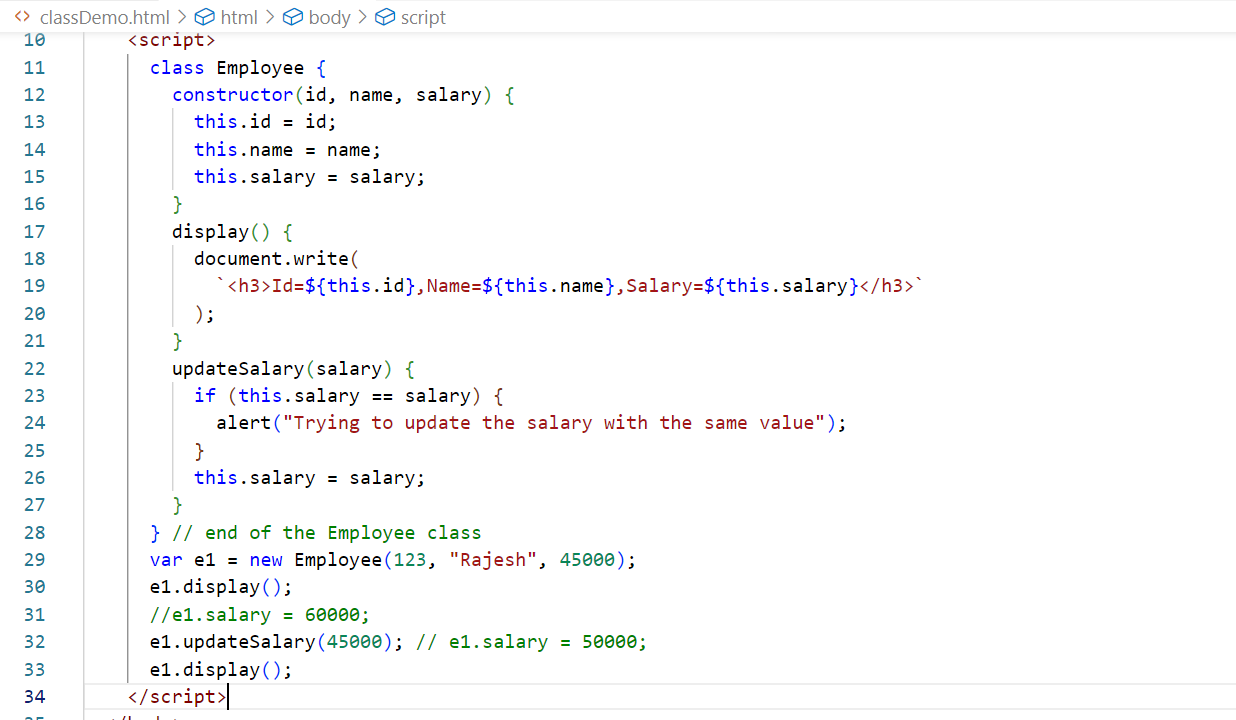
obj.display();

New approach: ES6 introduced classes & constructors to simplify creating objects

class Employee {   
 constructor( id, name, salary) {   
 this.id = id;   
 this.name = name;  
 this.salary = salary;  
 }  
 display() {   
 document.write(….);  
 }  
}

let obj = new Employee(2345, “Rajesh”, 42000);  
obj.display();

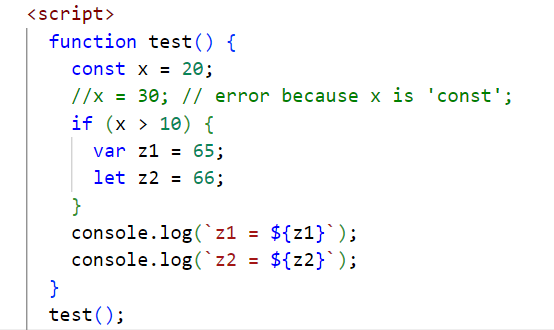
example



let & const keywords

They are used to create block scope variables, var was the keyword which was earlier used to create variables it is not a block scoped variable, it must be avoided

if(…) {  
 var cal = x + y;  
 let cal2 = x + y;  
 const cal3 = x + y;  
}  
you can access cal variable outside the if block



Event Handling: This was there even before ES6, it is used to detect the element generated the event & perform some action (executing some code)

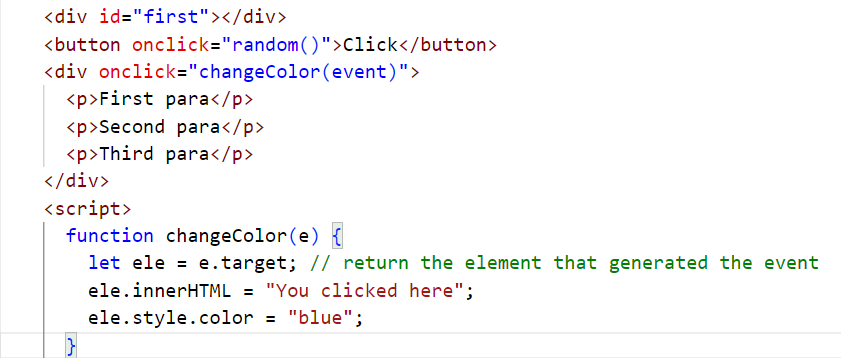
In Javascript an event is referenced using an event property, list of events that an element can generate

1. onsubmit
2. onclick
3. onmouseover
4. onmouseout
5. onfocus
6. onblur
7. onchange

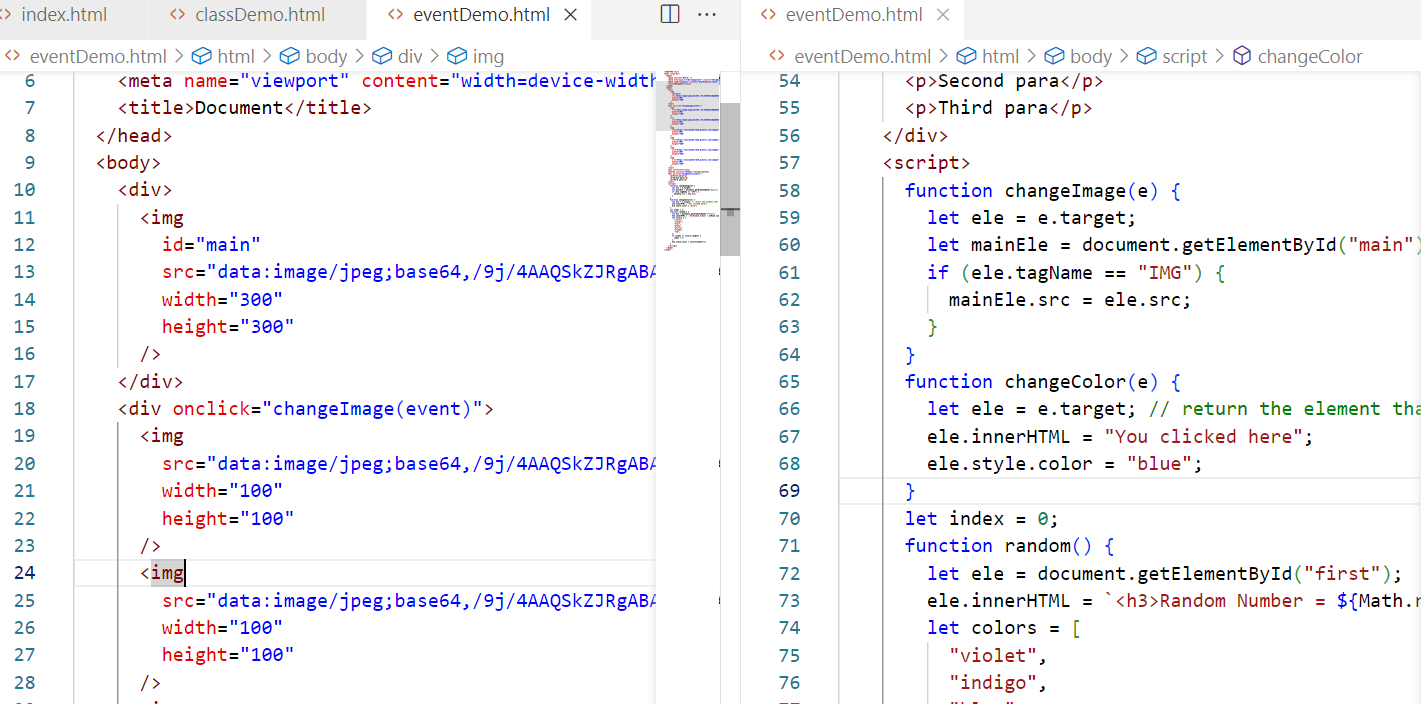


Accessing the HTML element using event object

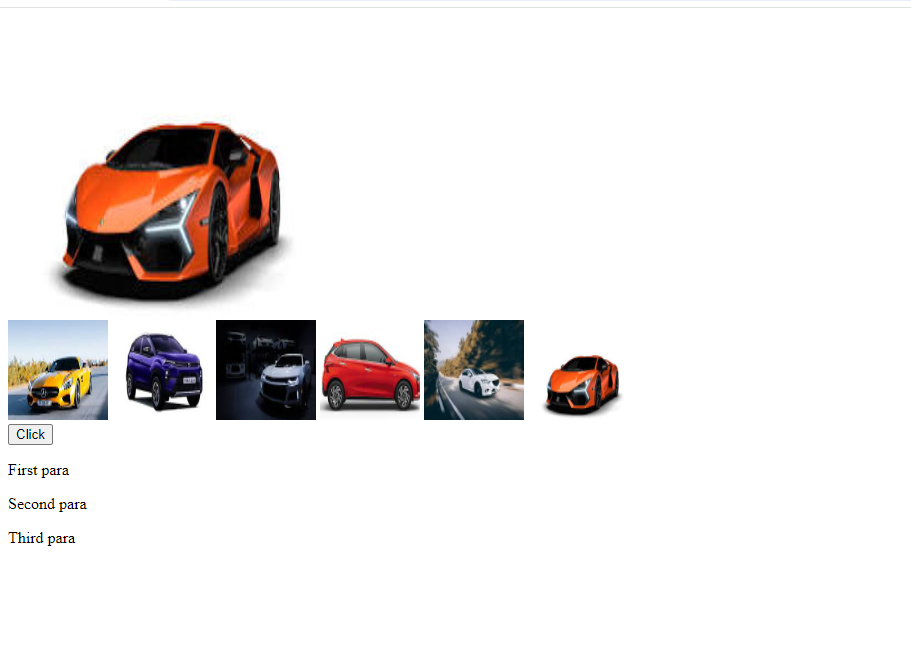
Every HTML element that generate the event can be identified by using the event.target property



Accessing the attributes using the events



Output:



Callback functions: These are the functions that are initiated now but executed later, i.e., when certain event occurs or when a response arrives

Callback functions are generally passed as a parameter to another function

ex: when you iterate an array you can use some inbuilt functions like forEach(), map(), sort() and so on.

let items = [ 20, 30, 10, 50, 40 ];

for(let index = 0; index < items.length; index++) { … } // traditional approach

You can also iterate using forEach function, which is part of an array

items.forEach( callbackFn(value, index){} )

Note: forEach automatically calls the callbackFn by passing the iterated value to the 1st parameter & index to the 2nd parameter

Note: You can give any name to the parameters.

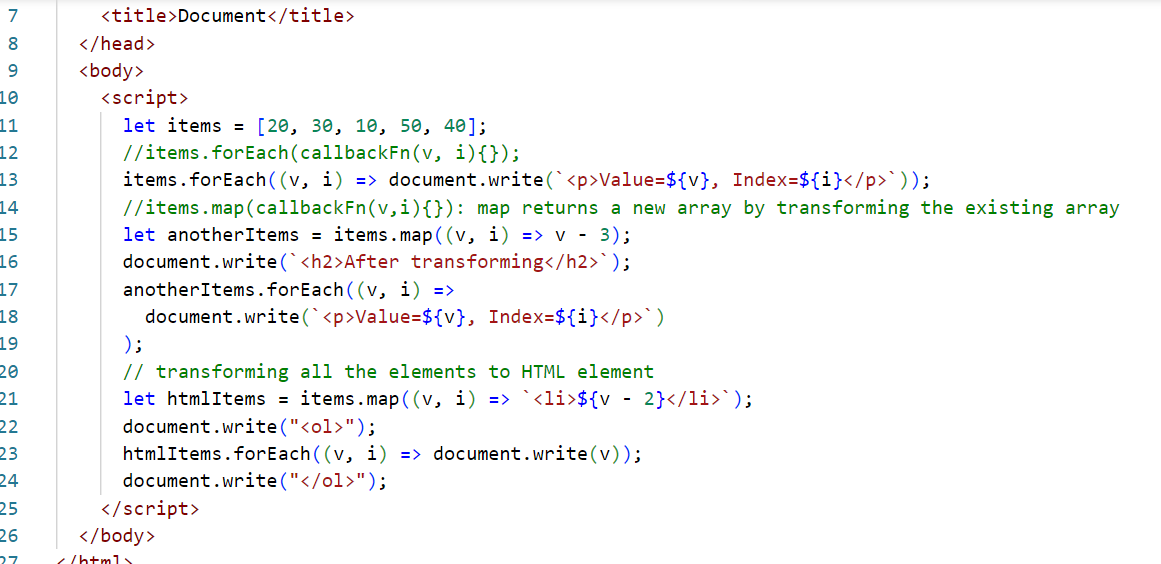
Arrow function:

It is introduced in ES6 to simplify writing callback functions, if the callback is only one line then no need to use { } or return if in case callback returns value, and also no need to use function keyword

Note: { } & return is required only if the callback has more than one line

|  |  |
| --- | --- |
| Callback | Arrow Function |
| function(v) { one-statment } | (v) => one-statement  [or]  v => one-statement |
| function(v) { return value; } | v => value; |
| function(x, y) { return x + y; } | (x, y) => x + y;  [or]  (x, y) => { return x + y; } |
| function(x, y) { console.log(x+y) } | (x, y) => console.log(x+y); |
| function(x, y) { 1st line; 2nd line } | (x, y) => {1st line; 2nd line } |
| function(x, y) { 1st line; 2nd line; return value } | (x, y) => {1st line; 2nd line; return value; } |
| ex: forEach ( function(v, i) { } ) | ex: forEach( (v, i) => { } ) |
| ex: map(function(v, i) { return v + 2; } | ex: map( (v, i) => v + 2 ); |

Replacing all the callbacks to arrow function



Activity:

Assuming you have an array of objects, print those items in an HTML table, below is the array  
Input:

[

{id:1, name:”Vijay”, address : {state:”KA”, city:”BLR”, pin:560001 } },

{id:2, name:”Ajay”, address : {state:”KA”, city:”MYS”, pin:560061 } },

{id:3, name:”Ravi”, address : {state:”MH”, city:”MBI”, pin:760001 } },

{id:4, name:”Siddharth”, address : {state:”TN”, city:”CHN”, pin:660001 } }

]

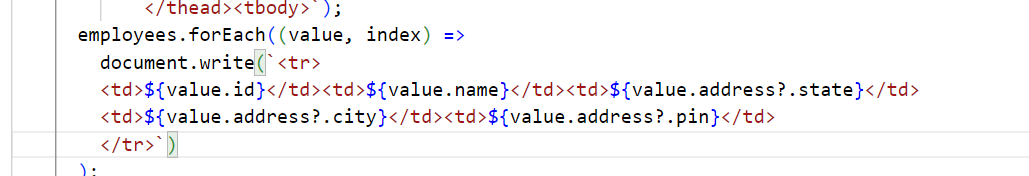
Output:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | Name | State | City | Pin |
| 1 | Vijay | KA | BLR | 560001 |
| 2 | Ajay | … | .. | .. |
| … | .. | … | .. | … |
| …. | .. | . | .. | . |

TypeError: Whenever we try to access the nested properties there could be chance that nested properties may not be present, in that case you get type-error, to avoid this developers used if(value.address != undefined) { then access value.address.state }

Optional Chain(?.): ES8 added new feature called Optional chain that will access the nested property only if the property exists

value.address?.state // if address is not undefined then access state.



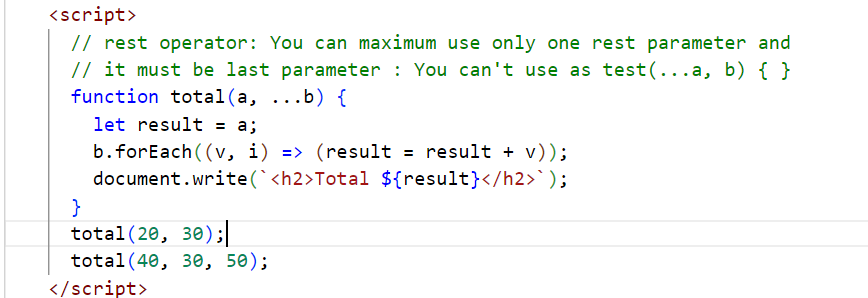
Rest & Spread operators

Rest operator: When a function needs to accept an optional parameter or 0 or more parameters, then you can use rest parameter

Earlier data’s were lost when there are no parameters to accept some values

ex: function test(x, y) { }   
now you can call test by passing only 2 parameters, but if you pass more than 2 parameters then all those data will be lost

function test(x, …y) { } // here y can accept 0 or more parameters, y also acts like an array



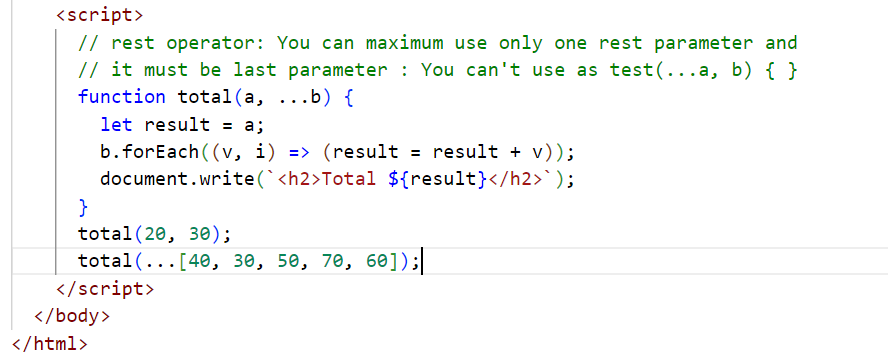
Spread operator: It is similar to rest parameter, however used to distribute the list of values to multiple parameter of a method, used while calling the method

function test(x, y, z) { }

let items = [20, 10, 30];  
test(items); // x = [20, 10, 30], y = undefined, z = undefined

// spread operator  
test(…items); // x=20, y=10, z=30;

Note: If the number of parameters in the function is less, then the last parameter must be the rest parameter, so that the last parameter accepts rest of the values



Destructuring: It is used to assign values of the array or an object to a separate variable in a single line.

Earlier:   
let items = [20, 10, 30, 50];

If each item needs to be assigned to separate variables

let a = items[0];   
let b = items[1];  
let c = items[2];  
let d = items[3];

let obj = { id : 1, name : “Raj”, salary: 45000 };  
let id = obj.id;  
let name = obj.name;  
let salary = obj.salary;

From ES6 onwards you can use destructuring

Array destructuring  
let items = [20, 10, 30, 50];

let [a, b, c, d] = items; // each item is assigned to the variables in the order they appear

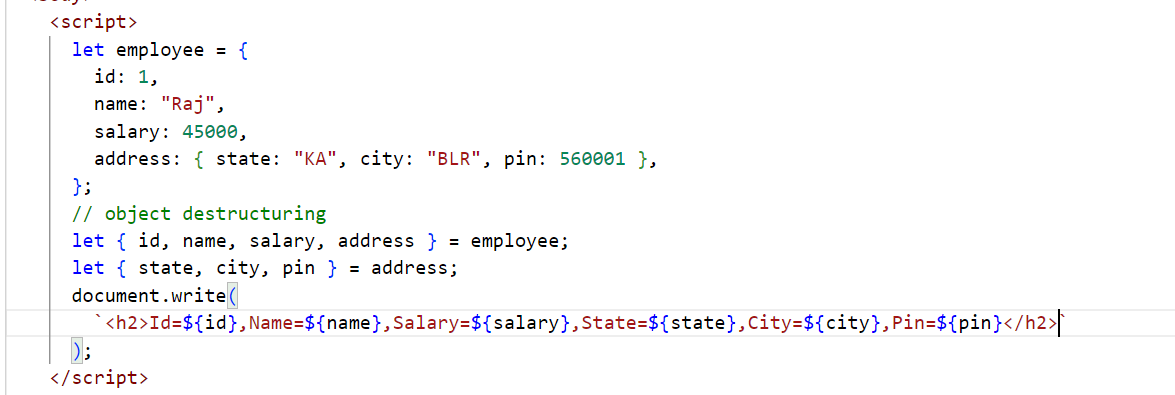
Object destructuring

let obj = { id : 1, name : “Raj”, salary: 45000 };  
let { id, name, salary } = obj;

function test([a, b, c, d]) // array destructuring  
test(items);

function display( {id, name, salary} ) { } // object destructuring

display( obj );



Default values: When a parameter doesn’t get any value this default value will be given

function test(x = 0, y = 0) {   
  
}  
test(); // x & y will be 0  
test(2); // x = 2, y = 0

How to access backend services from javascript

Whenever front-end sends the request to the backend services it gets the data in the form responses, there are few approaches javascript uses

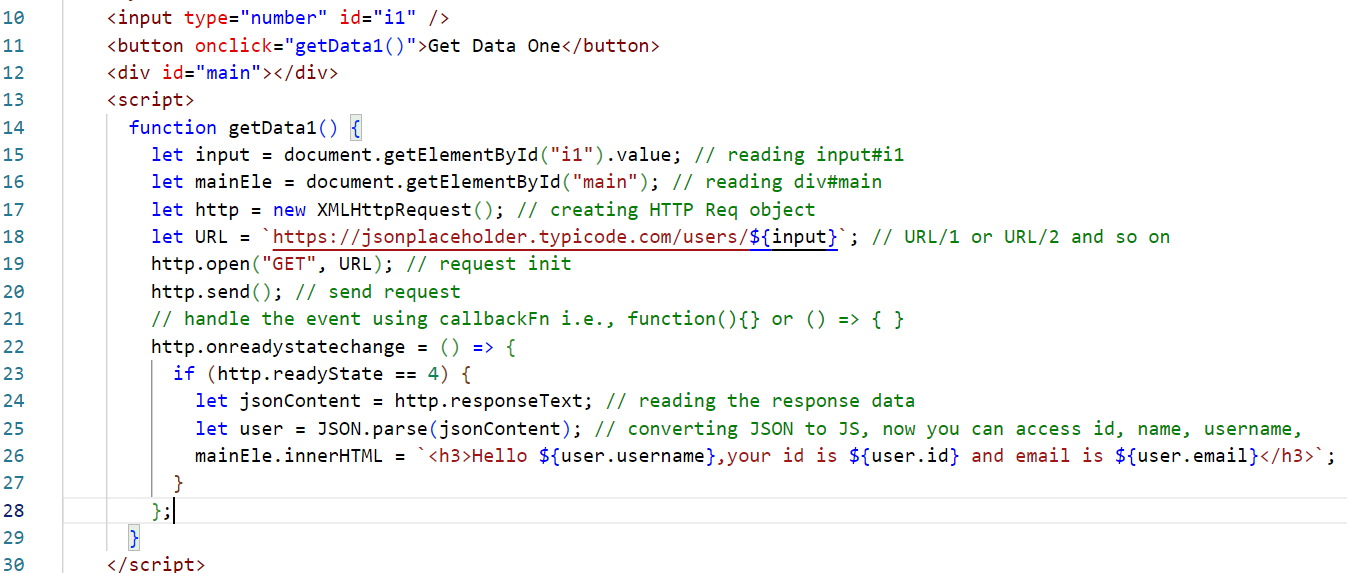
1. Old approach : XMLHttpRequest which is callback based
2. New approach: fetch which is Promise based
3. Enhancement to the Promise: async/await

XMLHttpRequest: It is an object used earlier to make HTTP calls(HTTP requests) to the backend, it gets the response through some events and invokes the callback based on those events, list of properties & functions in XMLHttpRequest

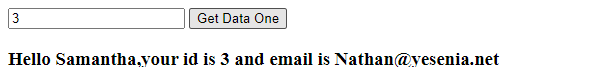
1. readyState: it maintains the value ranging from 1 to 4, if 1 then request is initialized, if 2 then request is sent, if 3 then half of the response is ready, if 4 then full response is ready means request is complete
2. responseText: it maintains the response data
3. onreadystatechange (everything in lowercase): this generates an event each time the readyState value changes & it invokes the callback attached to this
4. open(httpMethod, URL): to initialize the request with HTTP methods & URL
5. send(): to send the request

In order to send HTTP request to the json placeholder we must use HTTP GET method & an URL

<input id = “i1” > <button onclick=”getData()”>Get Data</button>  
function getData() {  
 let id = read the #i1.value  
 let url = “url/”+id;   
 let http = new XMLHttpRequest();  
 http.open(“GET”, url);  
 http.send();   
 http.onreadystatechange = callbackFn() { …when readState==4 read responseText. }  
}



Output:



Promise based:

Promise is an object that will have two status based on the request

1. Resolved: If the promise is fulfilled / success then it is treated as resolved
2. Rejected: If the promise is rejected/failed then it is treated as rejected

Promise provides two methods to handle success & failure, those are then(callback) and catch(callback)

then(callback): it is invoked when the promise successfully resolved

catch(callback): it is invoked when the promise is failed or when there’s any error

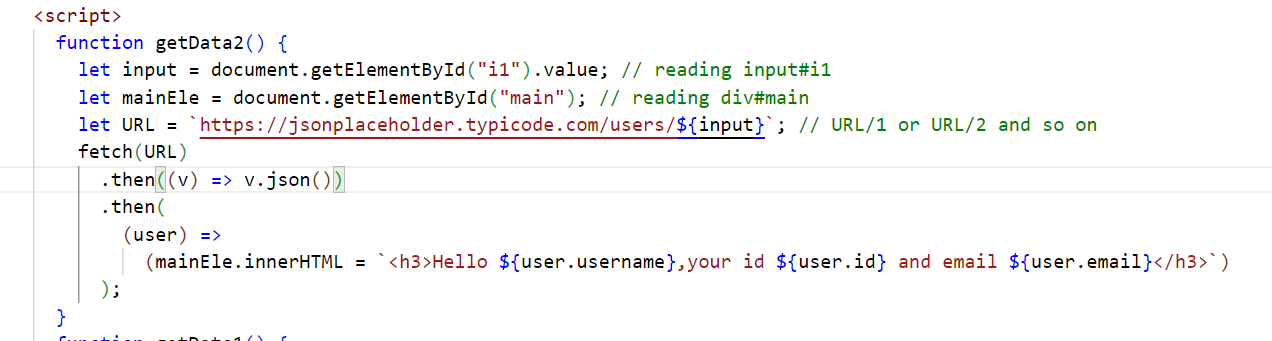
both of them takes callback which is executed based on the promise successful or reject status

fetch(URL): It is an inbuilt method that returns a Promise when the request is made, this can be resolved or rejected

fetch(URL).then( callbackFn ).catch( callbackFn )

getData2() {  
 let id = read the #i1.value  
 let url = “url/”+id;   
 fetch(url).then( (res) => { } ).catch( (err) => { } )

}



Async/Await: It is used whenever you need to wait for the asynchronous operations to complete and then you want to proceed

async function getData3() {  
 let res = await fetch(URL) // no need to use .then() or .catch  
 print res; // executed only if the Promise is either rejected/resolved  
}

