Basics of Javascript

Software’s requirement

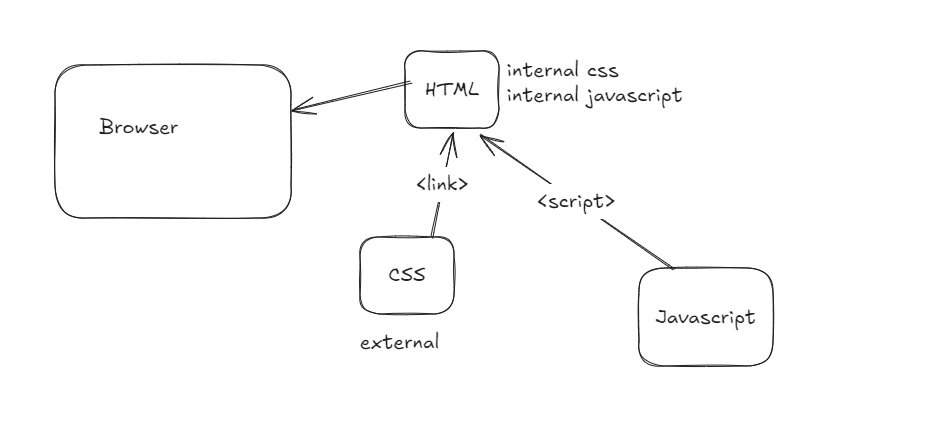
* VS Code editor
* Browser

Web technologies

* HTML: Display the content on the web page
* CSS: It is used to style the HTML elements
* Javascript: It allows you to add dynamic effects at runtime

Fundamentals of Javascript

* variables
* functions
* conditional constructs
* looping statements
* arrays
* objects
* event handling
* form validations
* DOM manipulations

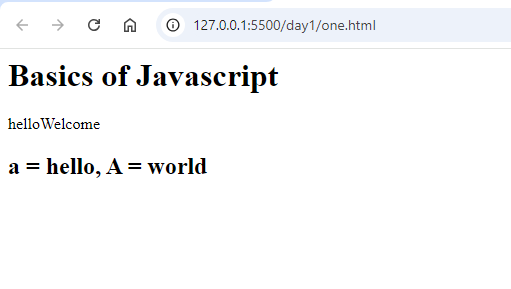


HTML is the file the browser loads, where you can have Javascript & CSS linked to the HTML or internally also you can have HTML & CSS

one.html



Output:



In Javascript you don’t have types for variables unlike other programming languages like Java, C, C++, C#, which means a variable can store any kind of value

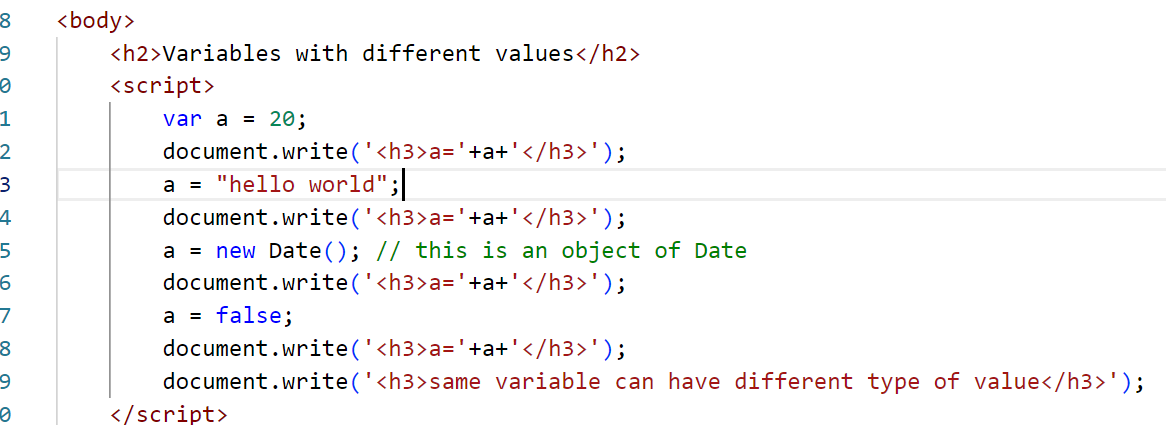
a = 10;  
a = “Hello”;  
a = true;

a = ‘x’; // it is still a string

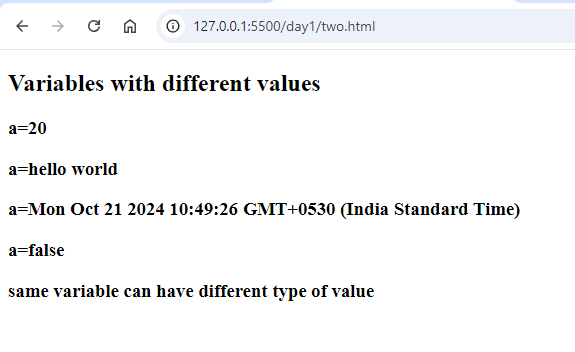
a = ‘hello world’;

a = new Date(); // a is storing a Date object

two.html



Output

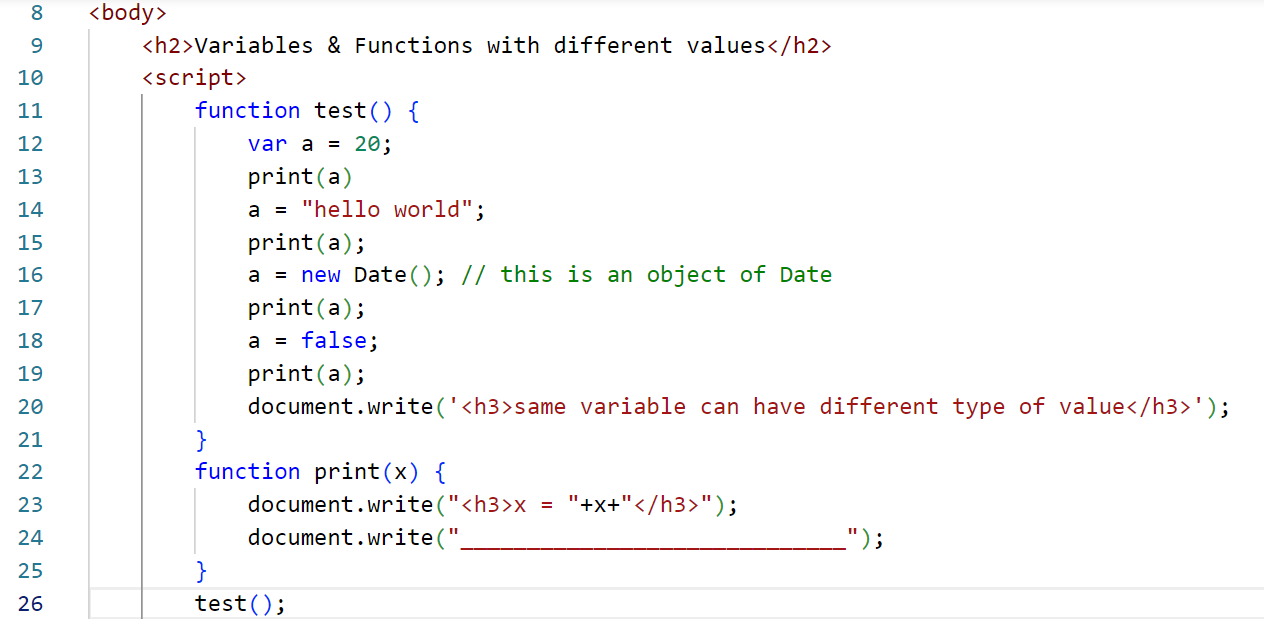


Functions: These are named blocks which are invoked by its name, they are used to create a reusable statements that can be called later

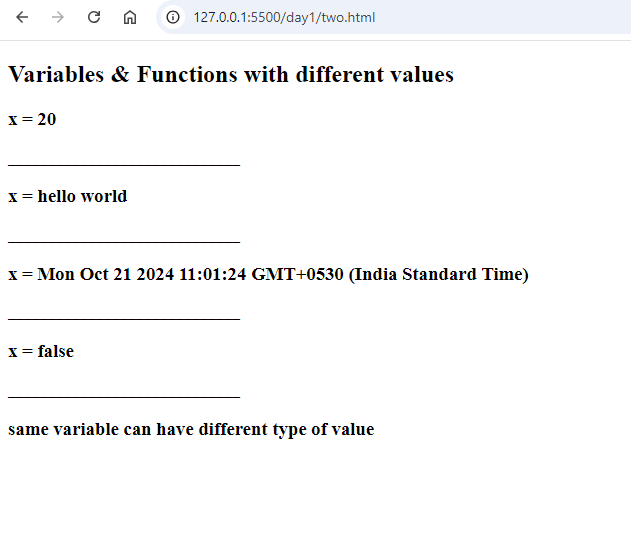
Syntax:

function function\_name(parameters, parameters) {   
 statements; // one or more lines of statements  
}  
Invoking function:

function\_name(arguments);



Output:



Predefined functions in Javascript

In Javascript there are some inbuilt objects like

* window
* document
* console
* Math
* Date
* String

window is the top level object whose function/properties you can access without using its name.

Object: It is a real world entity, it will have 2 things

* properties: variables
* functions: actions

window.alert(“creates an alert message);  
or  
alert(“creates an alert”)

window.parseInt(“”) or parseInt(“”)

window.parseFloat(“”) or parseFloat(“”)

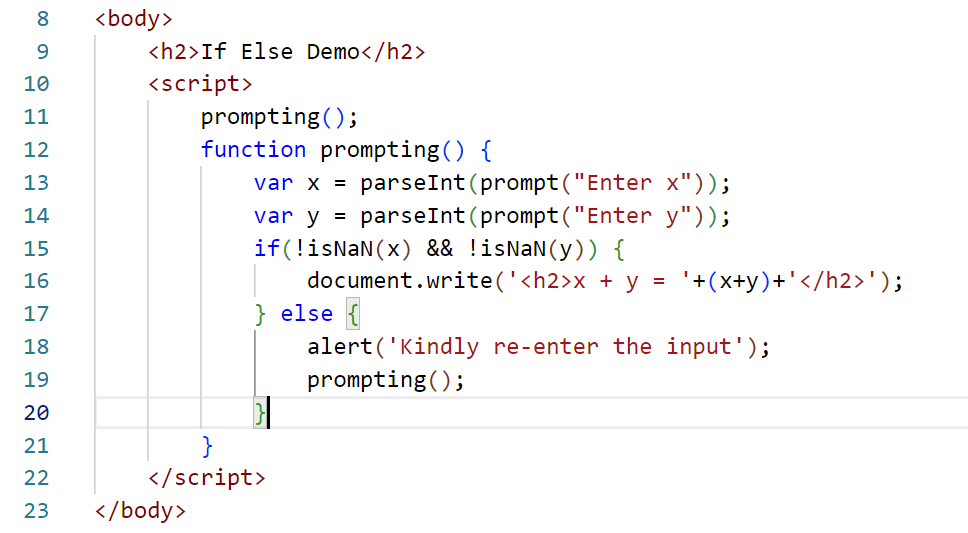
Conditional Statements

Javascript supports following conditional statements

1. if
2. if – else: if (….) { stmt; } else { stmt; }
3. if – else if else if … else
4. switch

Activity:

Using if & else, try to take 2 inputs that needs to add those inputs only if both the input’s are numbers, else prompt again to ask the input



There are two parse functions in Javascript

* parseInt
* parseFloat

parseInt(“12345.45”): It returns 12345  
parseFloat(“12345.45”): It returns 12345.45

Complex types

1. Arrays
2. Objects

Array: These are the variables that can store more than one value

Object: These are the variables that will have properties & functions

array is created using [ ] in Javascript

var items = [1, 2, 3, 4, 5];  
var names = [“Alex”, “Virat”, “Rohit”]  
object is created using { } in Javascript

var user = { userId: 134, name : “Virat”, age:38 };

Object Array is created using [{ }, { }, { } ]   
var users = [ {userId:123, name: “Virat”, age:38}, {userId: 456, name:”Rohit”, age:37}]

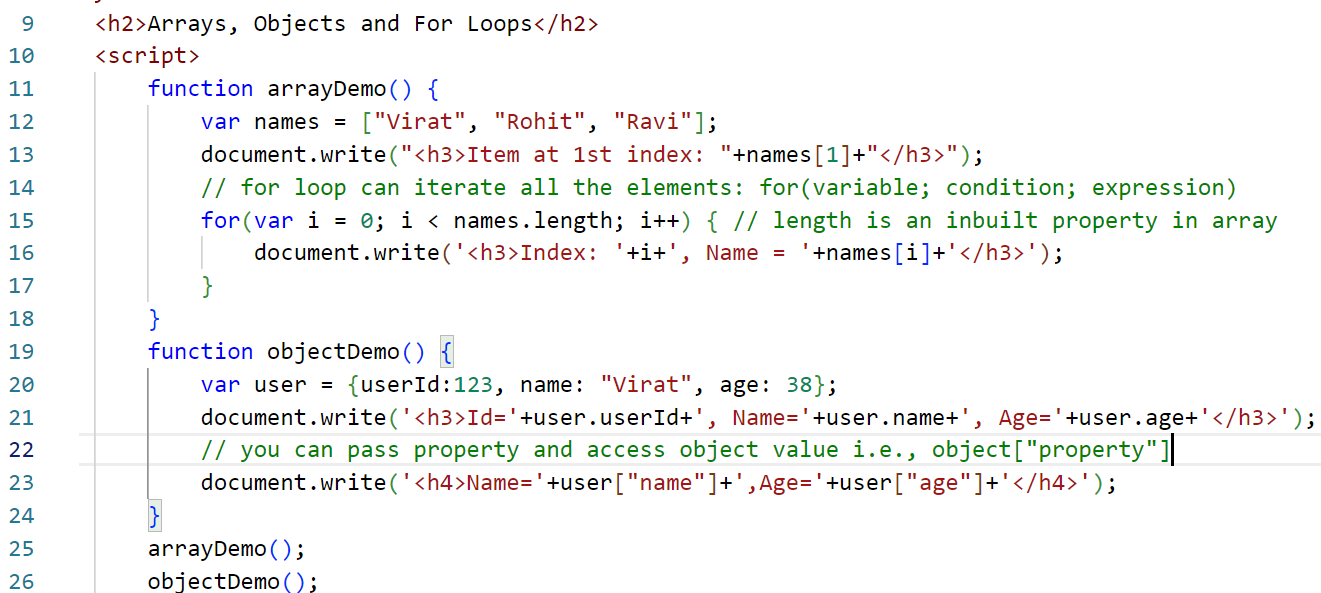
How to access an element in an array

items[0], items[1], items[2] or you can use for loop to iterate all the elements

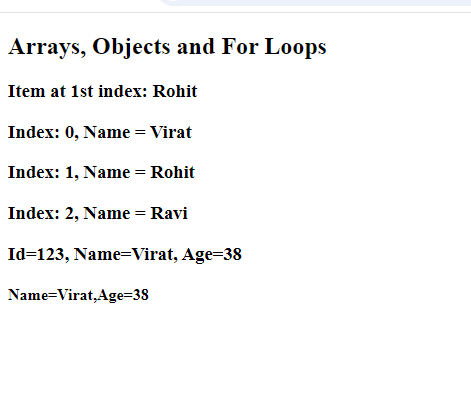
How to access object property

user.userId, user.name, user.age

users[0].userId, users[1].userId, users[2].userId ….



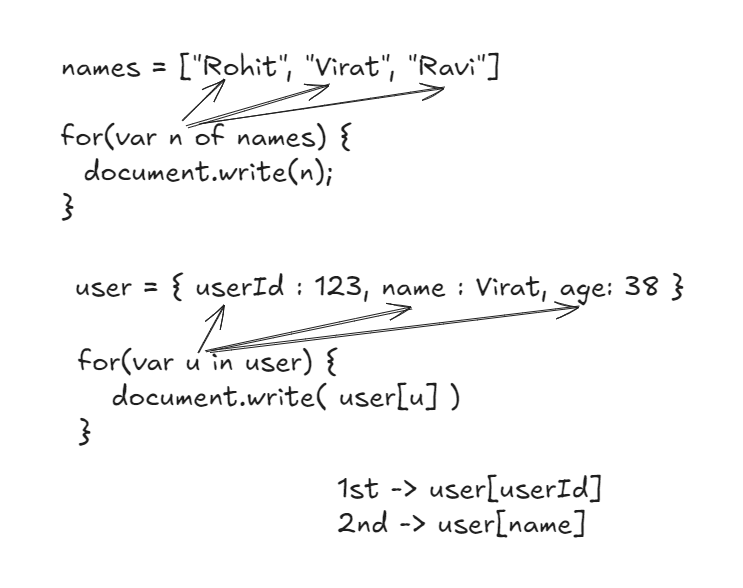
Output:



Javascript has three types of for loops

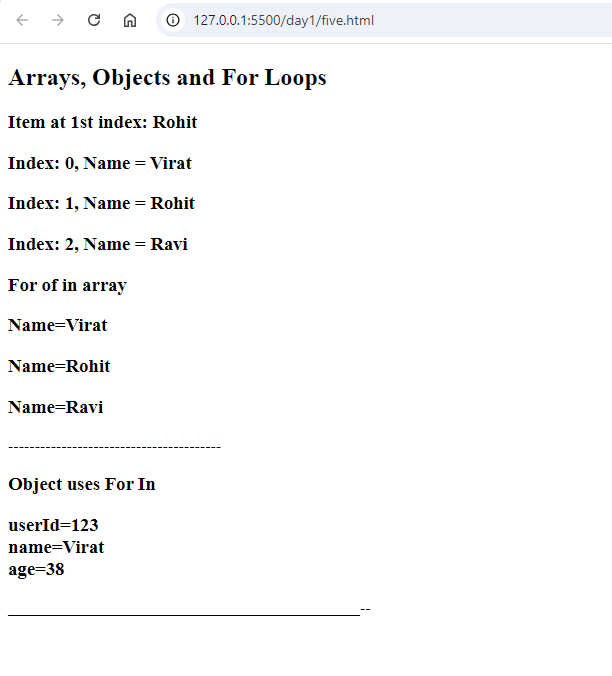
1. traditional for loop which uses index to access the array  
   i.e., for(init; condition; expr) { … }
2. for .. of : to iterate an array, but it doesn’t need index to access the element, it directly iterates the value  
   i.e., for(var x of names) { print x }
3. for.. in: to iterate an object with properties, it doesn’t need you to know what properties an object have

i.e., for(var u in user) { print user[u] }





Output:



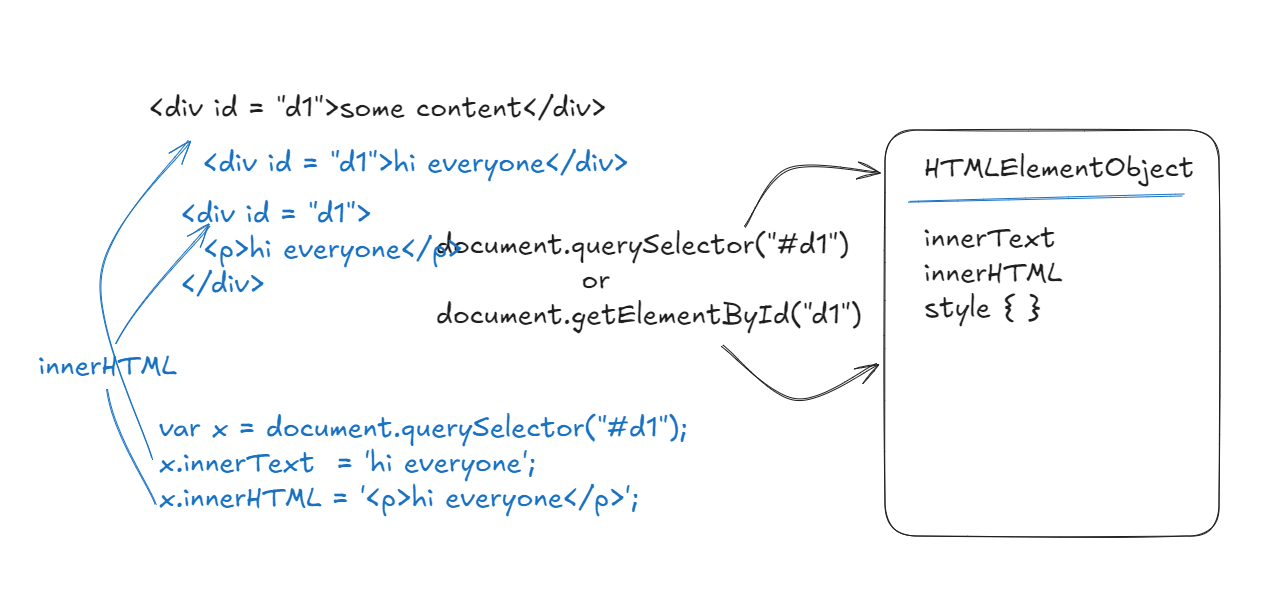
DOM manipulation:

Using Javascript you can access HTML and manipulate it like adding/removing contents/attributes, you can use some of the inbuilt functions provided by document object which are:-

* getElementById(“id”)
* getElementsByTagName(“tag”)
* getElementsByClassName(“class”)
* querySelector(“selector”): It can take tag, class and id
* querySelectorAll(“selector”): it can take tag, class & id

<div class = “c1”>hello</div>   
<div class = “c2”>thankyou</div>  
<p id = “p1” class = “c2”>hello p1</p>   
<p id = “p2” class = “c1”>hello p2</p>

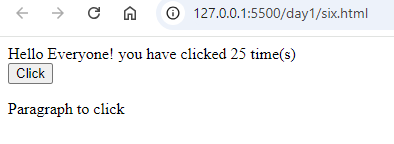
document.getElementById(“p1”) ; // <p id = “p1” class = “c2”>hello p1</p>  
document.getElementsByTagName(“p”); // [ <p>, <p> ]  
document.getElementsByClassName(“c1”); // [ <div.c1>, <p.c1> ]  
document.querySelector(“.c1”); // first element using the class c1 i.e., <div class=”c1”>  
document.querySelector(“#p1”); // <p id = “p1”>  
document.querySelector(“p”); // first occurrence of <p> i..e, <p id = “p1” class=”c2”>hello p1</p>  
document.querySelectorAll(“.c1”) or document.querySelectorAll(“div”)



x.style.color = ‘red’; >> <div id = “d1” style=”color:red”>



Output:





Above code at 28th line it is not calling handleClick, it uses handleClick as an handler that is executed when you place mouse over the element #btn2

event object:

This is an object that is supplied as a parameter to the handlers

btn2.addEventListener(“click”, function(ev) {   
 // ev is an event object that generated the click event  
});

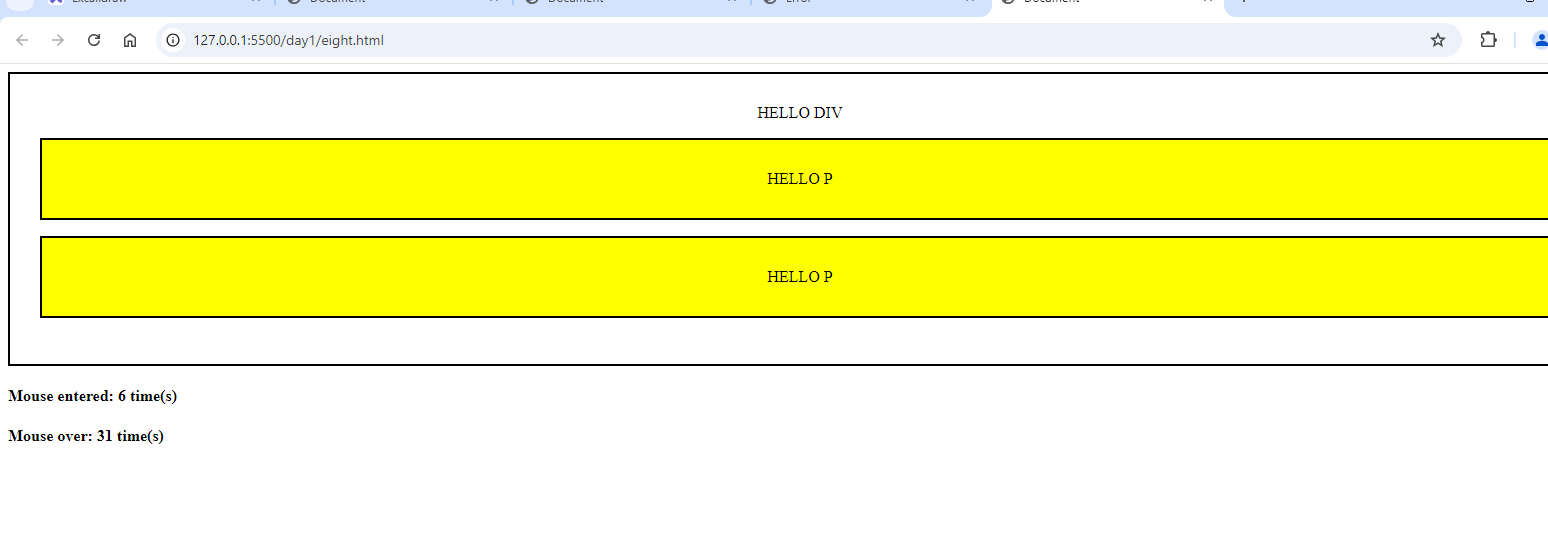
mouseover vs mouseenter: mouseover occurs even for the child elements, whereas mouseenter occurs only to the parent element using mouseenter

mouseout vs mouseleave: mouseout occurs event for the child elements, whereas mouseleave occurs only to the parent element using mouseleave

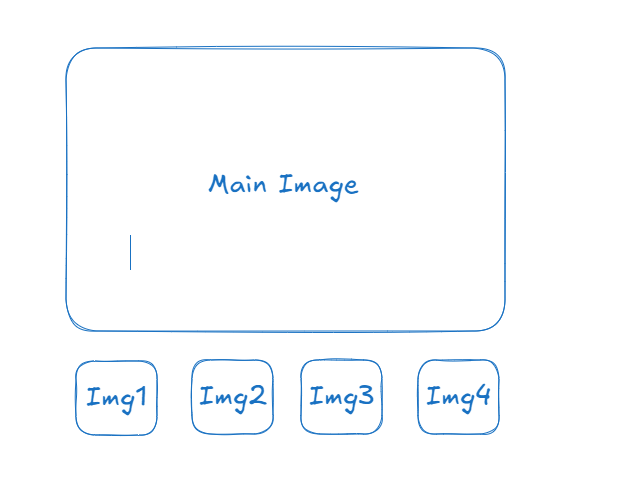
<div onMouseOver = “fn1()” onMousenter=”fn2()”>  
 <h2>..</h2>  
 <p>..</p>  
</div>



Output:



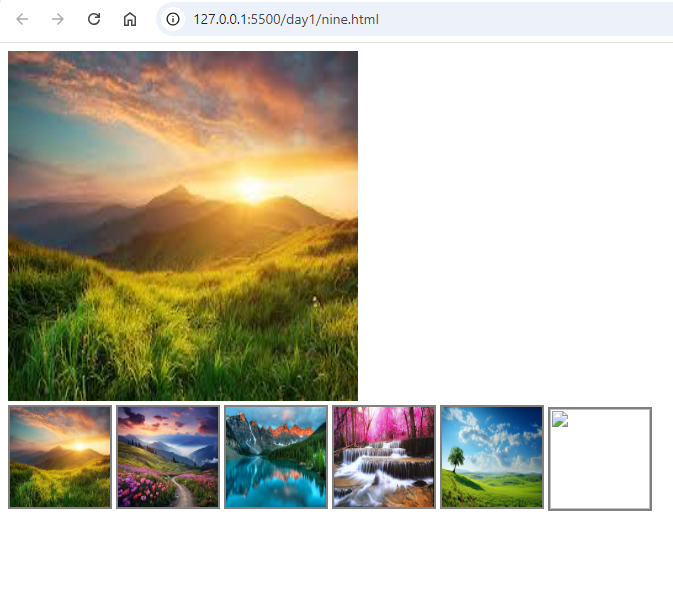
Using event handling to display the image that you click in the thumbnail, when you click on the thumbnail image, the main image will show the image you clicked on the thumbnail



Use <img> tag with src, width & height attributed i.e.,   
<img src = “ImageURL:GoogleImage” width = “300” height = “300” id = “main”>



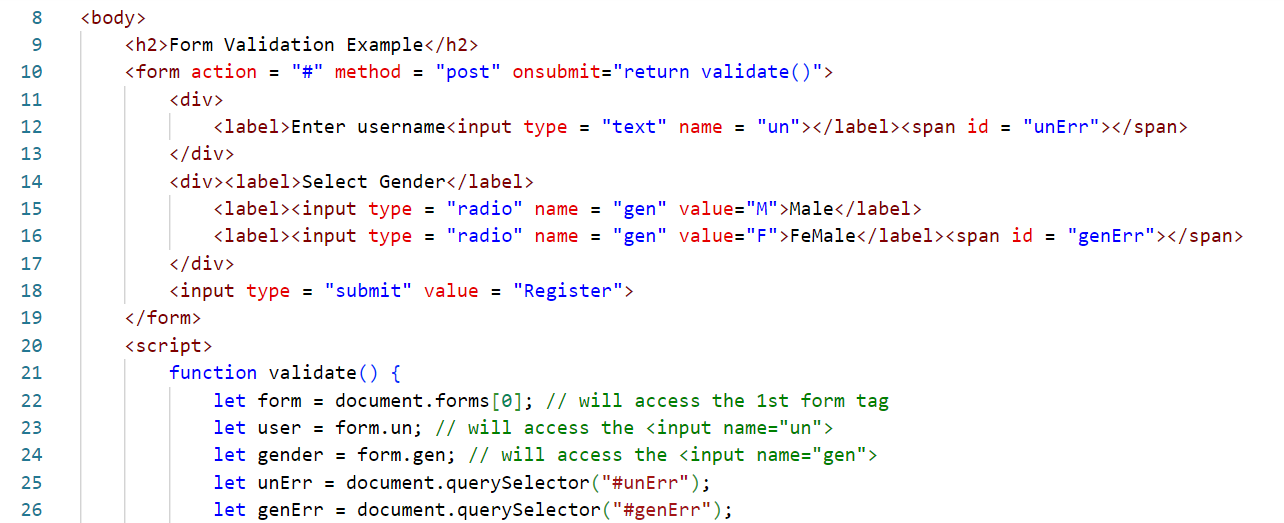
Output:

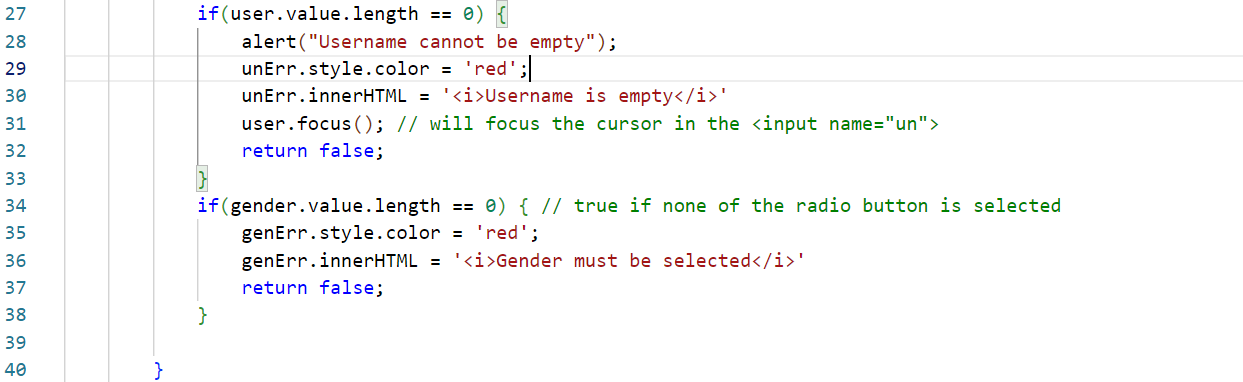


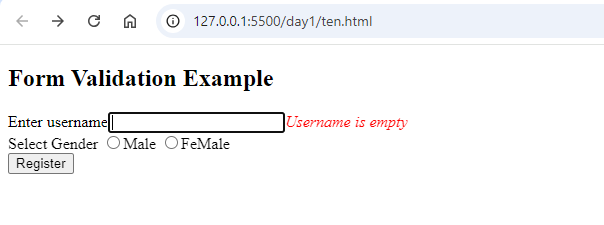
Form Validations in Javascript

These are client side validations that can restrict client to submit invalid inputs like

password & confirm password must be same, an account number / amount must not allow alphabet



Output:

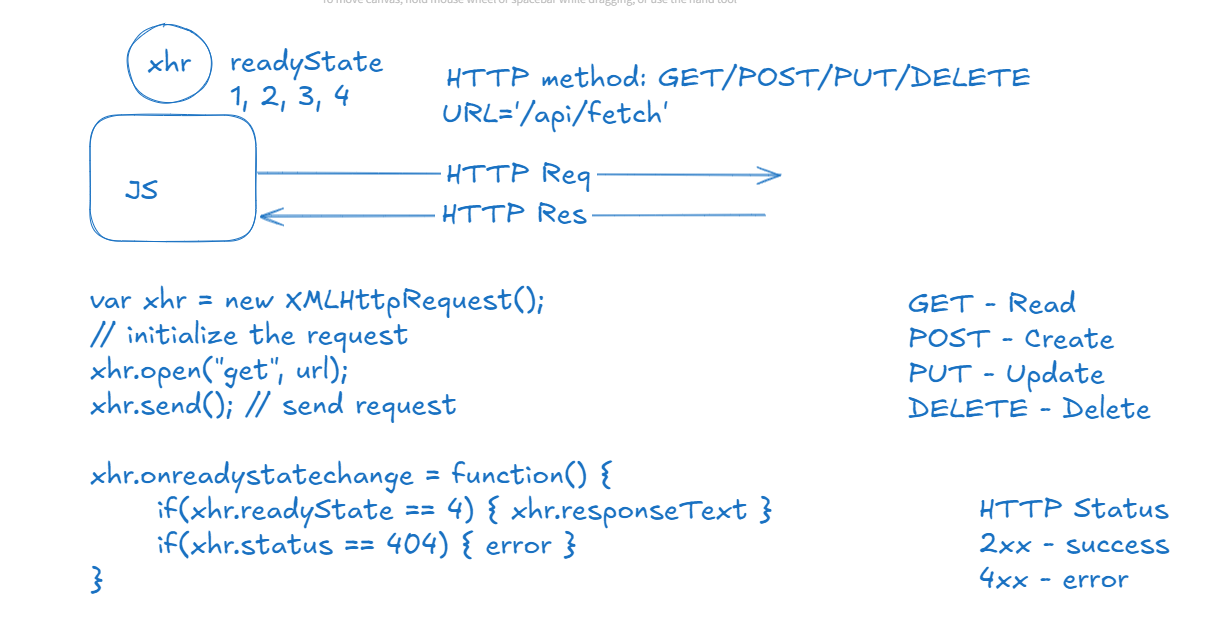


XMLHttpRequest

In Javascript it is like an HTTP client, It is used to make HTTP requests to the backend services, this was the very first approach javascript used to make HTTP requests.

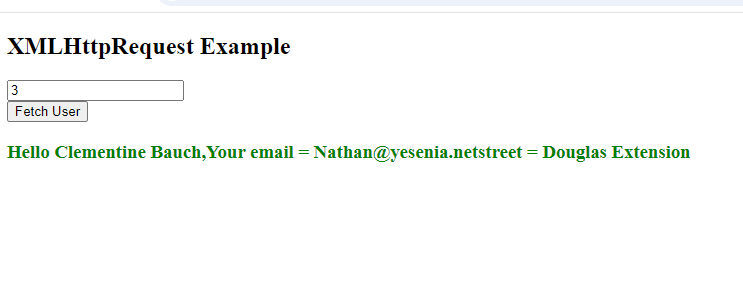
XMLHttpRequest object provides some properties & functions while making the request

|  |  |
| --- | --- |
| Properties | Functions |
| readyState | open(…) |
| onreadystatechange | send() |
| responseText |  |





Output:



ECMAScript new features i.e., ES6 or later version features

ECMAScript is a standard for Javascript, Browser, Node.js and many Javascript environment, they specify the standards about the new features and those standards must be followed by Javascript, Browser, Node.js and many other JS environment

ECMAScript called as ES released lot of changes in the syntax to simplify the syntax, they are

* New keywords like let, const, class, super, extends
* Template Strings
* Arrow functions
* Rest & Spread operators
* Array & Object Destructuring
* Promises
* Async / Await

Template String: It helps you to write the strings without breaking the string with + operator, whenever you want to add the javascript expressions in the string

ex: “<p>Hello “+name+”, your age “+age+”</p>”;

With template string

`<p>Hello ${name}, your age ${age}</p>`

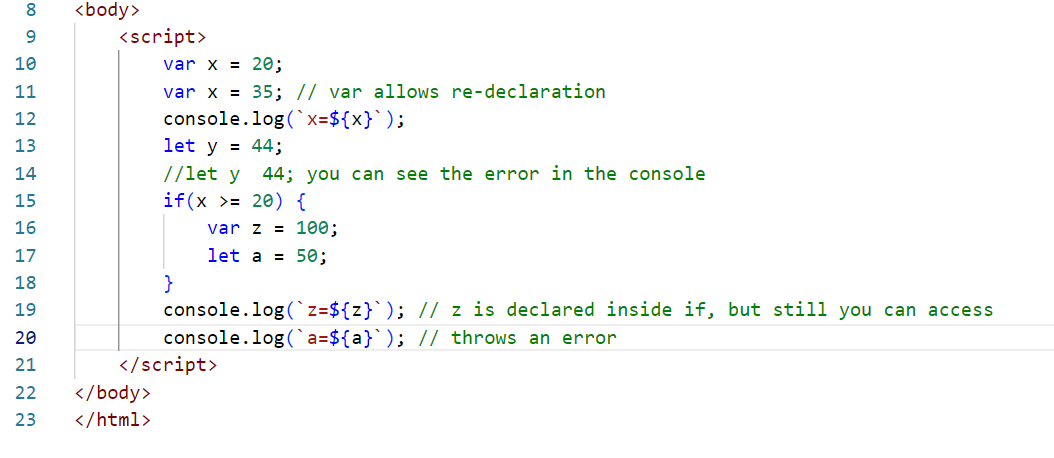


let & const keywords

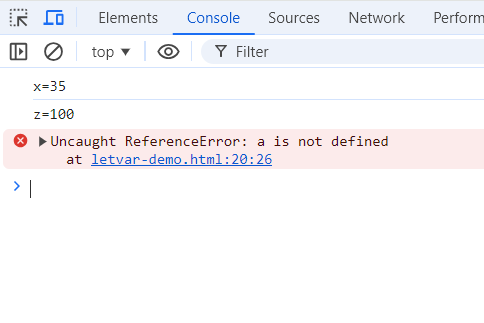
These two are block scoped variables, we must avoid using the var keyword because they are not block scoped

i.e.,   
if(…) {   
 var x = 20; let y = 30; const z = 50;  
}  
print x; // it prints 20  
print y; // it doesn’t work

let variables can be modified, however const variables can’t be modified



Output:



New methods are added to the Javascript array

1. forEach: It is for iterating the array
2. map: It is to transform each element into another form, it creates a new array
3. sort: it is used to sort, but takes comparable as a parameter

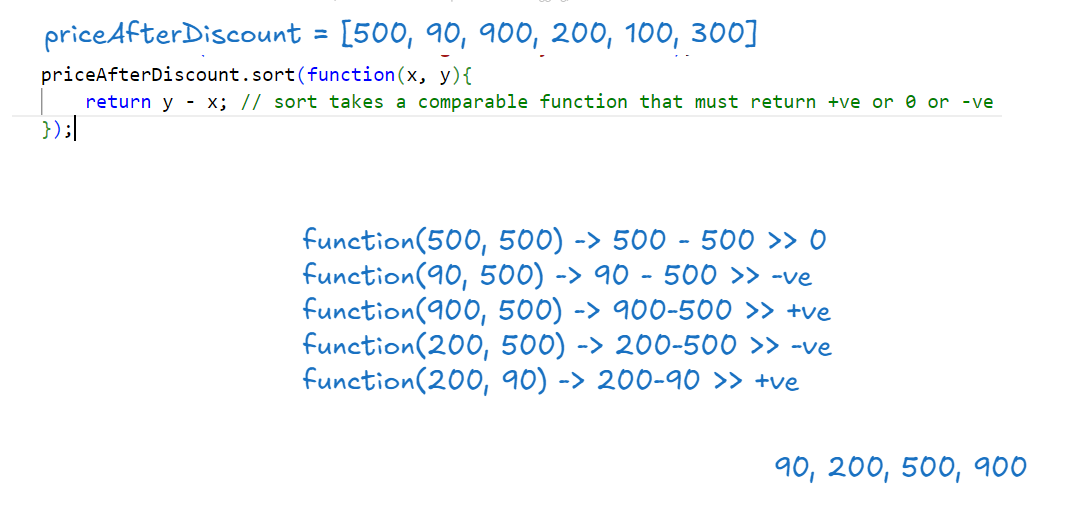
These do an internal iteration and takes a callback as an argument which is invoked on each iteration by passing iterated value & their index

let items = [2, 3, 1, 4, 5];

items.forEach( function(v, i) { } );  
let newItems = items.map(function(v, i){ return value; } );



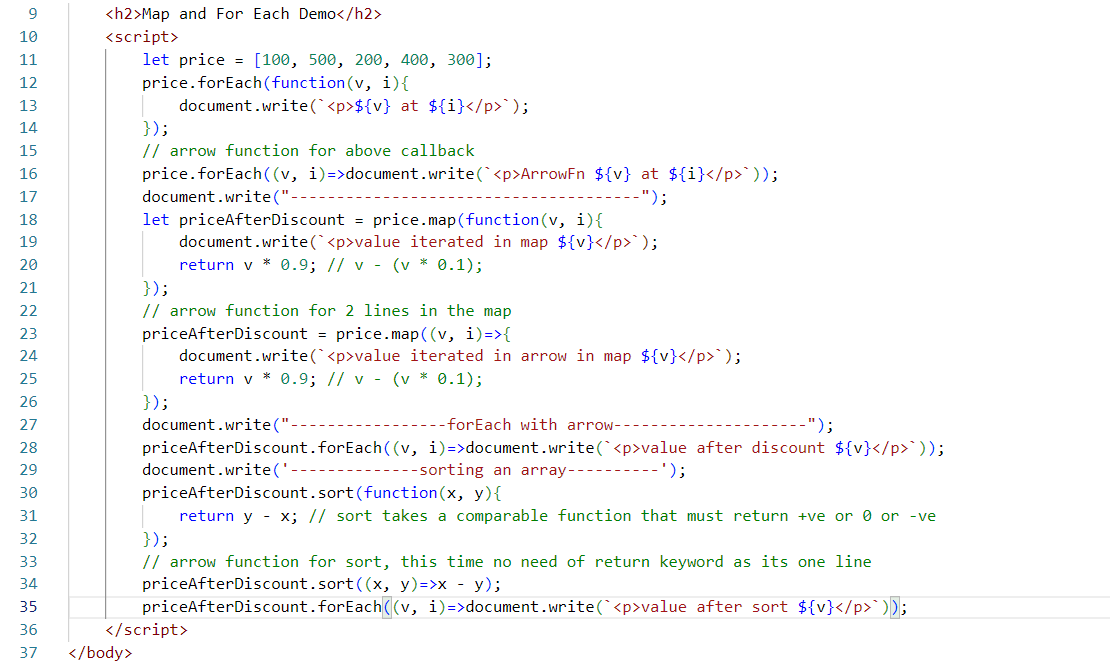
Output:





Arrow function: It is a new syntax that simplifies writing the callbacks, where you don’t have to use function keyword & return keyword (if callback has only one line statement)

|  |  |
| --- | --- |
| Callback | Arrow |
| function(a, b) { console.log(a + b) } | (a, b) => console.log(a + b);  or  (a, b) => { console.log(a + b) } |
| function(a, b) { return a + b; } | (a, b) => a + b;  or  (a, b) => { return a + b } |
| function(a) { return a \* 10; } | a => a \* 10; // no need of (a)=> when its single argument |
| map(function(x, y) {  return x \* 0.9 }); | map((x, y) => x \* 0.9); |
| sort(function(x, y) {  return x – y; }); | sort((x, y) => x – y ); |



Valid and Invalid arrow functions

(x, y) => x + y; // valid

(x, y) => return (x + y); // invalid

(x, y) => { x + y } ; // invalid, now return is must, as the statement is an expression

(x, y) => { console.log(x + y) } // valid, it has a statement

(x, y) => { return (x + y) } // valid

let items2 = [1, 2, 3, 4, 5, 6, 7, 8];

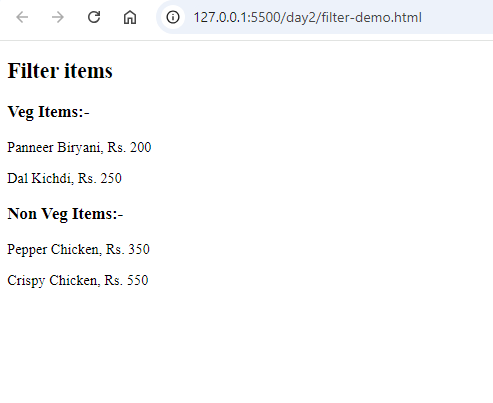
let filteredItems = items2.filter(function(x) { return (x % 2 == 0) });  
Use arrow function for the same

console.log(filteredItems);

filter(): it takes a callback that iterates all the elements and those elements that are true will be stored in a new array



Output:



class, constructor, extends, super keywords

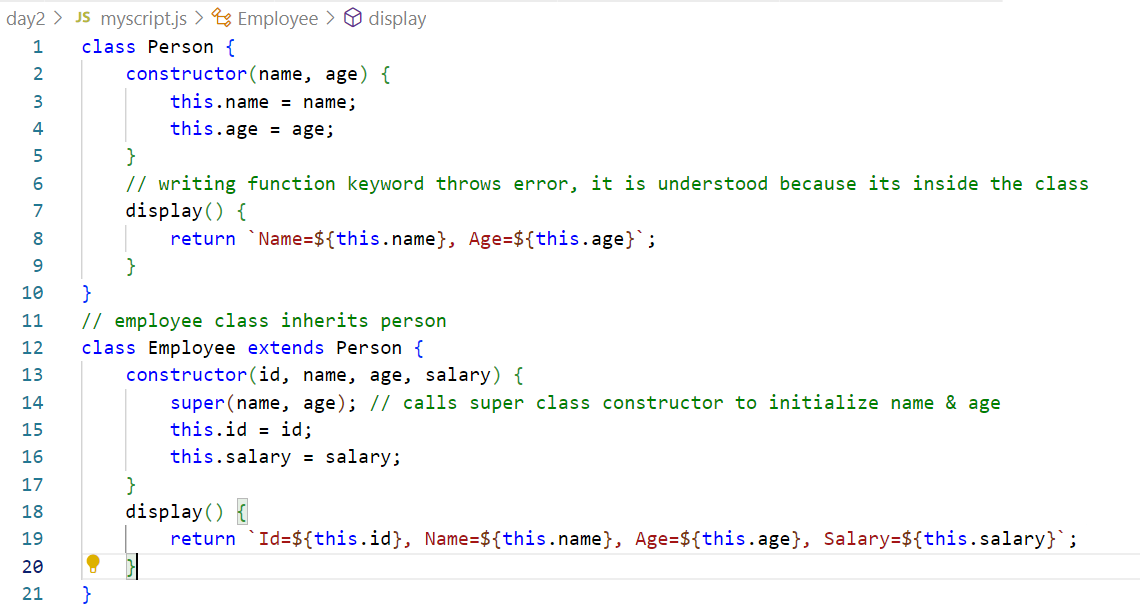
Earlier you need to create function to create object

function Person(name, age) {   
 this.name = name;  
 this.age = age;  
}  
Person.prototype.display = function() { print name & age }  
let p = new Person(“Raj”, 35);

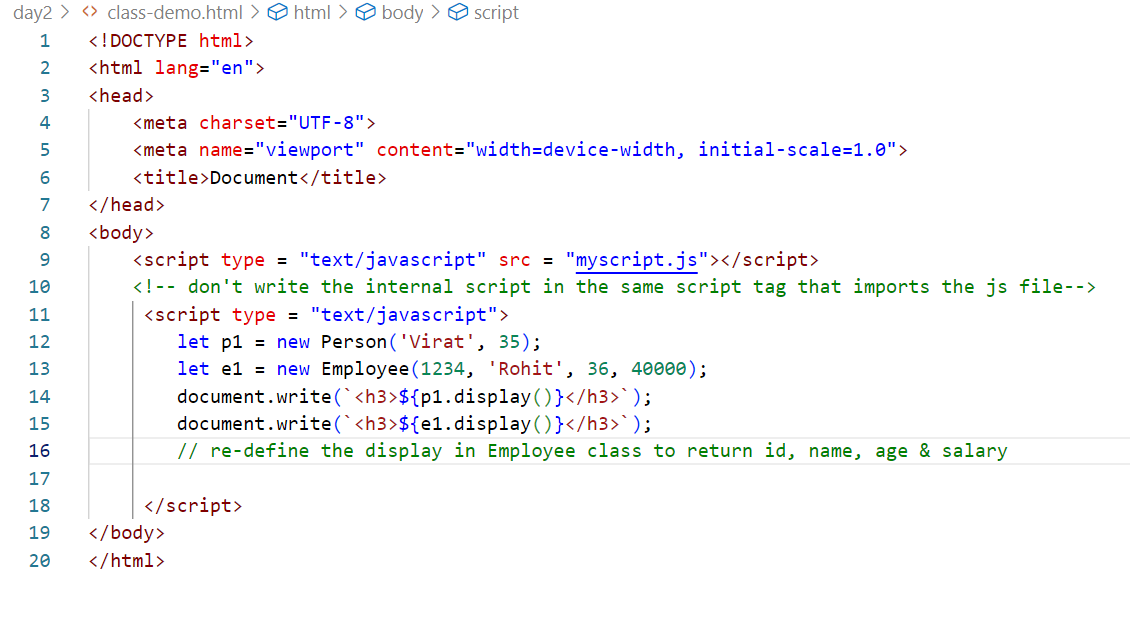
With the help of class, constructor you can modularize the function & properties

class Person {   
 constructor(name, age) { this.name = name; … }  
 display() { … } // don’t write function keyword  
}  
class Employee extends Person {   
 constructor(id, name, age, salary) {   
 super(name, age); // calls super class constructor  
 this.id = id; this.salary = salary;  
 }  
}

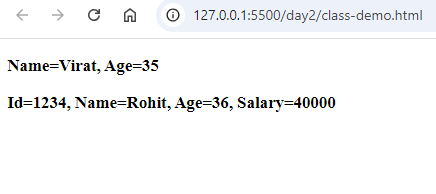
myscript.js



class-demo.html



Output:



Note: Javascript may support overriding, but it doesn’t support overloading, like same method having different arguments

Rest Parameter

It is used to allow a function to accept 0 or more parameters, in a function you can maximum have 1 rest parameter, but it must be the last parameter

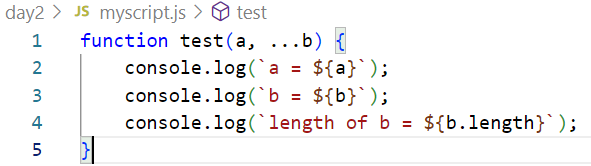
function test(a, …b) { … }  
test(25); // a = 25, b = []  
test(25, 30); // a = 25, b = [30]  
test(25, 30, “hello”); // a= 25, b = [30, “hello”];

function test2(a, b) { … }  
test(25); // a = 25, b = undefined  
test(25, 30); // a = 25, b = 30;  
test(25, 30, 40); // a = 25, b = 30

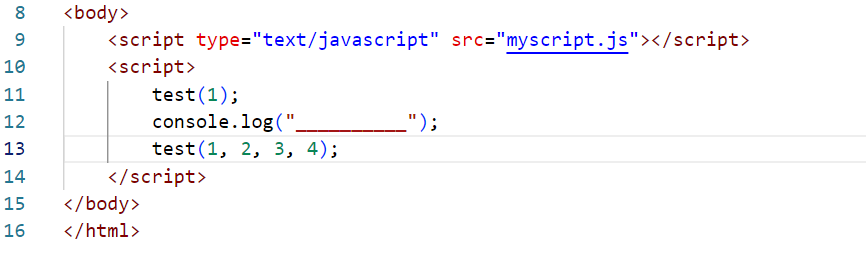
Note: Rest parameter is an array itself, you can call any array functions from the rest parameter

i.e., b.forEach(), b.sort(), b.filter() and so on

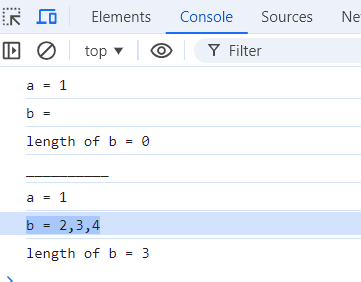
myscript.js



rest-spread-demo.html



Output:



Spread operator

It is used to assign a variable(array) to multiple parameters

function test2(x, y, z) { }  
let items = [10, 20, 30];  
test2(items); // x = [10, 20, 30], y = undefined, z = undefined  
test2(items[0], items[1], items[2]); // x = 10, y = 20, z = 30;  
test2(…items); // x = 10, y = 20, z = 30 – spread operator

Having rest & spread both

function test3(a, b, …c) { }  
let items = [20, 10, 30, 40, 50];  
test3(…items); // a = 20, b = 10, c = [30, 40, 50]

Activity:

Create a single function that can return sum, max and min values based on the parameters you pass, print that returned value in the browser document.  
ex: calci(‘sum’, 20, 30, 40); // return sum=90  
ex: calci(‘max’, 20, -1, 50, 30); // return max=50  
ex: calci(‘min’, 20, -1, 50, 30); // return min=-1

Destructuring

It is used to easily access array items or object properties in a less code

let items = [2, 1, 3];

let x = items[0];  
let y = items[1];  
let z = items[2];

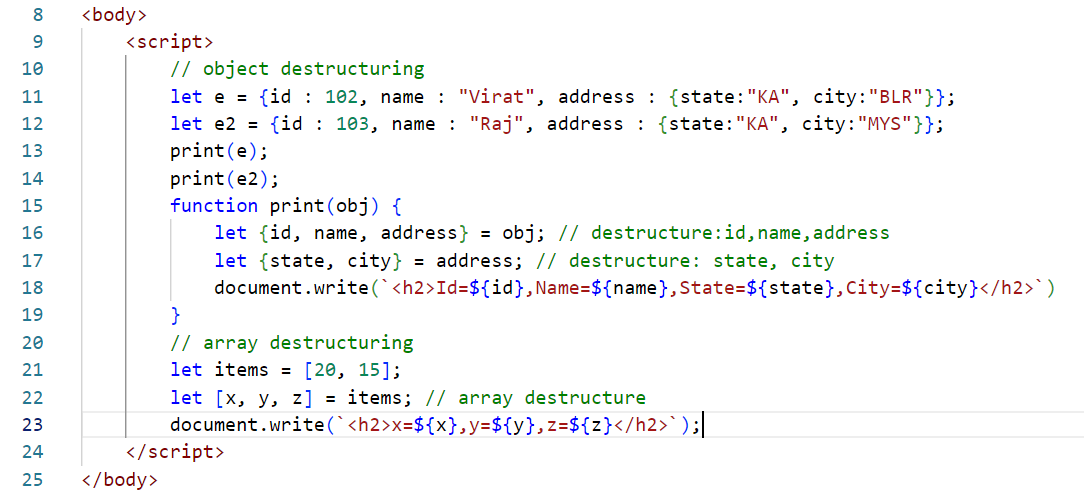
Destructuring array

let [x, y, z] = items;

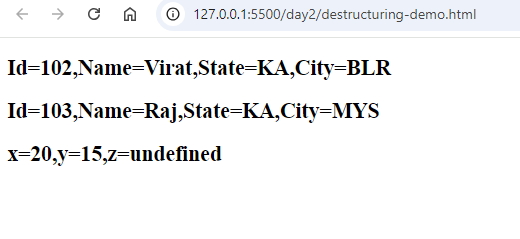
Destructuring object

let employeeObject = {id : 101, name : “Virat”, salary : 35000, address : {….} };  
let { id, name, salary, address } = employeeObject;

function test(emp) {   
 let {id, name, salary, address} = emp;  
 update(id, name);  
}  
test(employeeObject);



Output:



Optional Chain

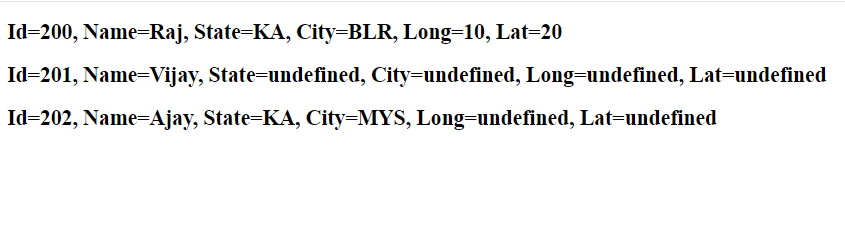
It is used to break the access to the nested property when the property is missing, so that it can avoid error at runtime

ex:   
users = [  
 {id: 1, name : “Raj1”, address : {state:”MH”, city:”MBI”} },  
 {id: 2, name : “Raj2”},  
 {id: 3, name : “Raj3”, address : {state:”KA”, city:”BLR”} }  
]

users.forEach(… value.id & value?.address.state..)

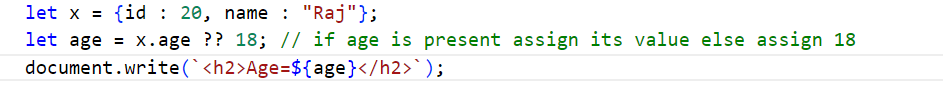


Output:



Checking Null/Undefined values(??)  
Assume user has list of contact details in an array  
user = { id : 100, name : “Raj”, contact = [ {userId, phone }, { userId, phone} ] }

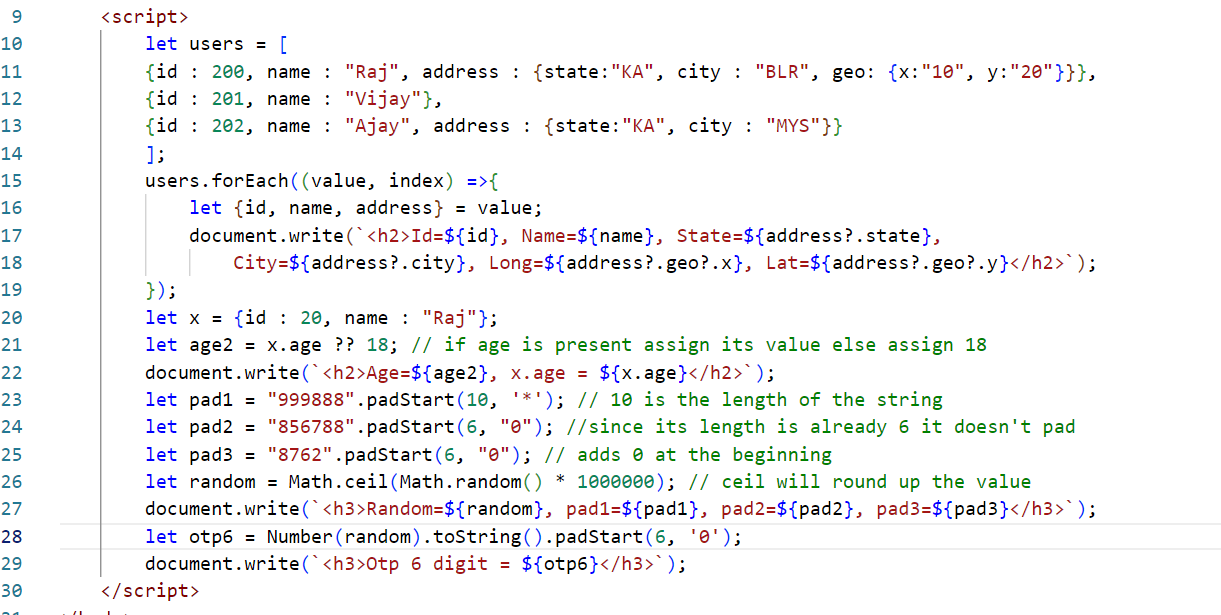
function test(user) {   
 let age = emp.age ?? 18; // if age is undefined or null assign 18  
 let contact = emp.contact ?? [];  
 contact.map(…); // will not throw error because contact is atleast []  
}



String padding:

Javascript provides some new functionalities to the string like padStart() & padEnd() which can add some masked characters either at the start or end of the string.

ex: 998877\*\*\*\* or \*\*\*\*\*\*\*9393 or abc\*\*\*@gmail.com  
ex: 089323



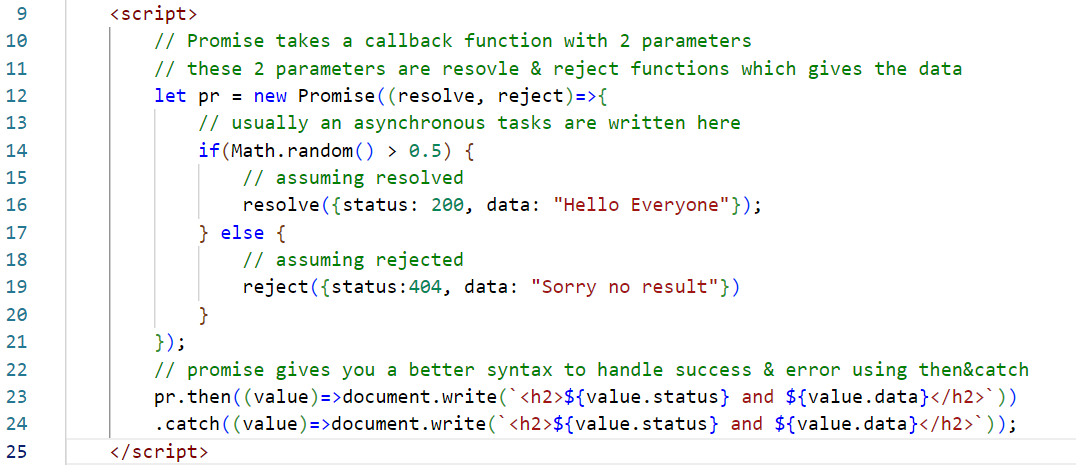
Promises:

These are objects the can do an asynchronous operations which can eventually succeed or fail i.e., resolved or rejected, these asynchronous operations can be accessing backend api’s, db connections, file reading/writing and so on.

We can create a Promise object using a Promise constructor

let p = new Promise( callbackFn(resolve, reject) {  
 // resolve & reject are the functions that can give value  
 resolve(data);  
 reject(data);  
});

Promise gives 2 functions to handle the asynchronous response   
.then(callbackFn) & .catch(callbackFn)



Output:

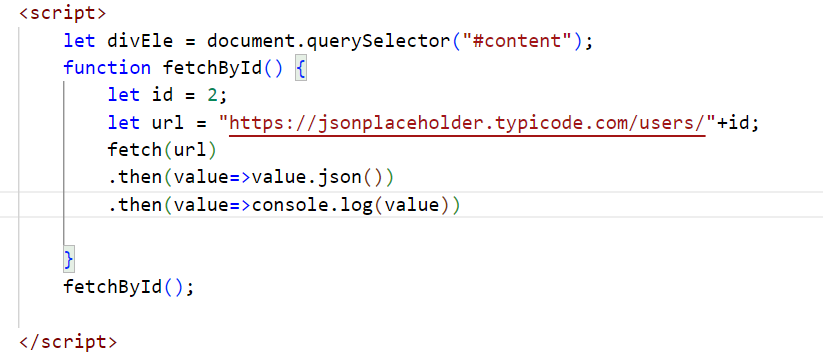


To consume the promise you must use .then(callbackFn) & .catch(callbackFn)

promise.then( callbackFn(value) ).catch( callbackFn(value) )

fetch(): It is a javascript function used to send asynchronous HTTP requests to the backend API, it’s an enhanced form of XMLHttpRequest

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



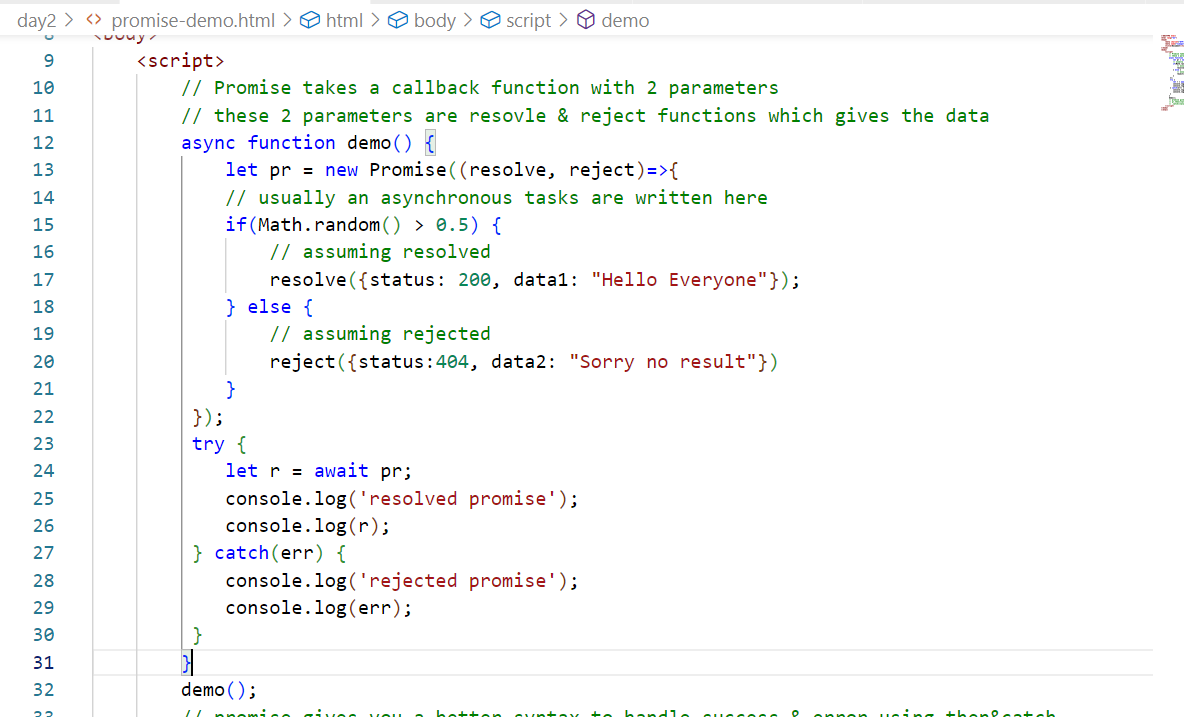
Async / Await

This is enhanced form of Promise, where you can wait for the promise to either resolve or reject and then process those data.

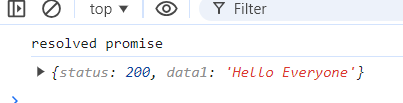
async function fetchById() {  
 let r = await fetch(url);  
 let r2 = await r.json();  
}



async/await with promise



Output:



rejected promise

