Day 1

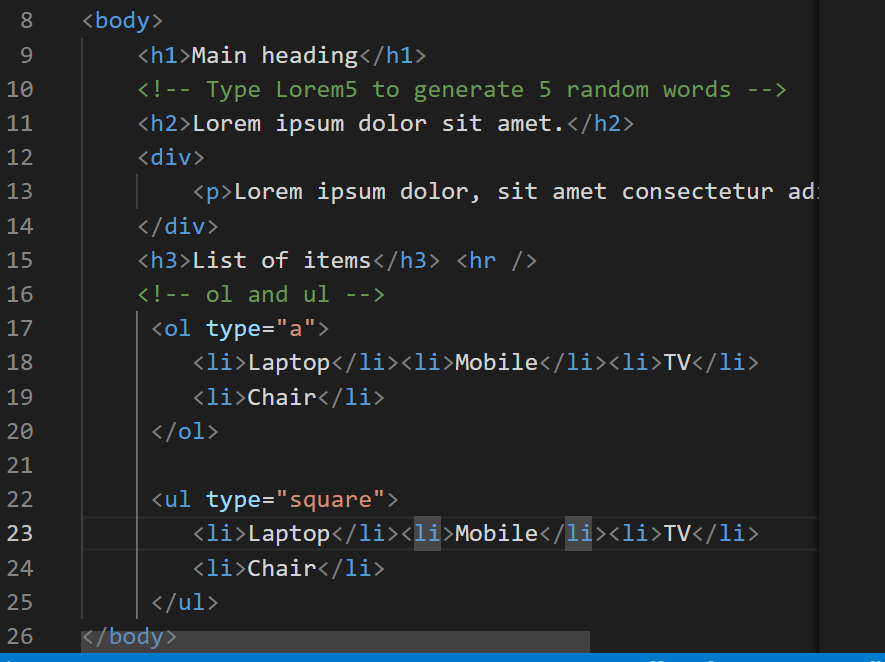
HTML, CSS & SASS/SCSS

HTML: Hyper Text Markup Language - Display the content

CSS - Cascading Style Sheet - Style

HTML file

Lists & headings



Output:



You can create HTML tables, forms and other elements

CSS:

It is used mainly to add the styles to HTML elements

RWD

It helps to fit your web page to all types of devices it could be desktop, mobiles, printing devices, tablets

Same HTML content to render differently in different devices

Medi Query: it applies the styles based on the conditions.

Grid: It arranges the elements in row & column format

Flex: It automatically arranges the elements based on the size like horizontally or vertically.

{display: flex }

SASS

Syntactically Awesome Style sheets

It avoids lot of boiler plate code which you write in CSS

SASS -> SCSS (this resembles css syntax), but sass follows indentation syntax

SASS syntax  
$font-style: arial;  
$color: blue

h1   
 color: $color;  
 font-family: $font-style

SCSS syntax:

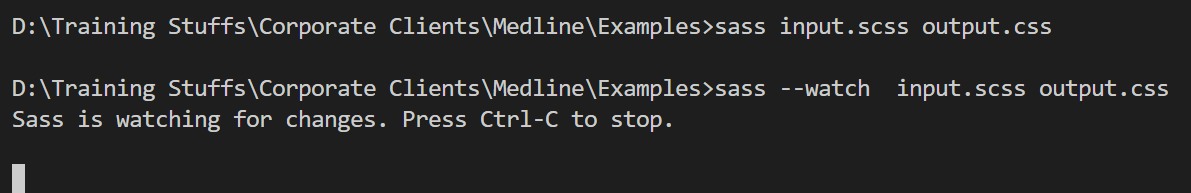
$color: blue;  
$font-style: arial;  
h1 {   
 color: $color;  
 font-family: $font-style;  
}

Install sass

npm install -g sass

To compile scss to css

sass input.scss output.css



--watch lets you to automatically compile scss to css

Mixins

These allow you to reuse the group of CSS declarations.

@mixin shape($color, $width, $height) {   
 width: $width;  
 height: $height;  
 background-color: $color;  
}   
.square {  
 @include shape(blue, 400px, 400px);   
}  
.rectangle {  
 @include shape(red, 400px, 200px);  
}

Parent selector

It is used in nested selector to refer the outer selector

.alert {  
 &:hover { color: red }  
}

In CSS you write

alert : hover { color: red; }

Placeholder

It is to reuse some selectors using inheritance

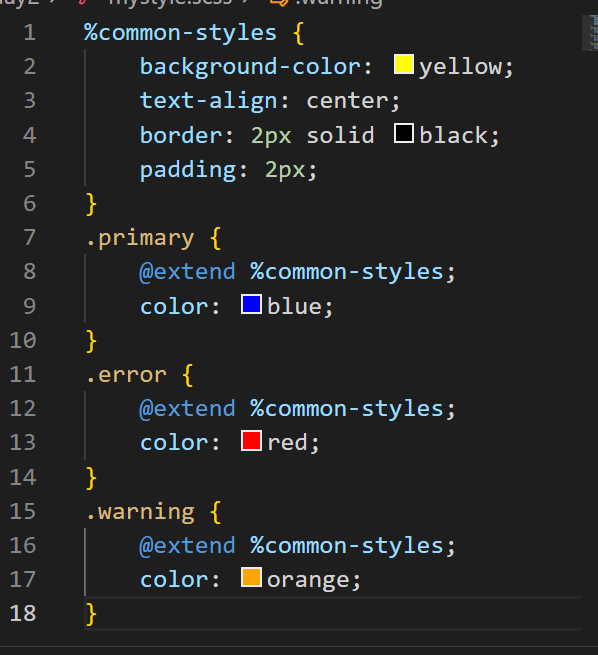
%name {  
 // css selectors  
}

.primary {  
 @extend %name;  
 color: blue;  
}  
.secondary {  
 @extend %name;  
 color: grey;  
}

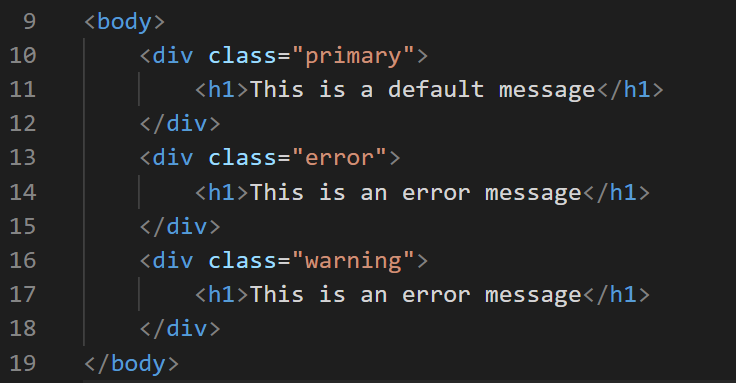
How to compile multiple SASS files

sass --watch file1.scss : file1.css file2.scss : file2.css

day2/mystyle.scss



index.html



Output:



Special functions:

SCSS provides various functions to perform complex operations

$colors: red, green, blue;  
.error { color : nth($colors, 1) }

.warning { color: nth($colors, 2) }

Mixing colors using mix()

.mixed-color { color: mix(red, green, 60%) }

Here 60% red & 40% green

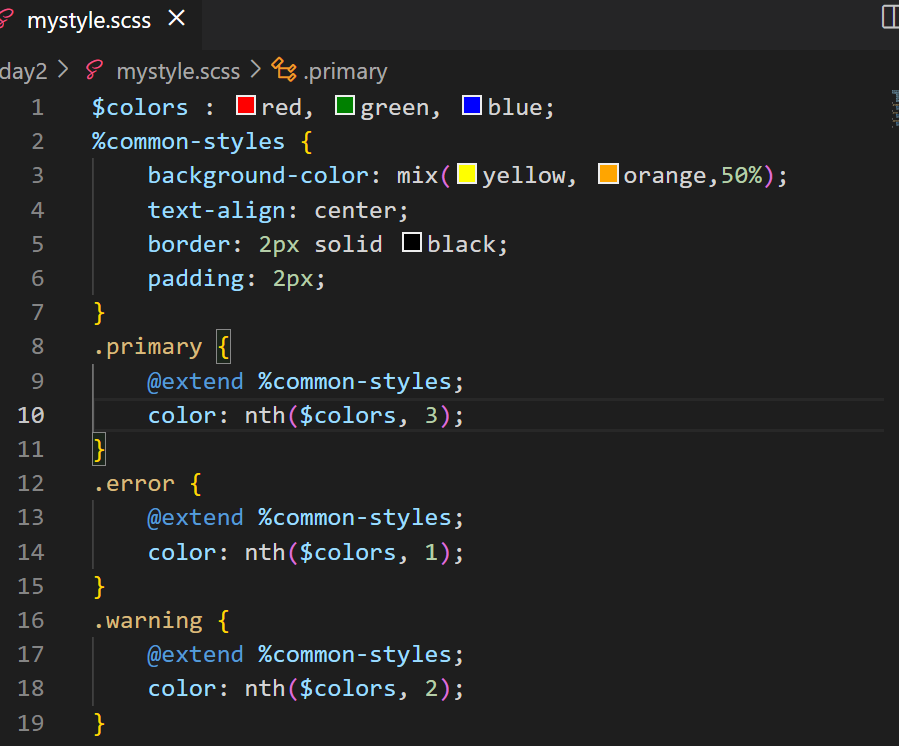
Note: You can only mix two colors, however you can get the reference of two colors and pass the color again to the mix

mix(mix(red, green, 60%), blue, 50%)

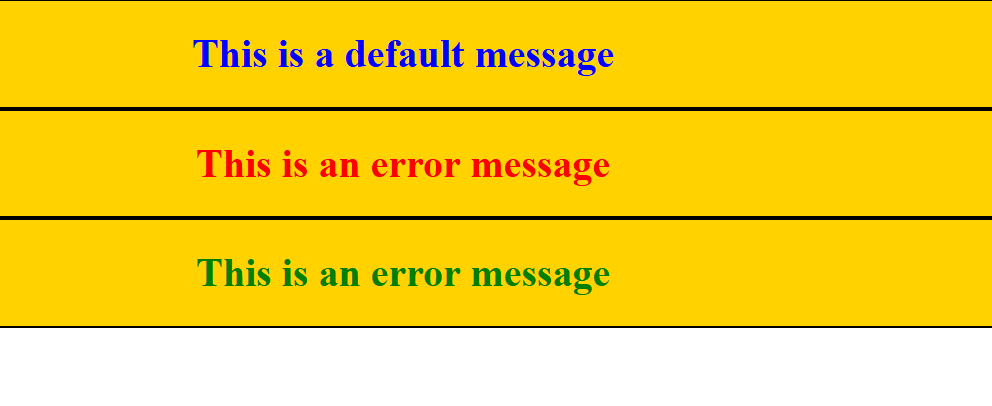
Round up the numbers

font-size: round(19.6px); // font-size: 20px

scss



Output:



SASS official document

@if & @else if : these are used to apply the CSS on a conditional based

!default: to provide a default values when the property doesn’t have a value

<https://sass-lang.com/guide/>

Javascript

HTML - displaying the content

CSS - styling the HTML

Javascript - It adds effects the web page by dynamically accessing HTML & CSS, it has various features like

* variables
* operators
* datatypes
* functions
* objects
* arrays
* conditional statement
* loops
* form validations

There are two ways you can execute Javascript

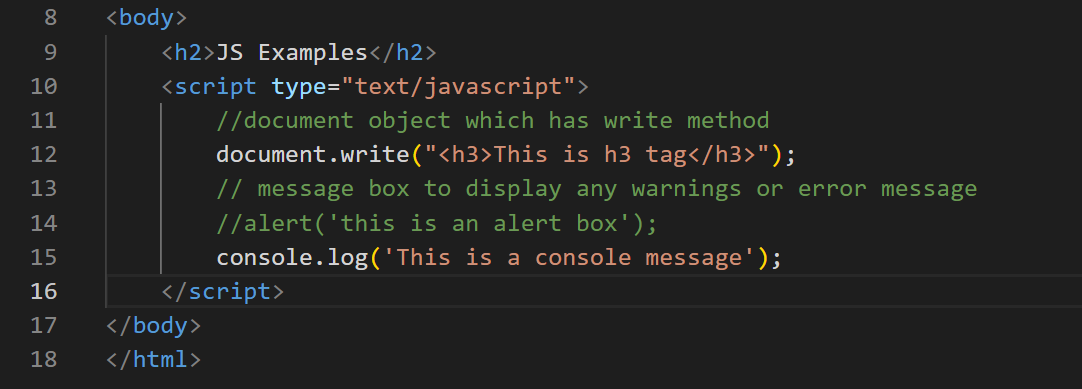
1. Using browser at the front-end
2. Using Node.js at the backend

Strings in Javascript: You can either use single or double quotes

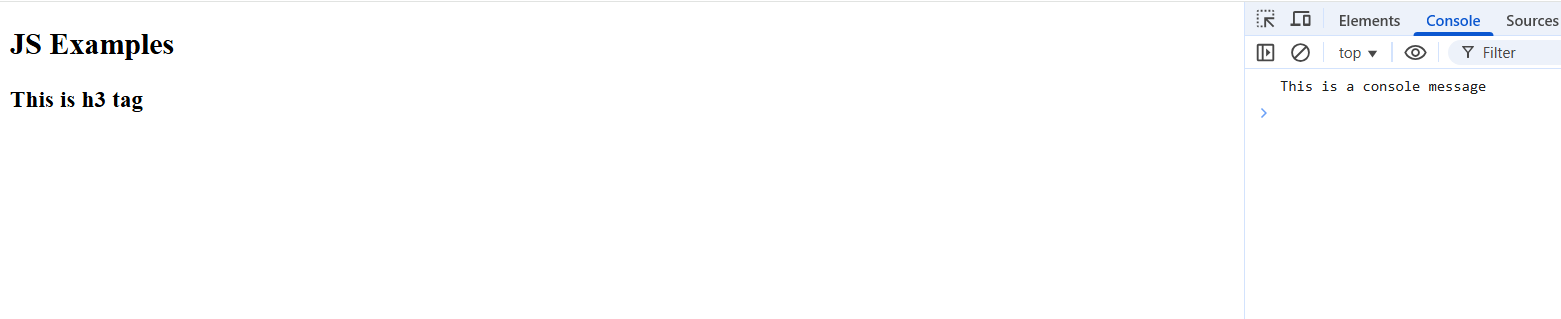
Note: single quote is not treated as character it is treated as string itself

Note: semicolon is optional in Javascript as long as you keep the statements in the new line

JS hello world program



Output:



Fundamentals of Javascript

1. Variables
2. Datatypes
3. Operators
4. Conditional Statements
5. Loops

Variables: It is to store the data in Javascript

old approach is to use var keyword

var employeeId = 1234;  
var employeeName = “Siddharth”;

I can assign different values to the same variable with different types

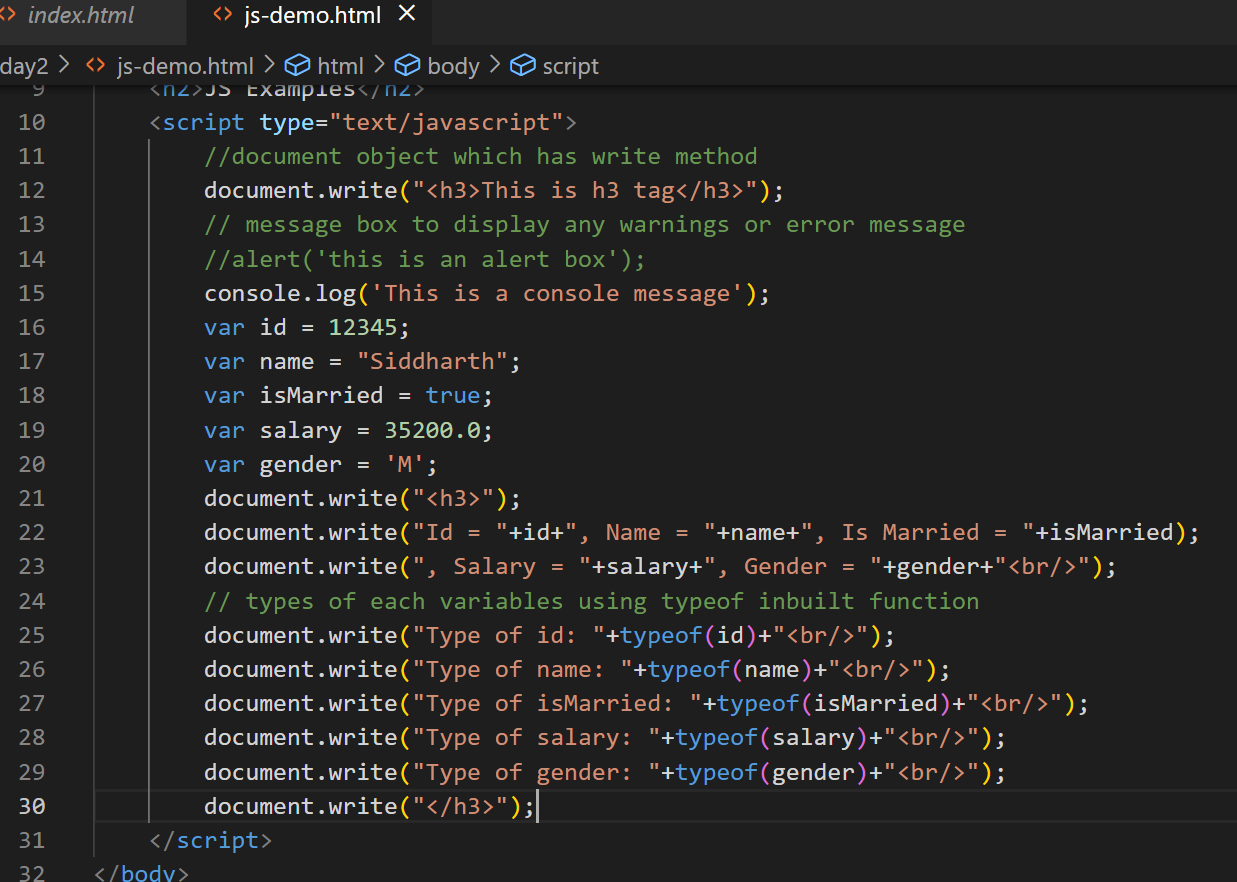
employeeId = “12345” or employeeId = true

Datatypes in Javascript

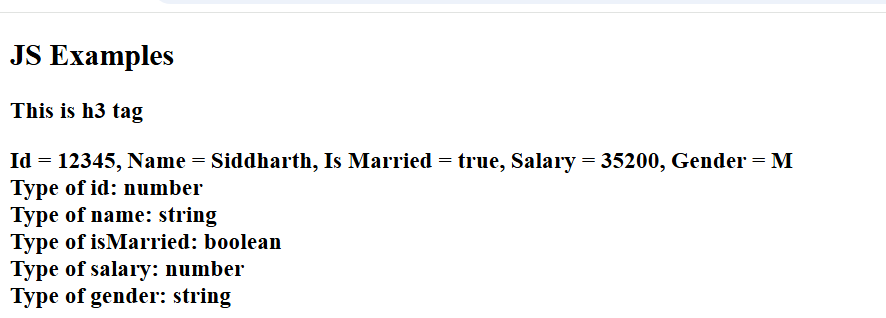
In Javascript variables get the types based on the type of value, you can use an inbuilt function in javascript typeof(variableName) to get the datatype

document.write(typeof(employeeId); // number or string or boolean

jsdemo.html



Output:



Javascript operators

Arithmetic operators: +, -, \*, /, %

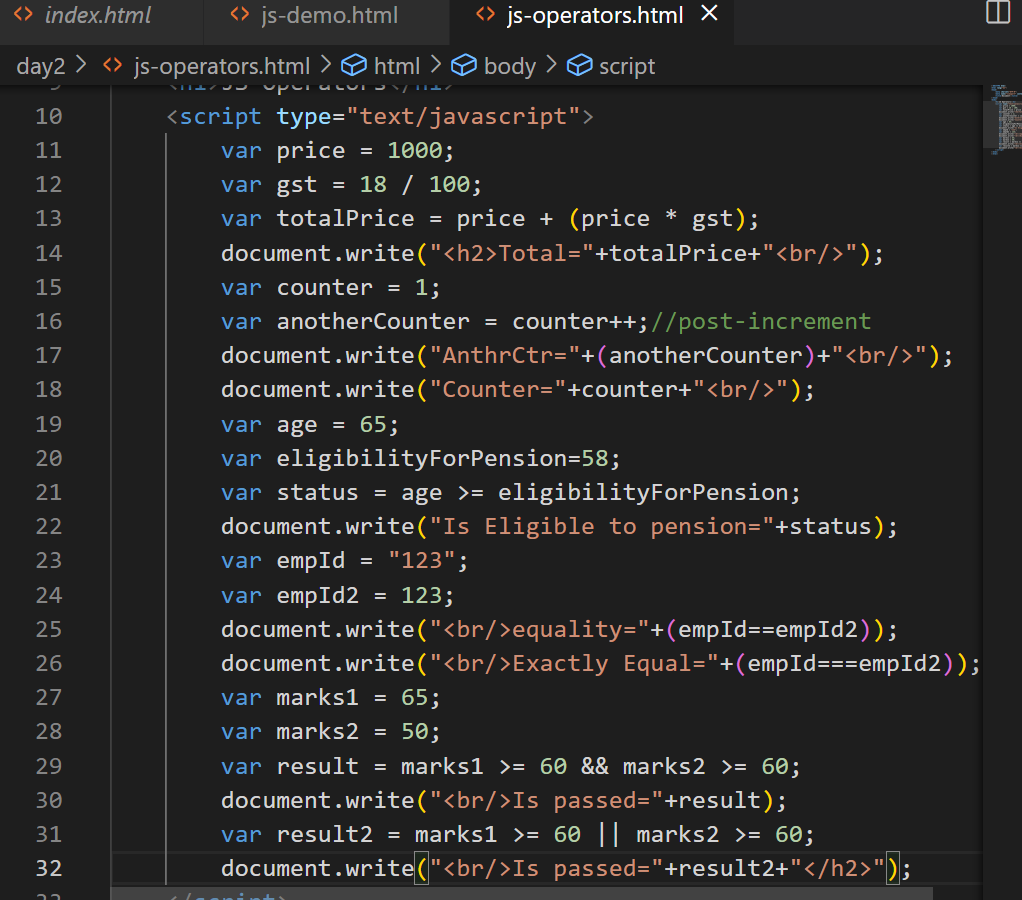
Assignment operator: =

Comparison operators: <, >, <=, >=, !=, ==, ===

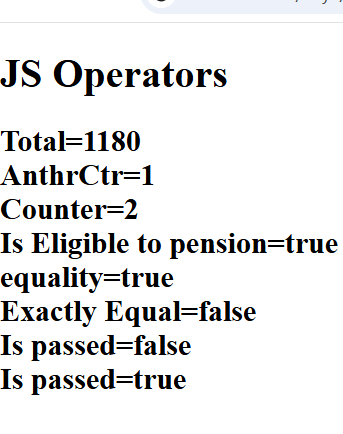
Increment & Decrement operators: ++, --

Logical operators: &&, ||

Operators Demo



Output:



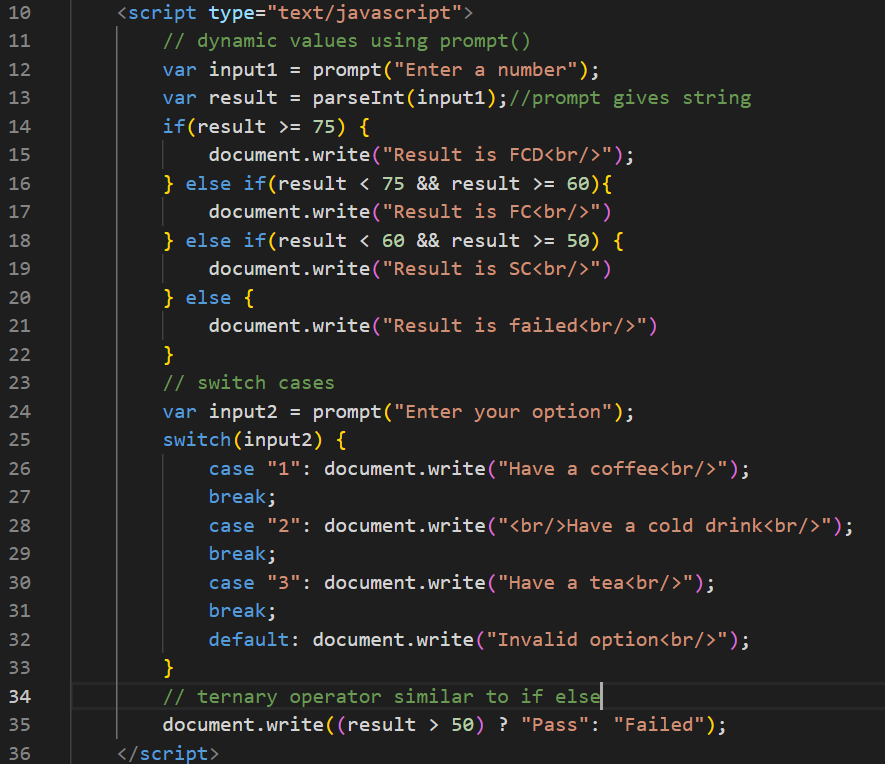
Conditional statements:

Statements that you want to execute only if its true or false

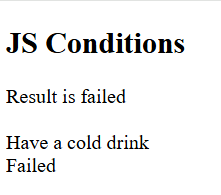
In Javascript we have following conditional statements

1. if
2. if else
3. if else if else if … else
4. switch

Conditional Statements demo



Output:



Day 3 agenda

**Loops**

* While Loop
* For Loop
* For...In Loop
* Loop Control

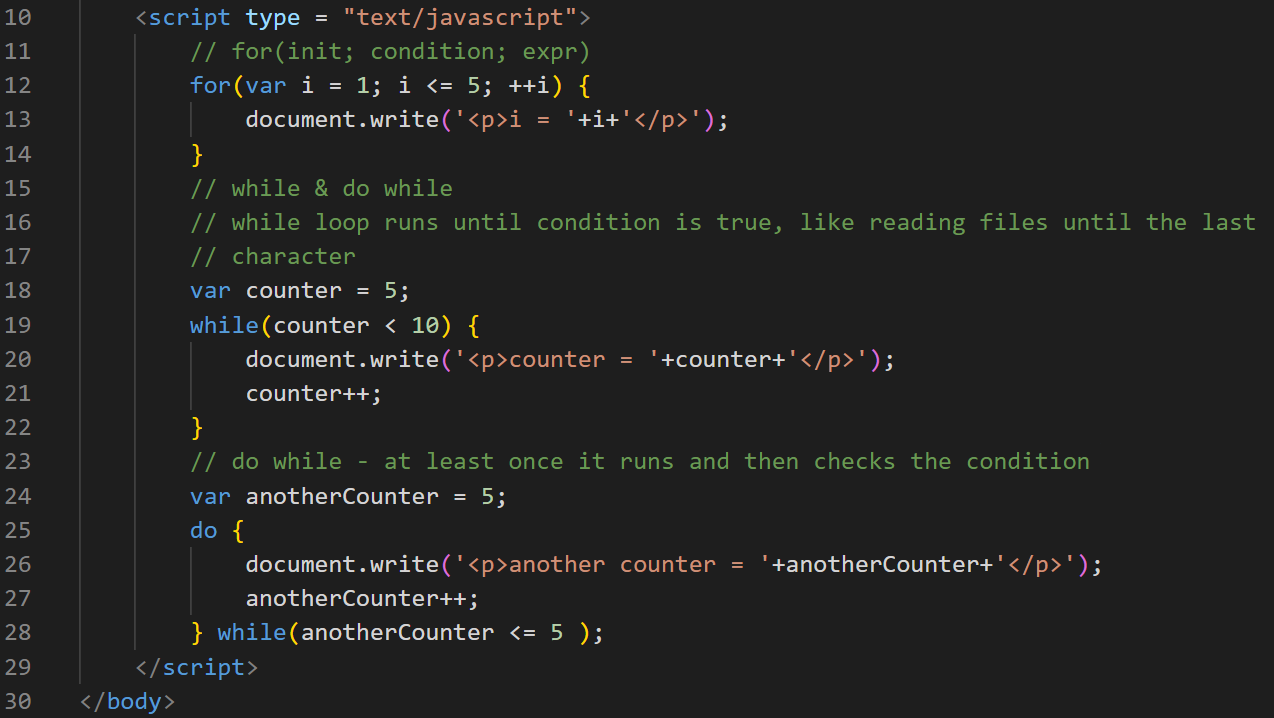
**JavaScript Functions & Objects**

* JavaScript Functions
* The Object Constructor
* Defining Methods for an Object
* JavaScript Boolean
* JavaScript String
* JavaScript Arrays
* JavaScript Arrays Method
* JavaScript Arrays Sort

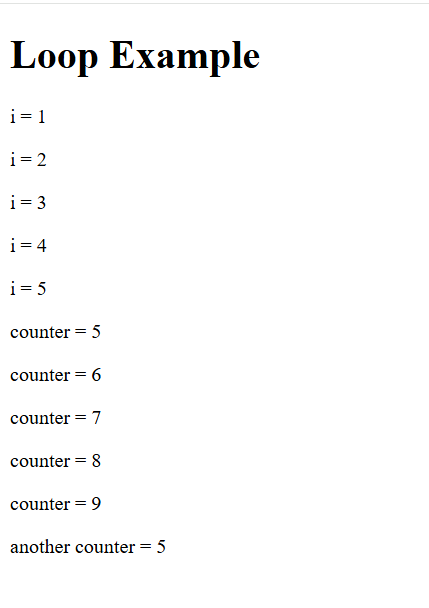
Loops: Same statements to execute until certain condition is true, there 3 types of loops

1. for loop
2. while
3. do-while

for loop: When you want to iterate fixed set of iterations then you can use for loops, it is used while reading array elements also.



Output:



Objects in Javascript

In Javascript you can create objects that will have properties & functions/methods, there are many ways you can create objects in Javascript

1. Literal style
2. functional style with object constructor
3. class style (modern syntax)

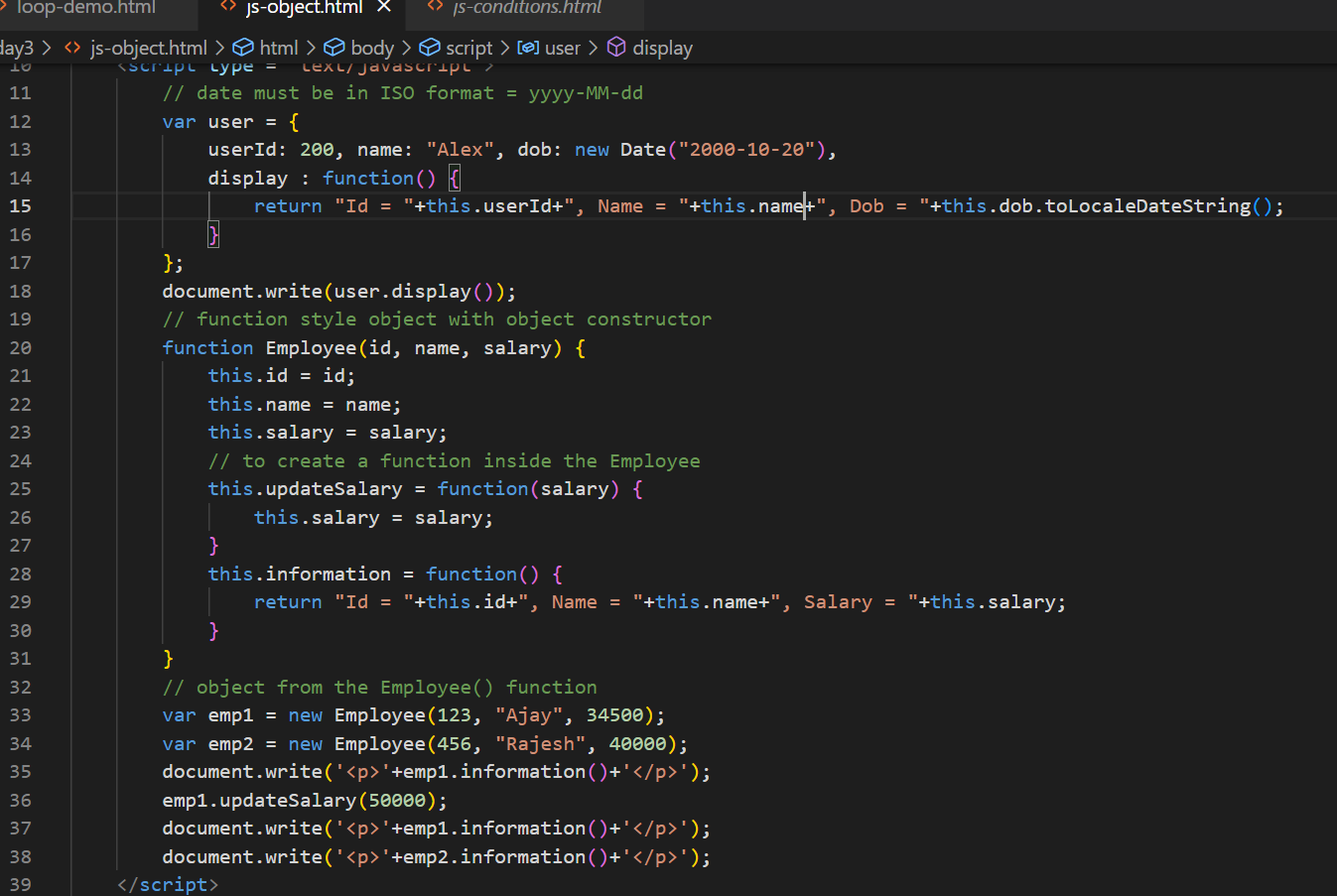
Literal style: This is better when you want one or two objects to be created or when you want some mock objects to be created

{ property : value, property : value, property : function() { } }

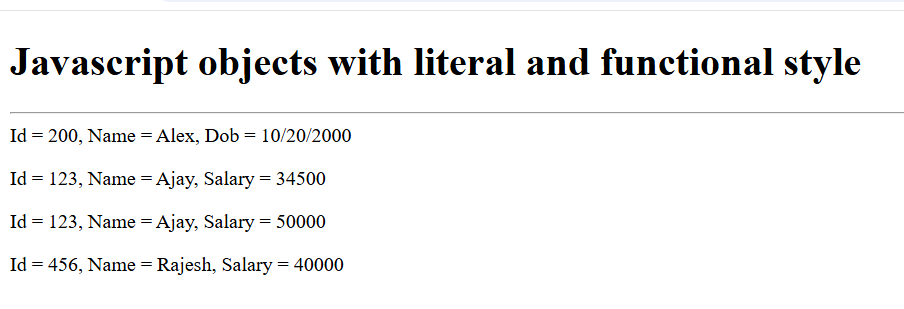
Function style: When you want multiple objects to be created with the same template then this is better, this style is more preferred in Javascript

function FunctionName(parameters) { … }

js-objects.html

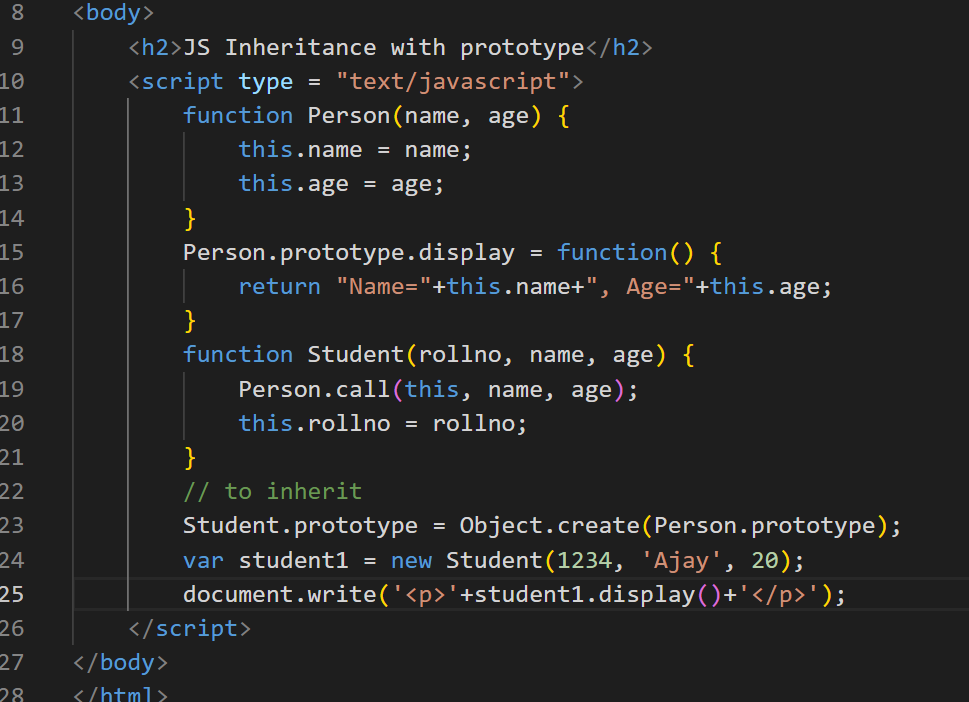


Output:



How to achieve inheritance in javascript

We can achieve inheritance with prototype if you are using old syntax, and a newer syntax uses classes & extends keyword



Output:



Arrays: It is a container object to store multiple values of same types or different types (in javascript it is allowed)

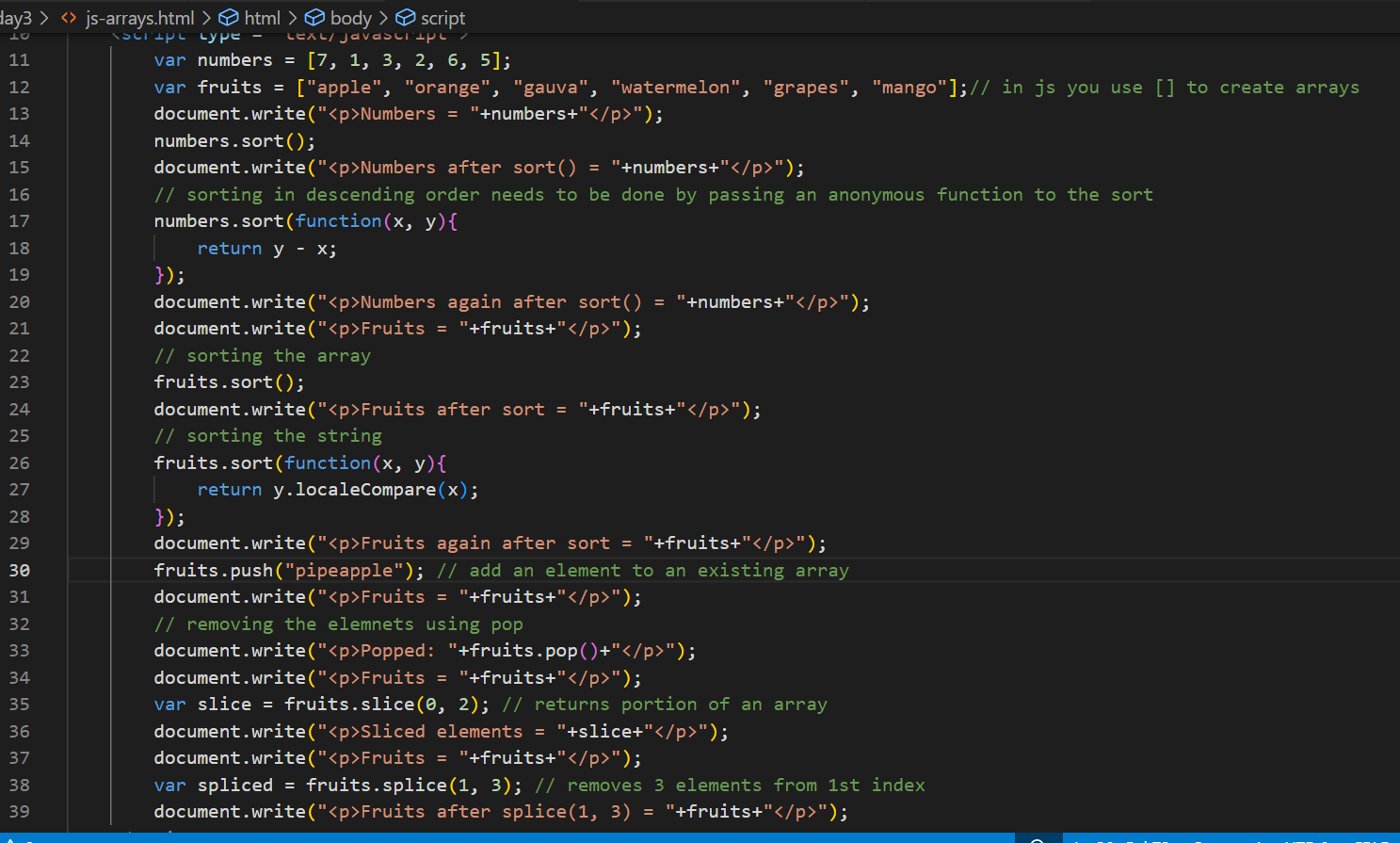
You can create simple to complex arrays

In Javascript you have lot of inbuilt functions in arrays like

push(): to add the element in the array  
pop(): to delete the element from the array  
sort(): to sort the elements  
slice()  
splice()

sort(): by default it sorts in ascending order, to sort in descending order you must use a function as a parameter that returns an int value to the sort to arrange the elements

sort(function(x, y){   
 return   
})  
Arrays methods & sorting with comparator anonymous function

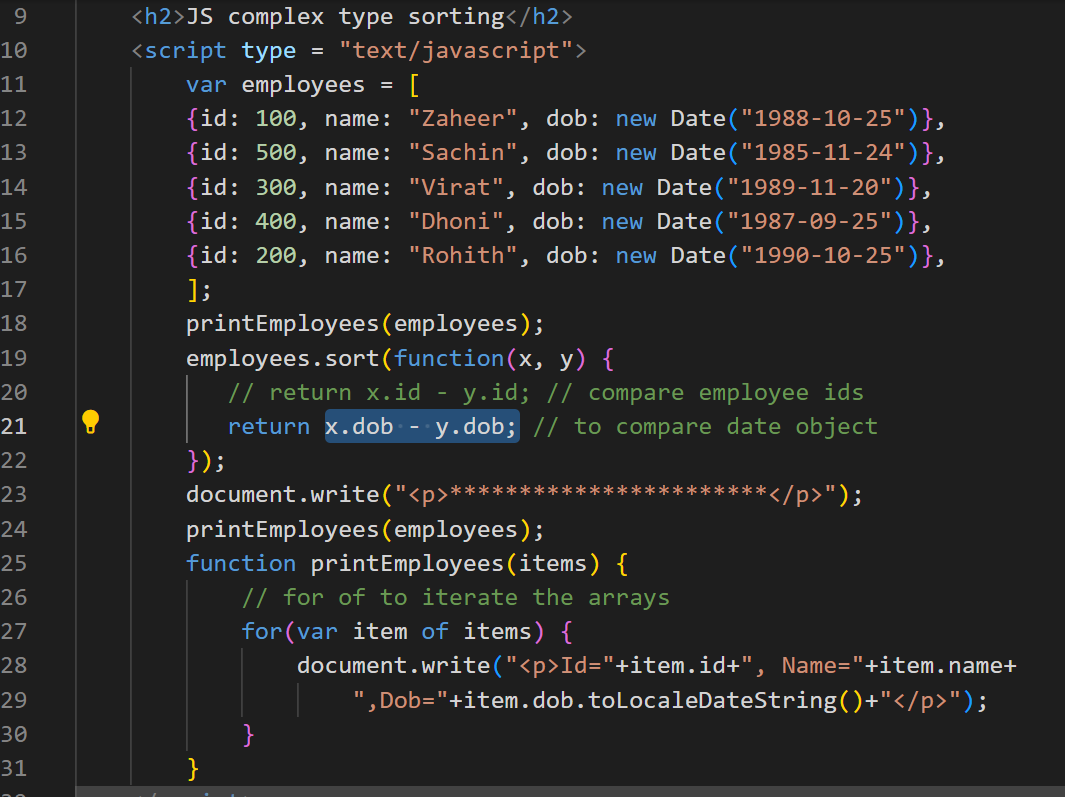


Output:

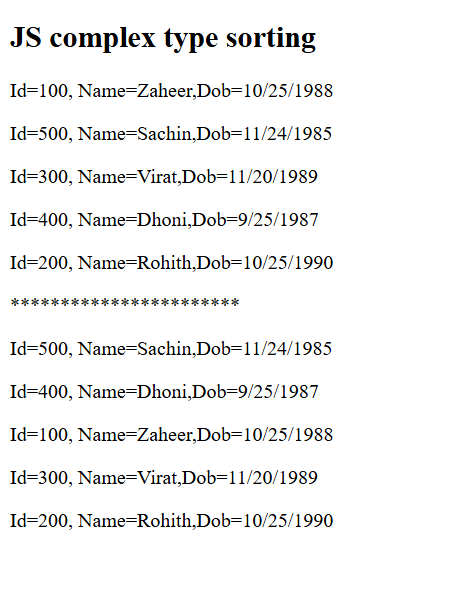


How to sort complex objects

You must use the same anonymous function and access the property of the objects and compare them.



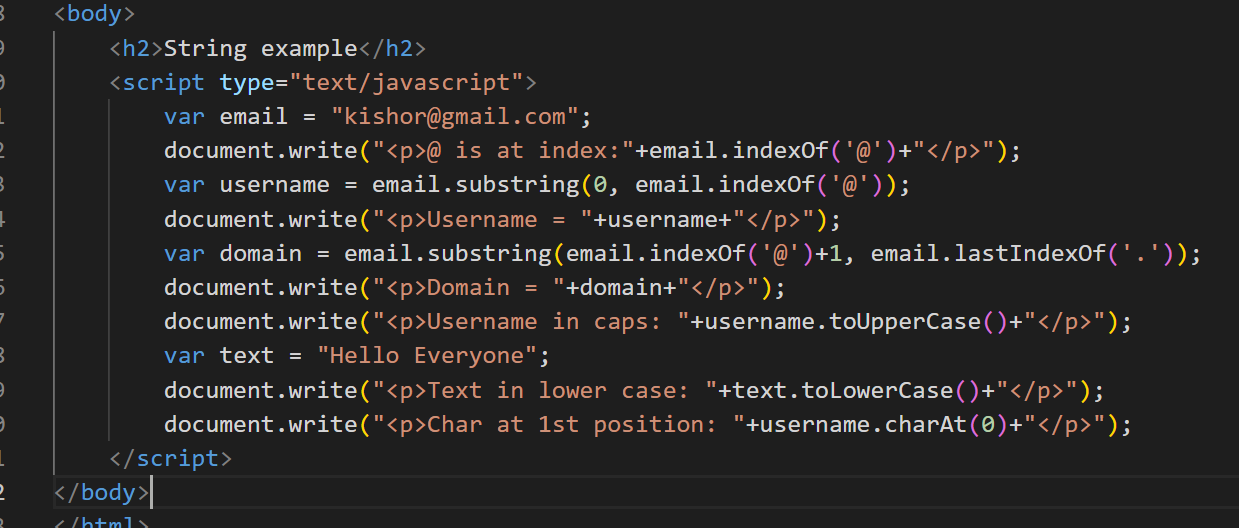
Output:



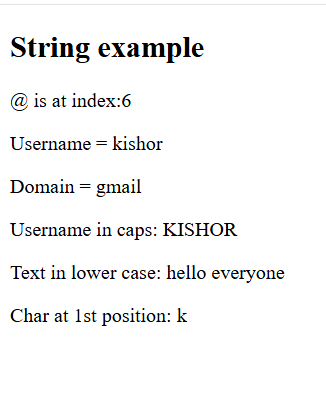
String methods:

In String there are various inbuilt methods like-

toUpperCase(), substring(), concat(), indexOf(), lastIndexOf()



Output:



Day 4 Agenda

**JavaScript HTML DOM**

* Introduction
* DOM Elements
* DOM EventListener

**JavaScript Form Validation**

* Create Form
* Styling Form
* Validate Username Input
* Validate E-Mail Input
* Validate Password & Radio Button

For in Loop

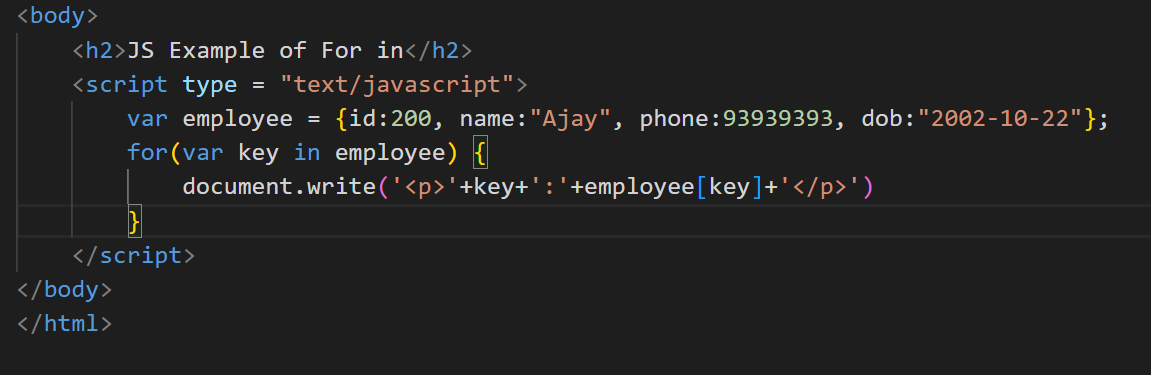
It is used to iterate the properties of an object.

let employee = { id:101, name:”Raj”, phone:939933, dob:”2004-10-22” };

let name = employee[“name”];

for(var key in employee) {   
 // key will be the property of an object

employee[key]  
}

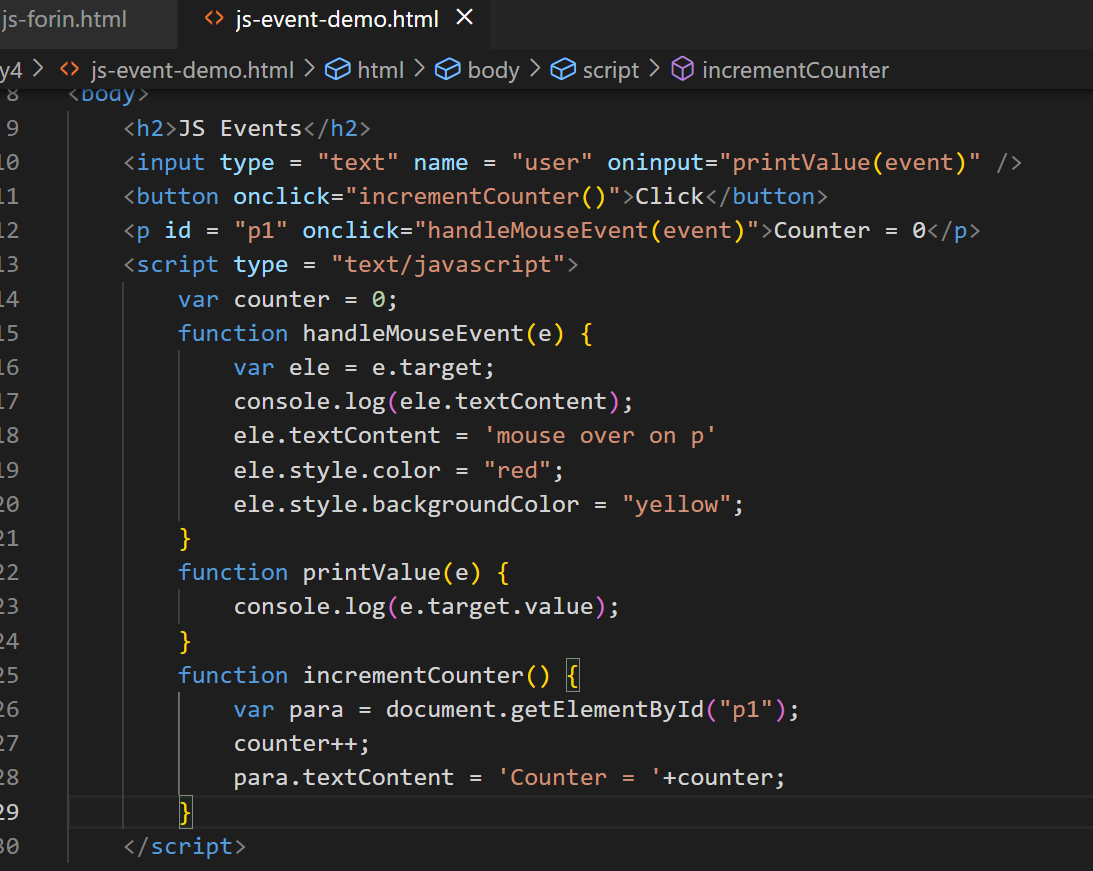


DOM events

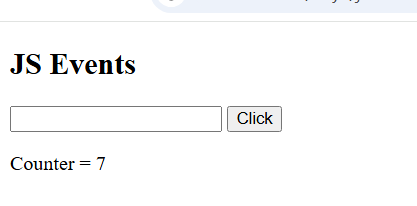
DOM stands for Document Object Model, it is a tree structure of HTML elements loaded in the browser, HTML elements would generate events when you do something with the element like mouseover, click, onchange, onsubmit

<button onclick=”fun()”>Click</button>

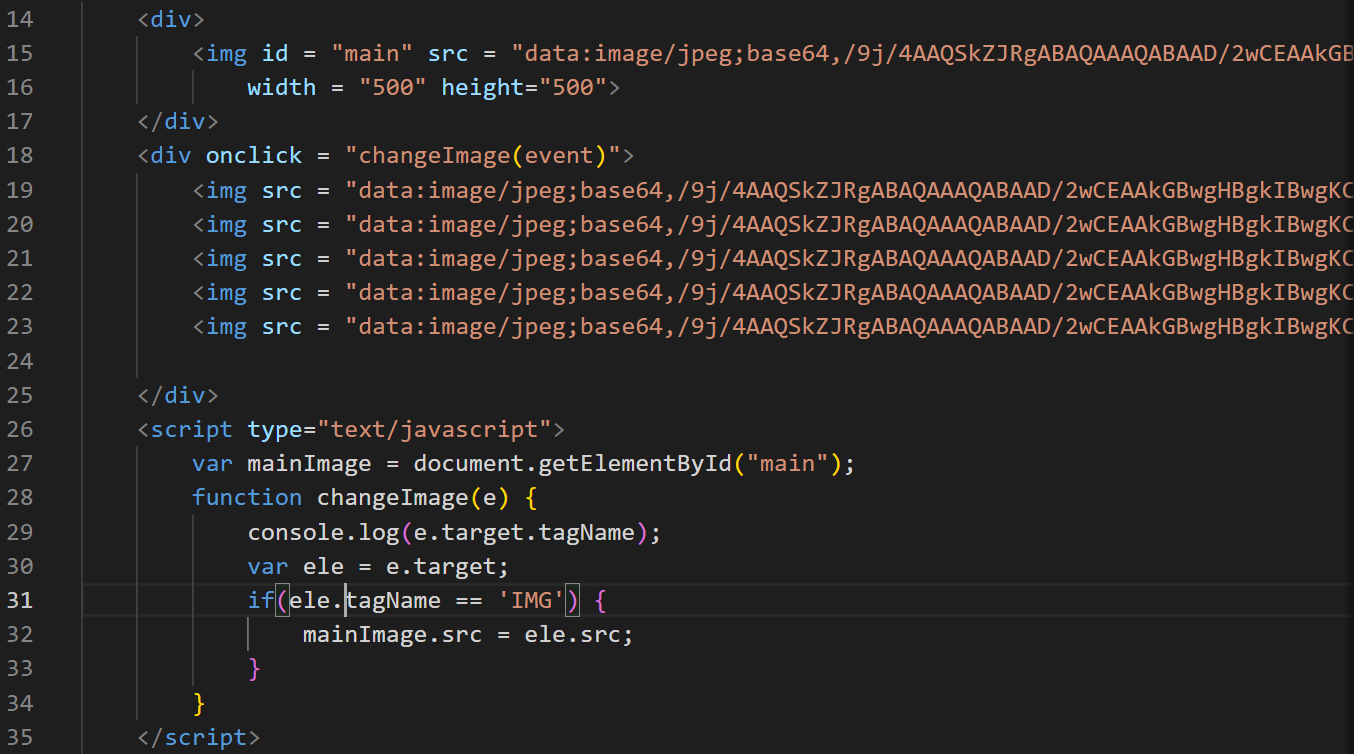
js-event-demo.html



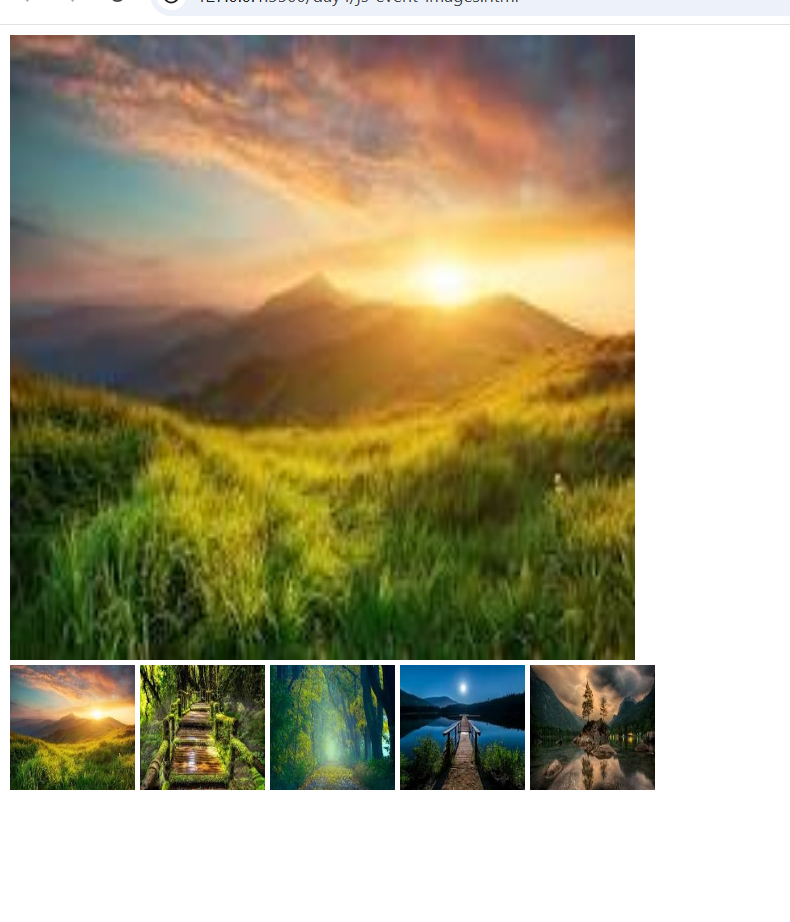
Output:



Changing the images based on the image you select



Output;



List of ways to access the element

1. document.getElementById(“id”): Pass the id to get the element
2. event.target: Get the element who generated the event
3. document.getElementsByTagName(“tag”): You get all the elements having the tag name in the form array
4. document.getElementsByClassName(“className”): You get all the elements having the class name in the form array
5. document.querySelector(“selector”): You can use a single method to get the elements by passing the id or class name or tag name.
6. document.querySelectorAll(“selector”): You can get all the elements having the same selector

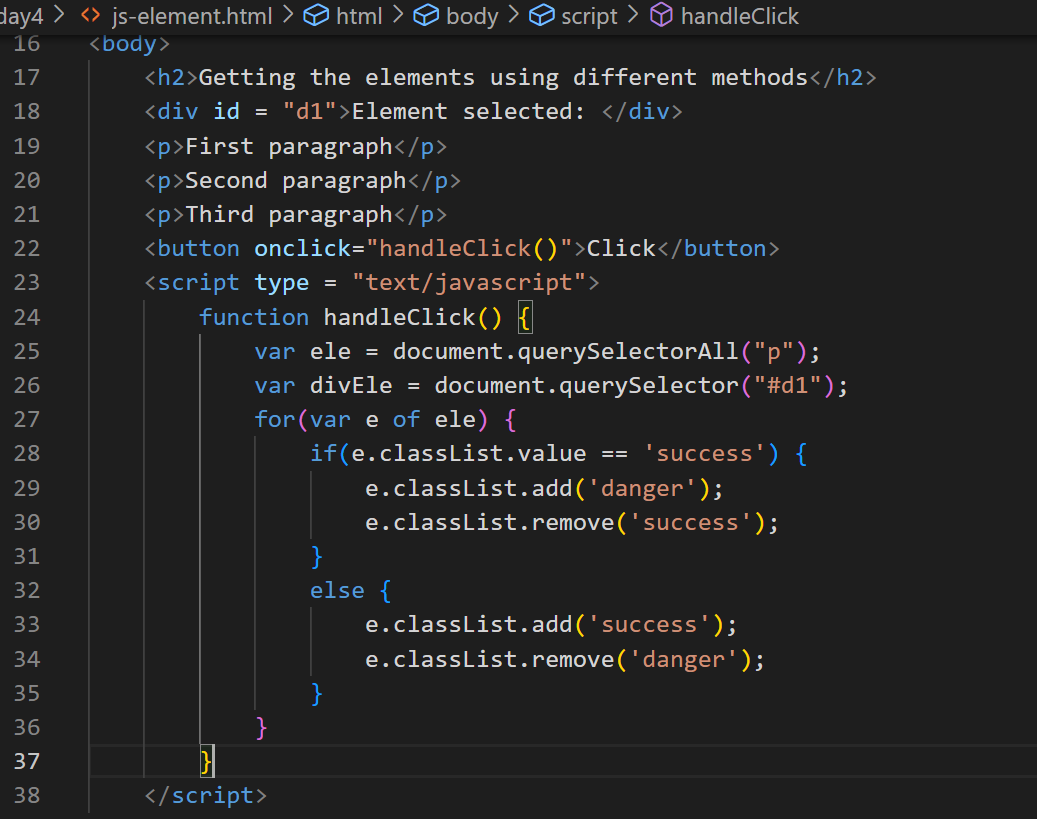
ex: document.querySelector(“#p1”): gets the element whose id is p1d

ex: document.querySelector(“.p1”): gets the elements which has the class name p1

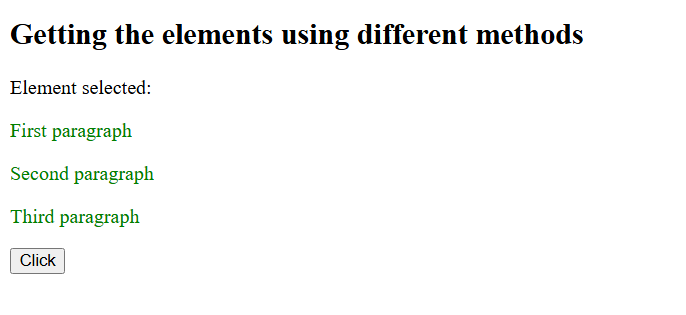
ex: document.querySelectorAll(“.p1”): get array of elements using the same class name

ex: document.querySelectorAll(“p”): gets all the p

ex: document.querySelector(“p”): gets the first p element in the DOM



Output:



Form Validation

When user enters input sometimes he/she can enter invalid inputs like empty username or weak password, not selecting the gender or entering an invalid email in that case you don’t have to submit the form to the server, you can do a client side validation and prevent the form submission.

Activity:

Create password input box and validate for stronger password it must have digits, lowercase, uppercase, special character

Create radio button and mandatorily one should be selected

Create an input box for the email and validate for the valid email

Day 5

**Modern Javascript (ES6, ES7 and others)**

* Classes
* Let & Const
* Template String literals
* Arrow Functions
* Spread syntax & Rest parameters
* Destructuring
* Promises
* Async/Await
* Generators
* Optional Chain
* Nullish coalescing
* String padding

Modern Javascript simplifies the Javascript syntax so that most of the codes looks easy to understand.

Old Javascript syntax

var name = “Ajay”;  
var age = 35;  
var phone = 993920393;  
var gender = “Male”;

var information = “Name = “+name+”, Age = “+age+”, Phone = “+phone+”, Gender = “+gender;

Modern Javascript syntax

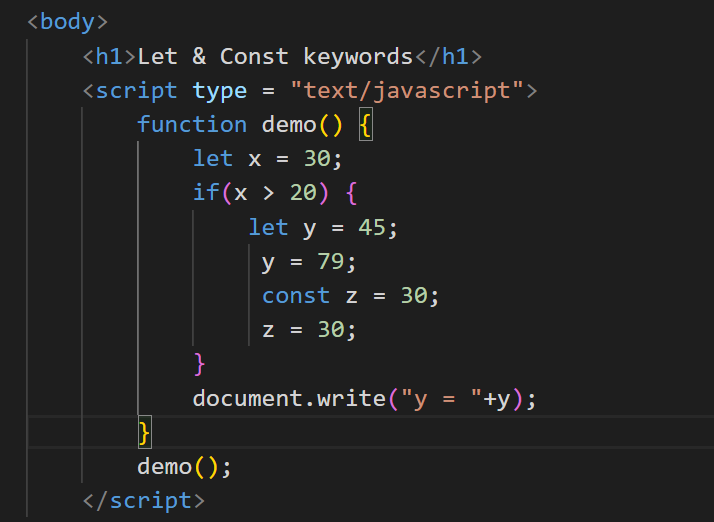
information = `Name = ${name}, Age = ${age}, Phone = ${phone}, Gender = ${gender}`;

Javascript is derived from a specification called EcmaScript, it has released ES6 version which changed Javascript syntax to look much easier to understand, following are the features introduced in ES6 or later.

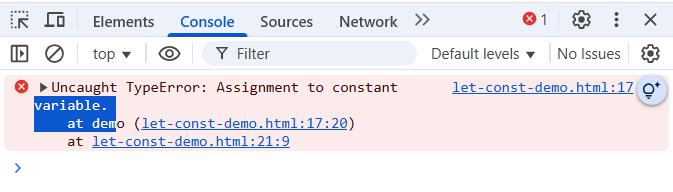
1. let, const, class, extends, super keywords
2. Template strings
3. Arrow functions
4. Spread & Rest parameters
5. Destructuring
6. Promises
7. Exponential operator
8. async/await
9. Generators
10. Trailing commas
11. Optional chain
12. Nullish coalescing

let & const: These allow you to create block scoped variables, which you can use instead of the var keyword, because var is not block scoped

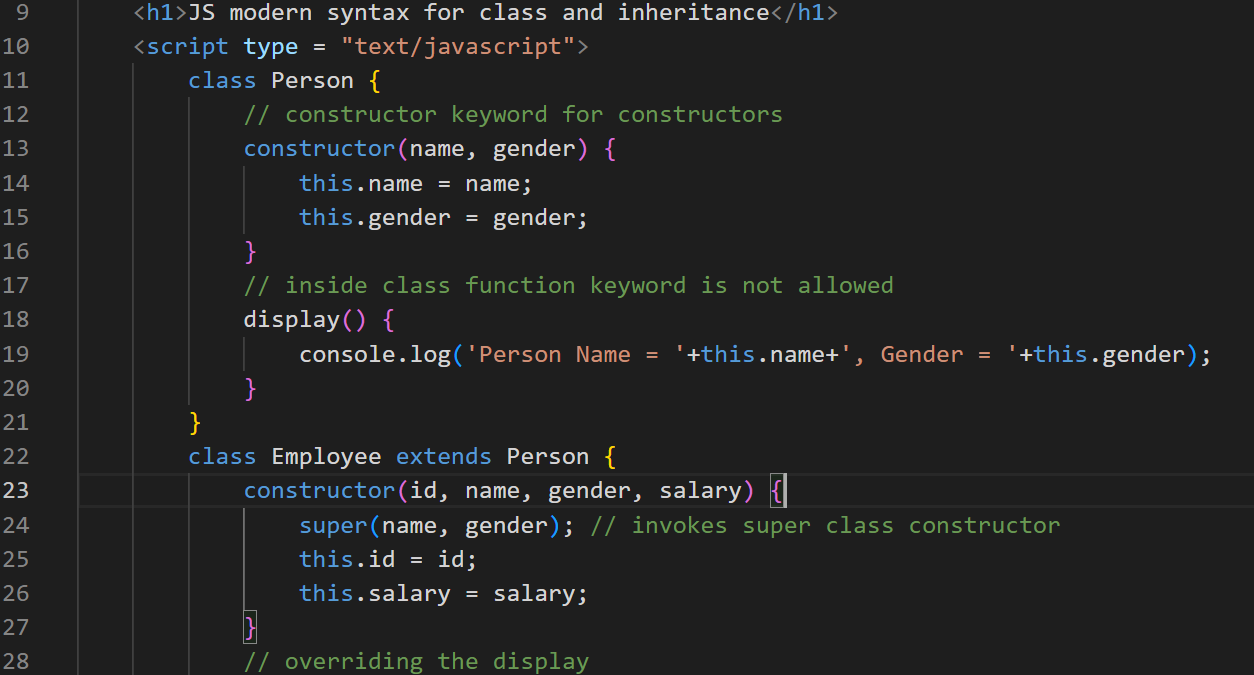
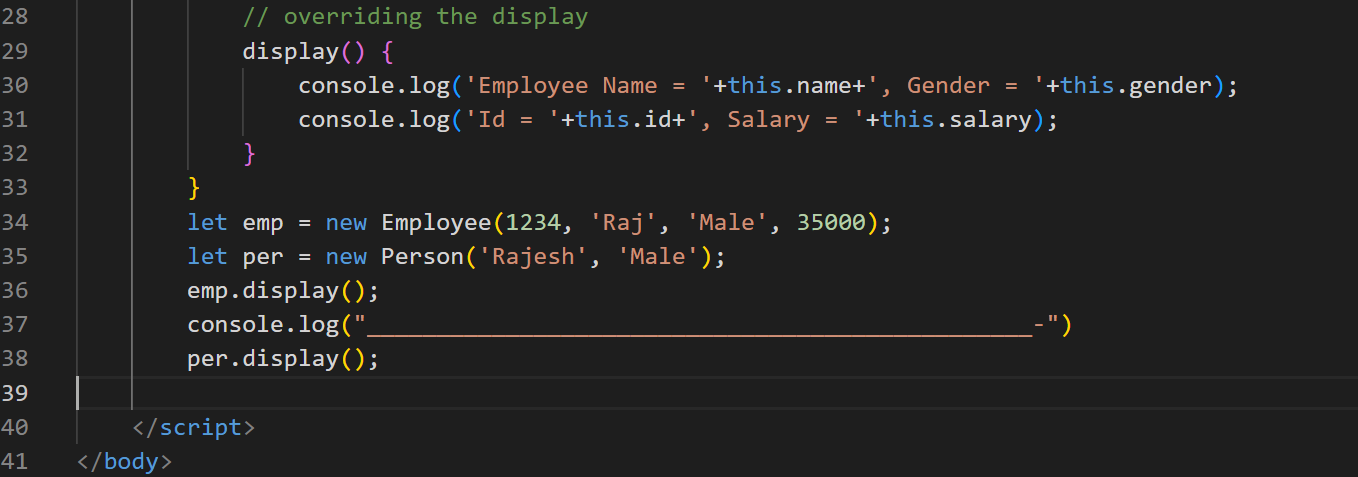
if(…) {   
 var price = amount \* total;  
 var price = amount \* total; // you are allowed to declare more than once  
}  
console.log(price); // but it will be accessible



Output:



Achieving inheritance

Output:



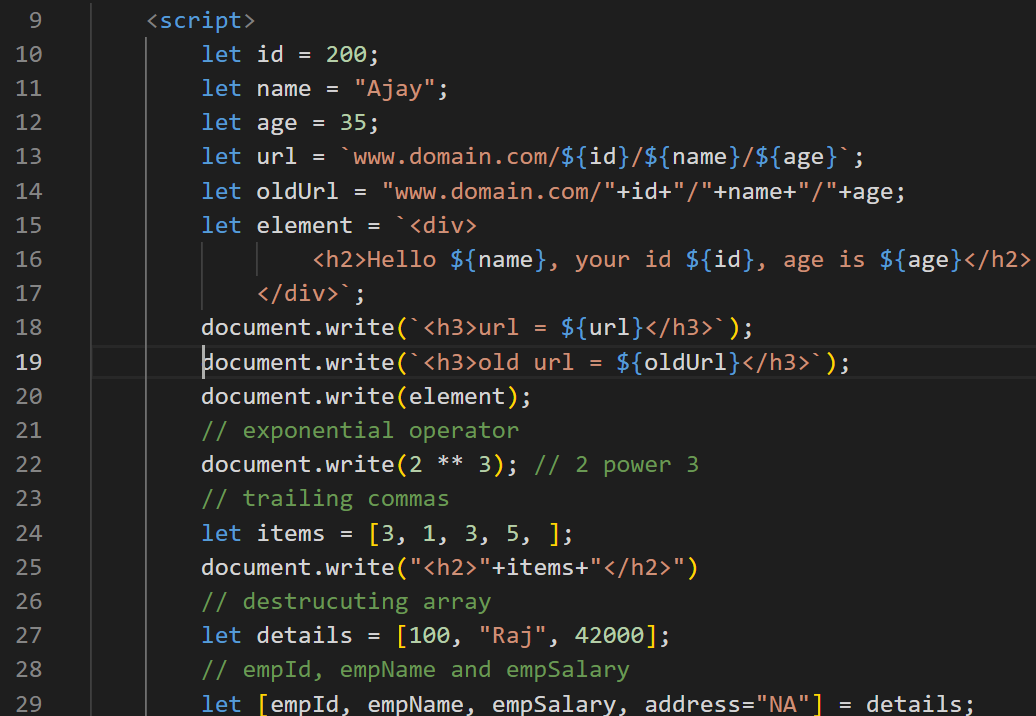
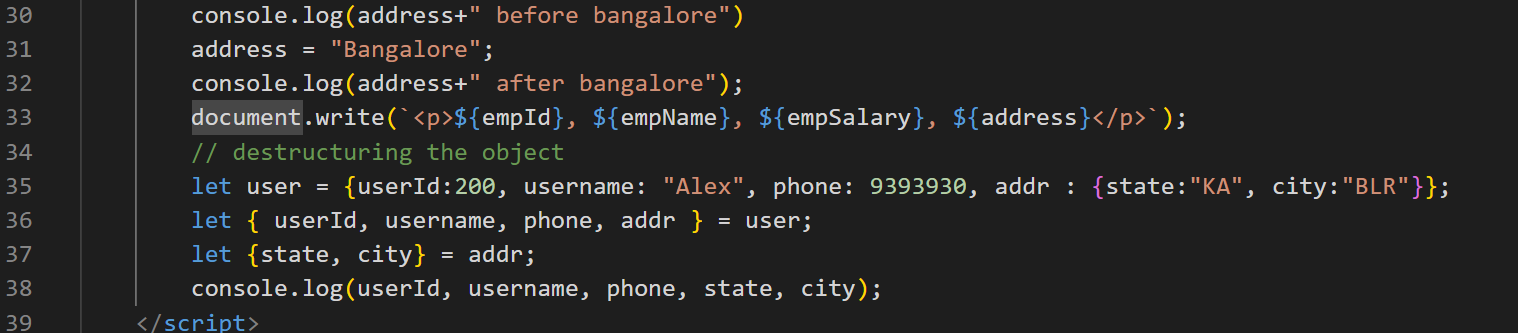
Template Strings

It helps to create strings with js expressions in a simple without breaking with + operator



Output:



Day 6

Rest & Spread parameters

In order to avoid a function losing the data we have these parameters

Rest: You can create a function that can accept any number of parameters

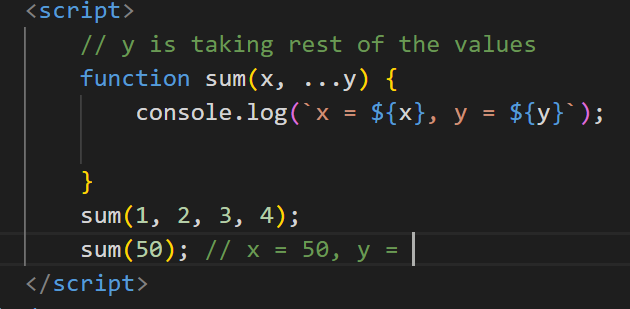
function sum(x, y) { }

sum(2, 3, 4, 5); // error or takes only few parameters

Since 4, 5 is lost we can create a Rest parameter that will be … (3 dots) before the variable name

function sum(x, …y) { }  
sum(2, 3, 4, 5); // x = 2, y = 3, 4, 5

Rest parameter



Output



Spread operator

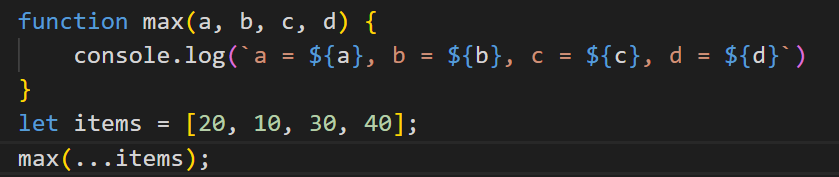
You want to distribute the array of values to multiple parameter, then you can use spread operator, it is used at the time of invoking the function, even spread also uses 3 dots

let items = [2, 3, 5];

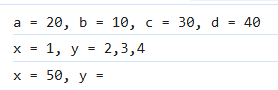
function max(a, b, c) {   
}

max(items); // a = [2, 3, 5]

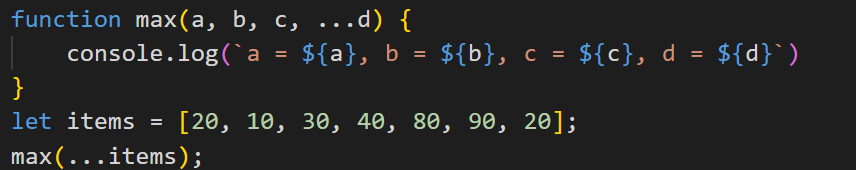
max(…items); // a = 2, b = 3, c = 5



Output:



We can combine rest & spread both as below



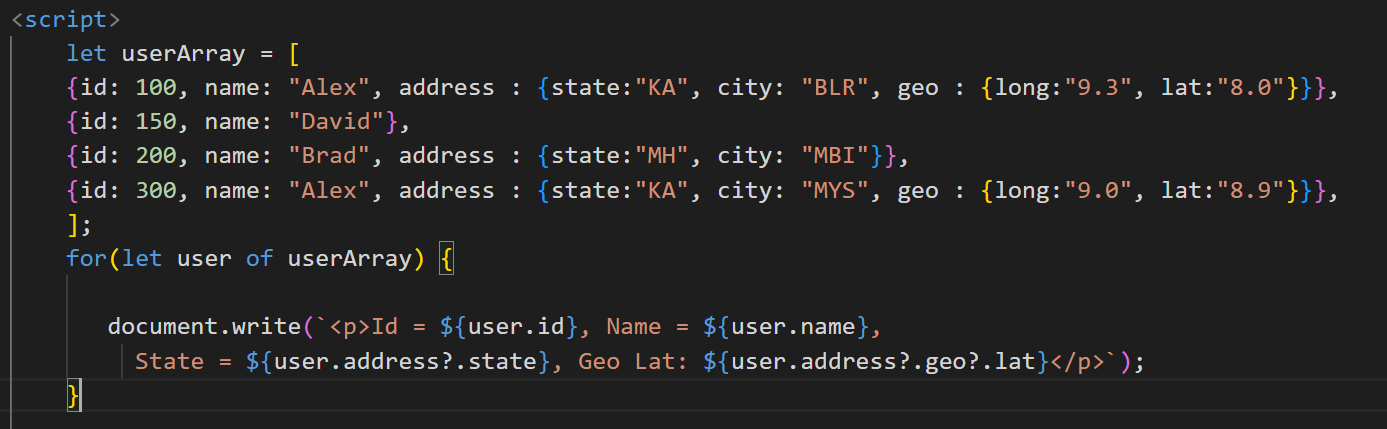
Output:



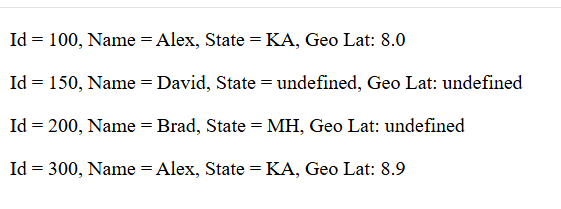
Optional Chain & Nullish coalescing

Optional chain

This is used to avoid errors while accessing the deeply nested properties



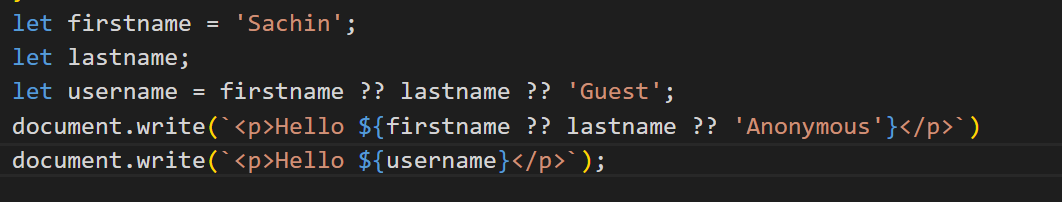
Output:



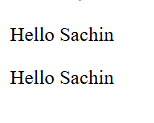
Nullish coalescing ??

You want to check the value is undefined or null & assign a different value when it is null or undefined

let x;  
let y;  
let z = x ?? y ?? ‘Anonymous’;



Output:

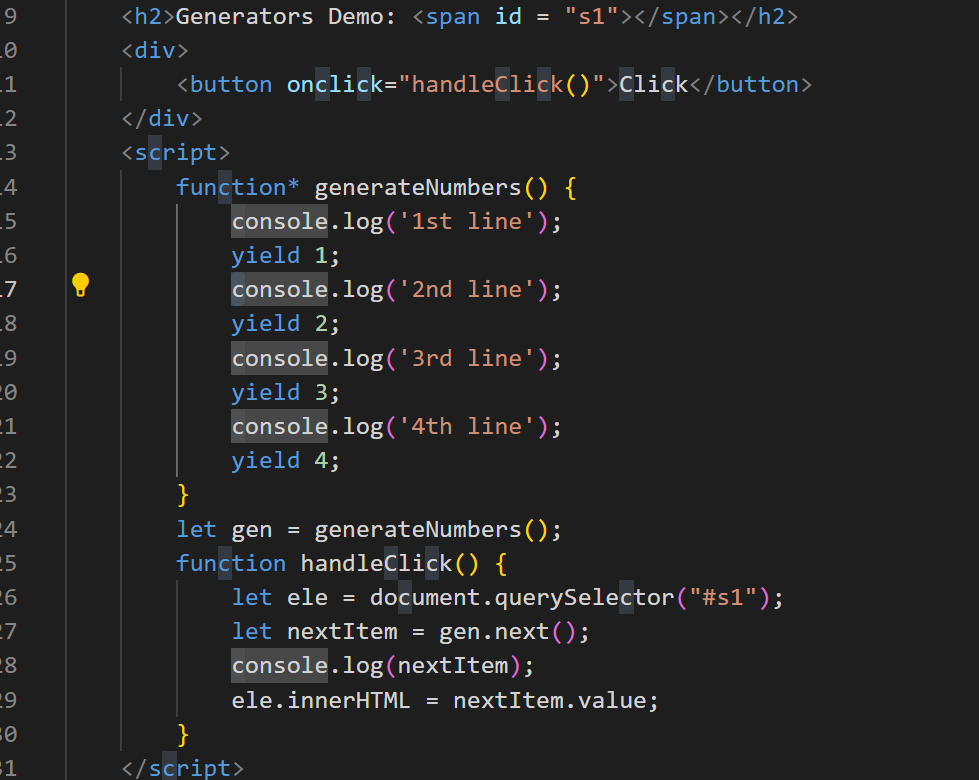


Generators

These are special type of functions which can pause the execution of the function and resume, when it pauses it can return a value as well using the yield keyword

function\* generateNumbers() {   
 yield 1;  
 yield 2;  
 …  
}

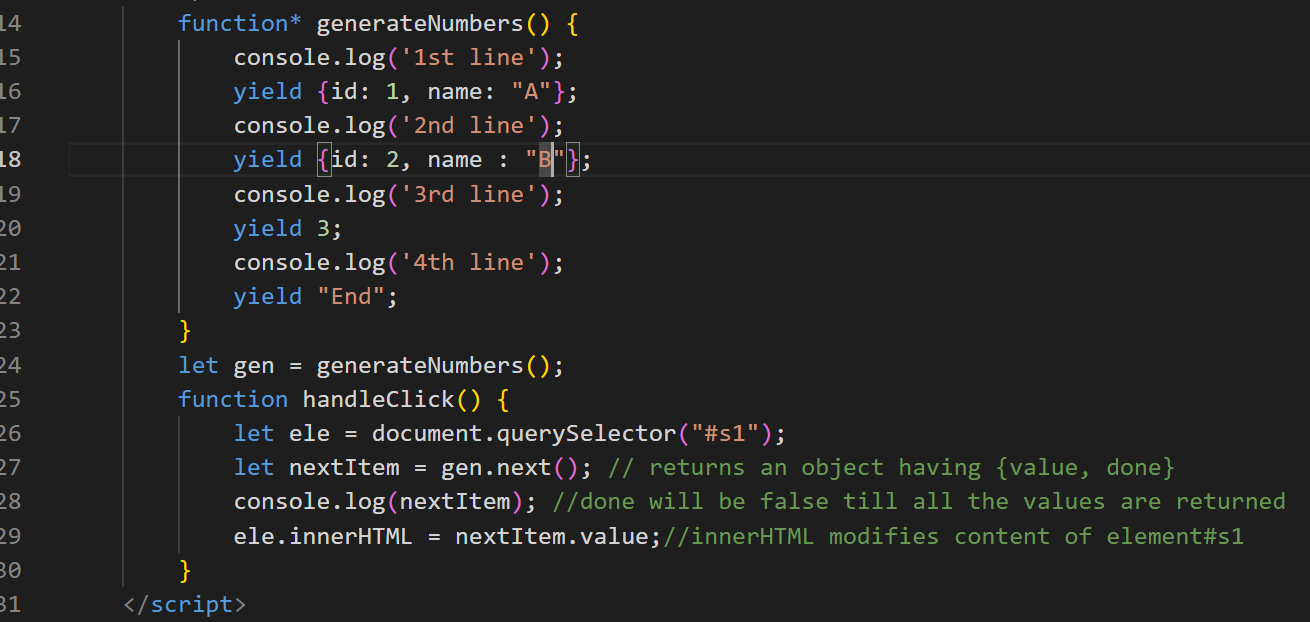
let gen = generateNumbers();  
gen.next() // it executes until it finds the yield & pauses there



Output:



We can yield some other type of values also



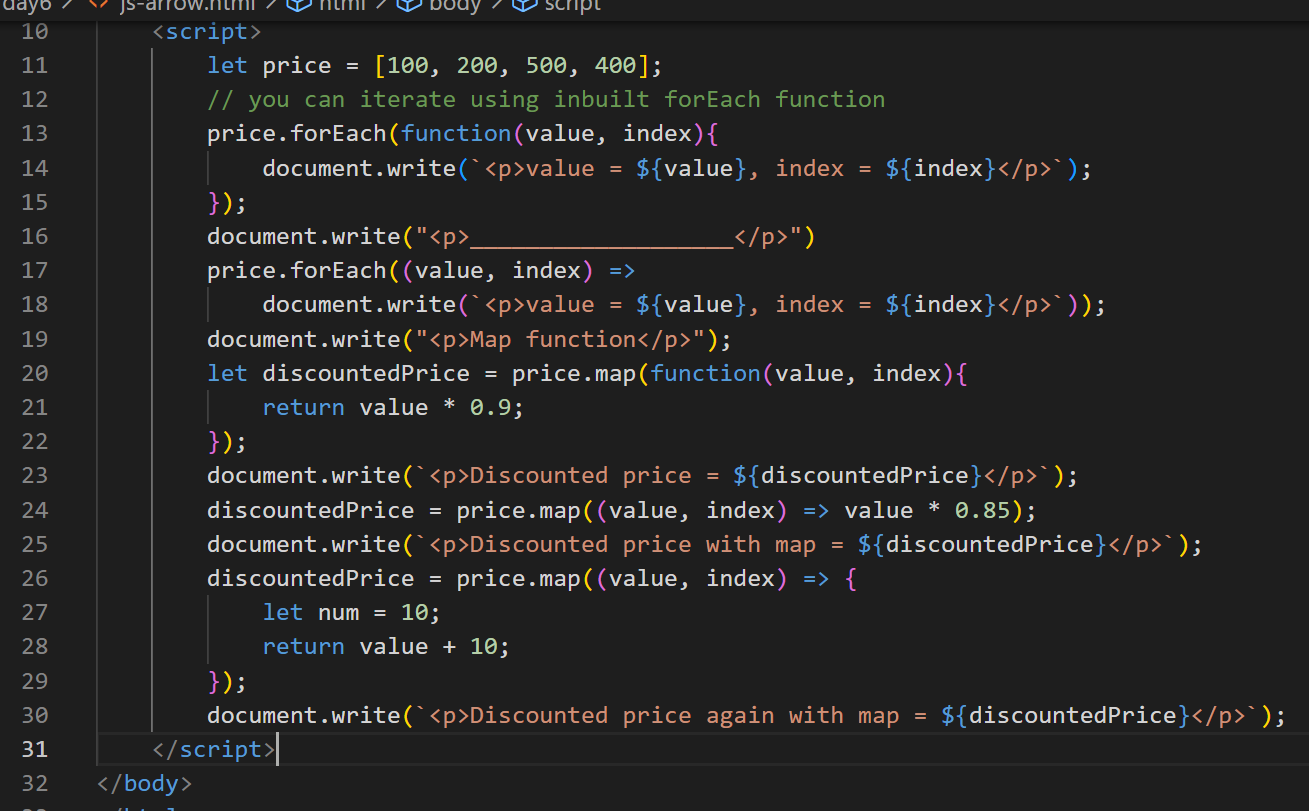
Output:



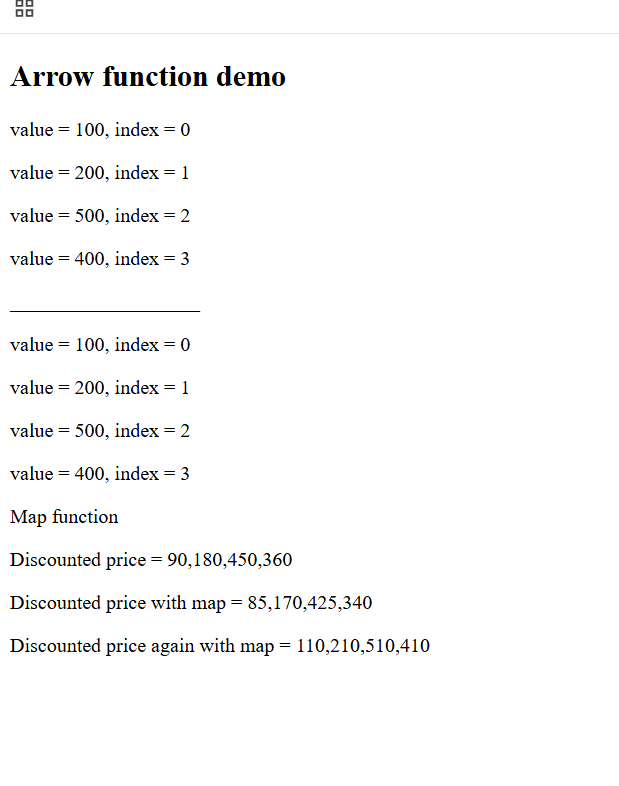
Arrow functions

These are the simplified functions for anonymous functions or callbacks

Callback functions are not immediately executed, they will be initiated immediately but will be called later.



Output:



Day 7

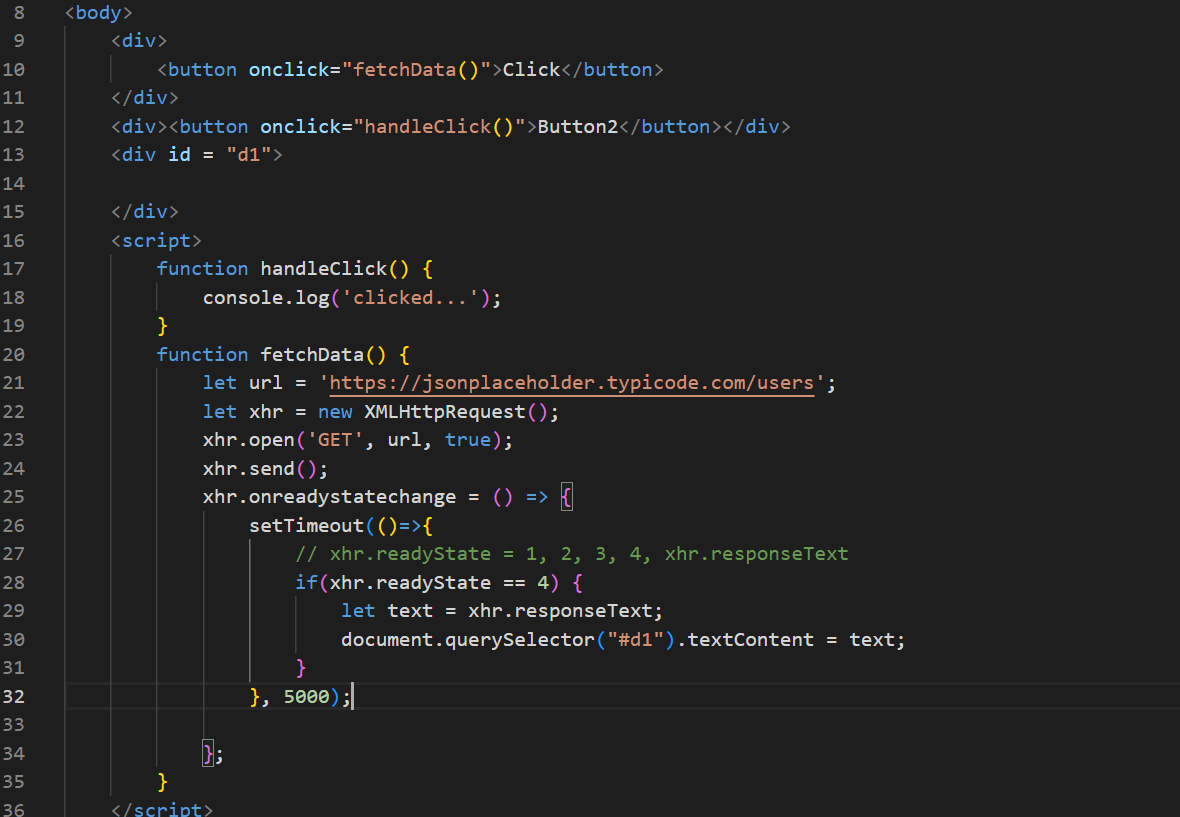
* Promises
* Async/Await
* String padding
* NPM basics
* Installing packages
* package.json
* Node Event Loop
* Core Module
* Local Module
* Reading & Writing Files
* Third party Module

Asynchronous actions:

These are the operations which you initiate but they are finish later, they should not block the user when the operations are performing behind the scene

ex: timeouts, event handling, calling the backend apis

XMLHttpRequest: This is used to make an HTTP requests.



Output:



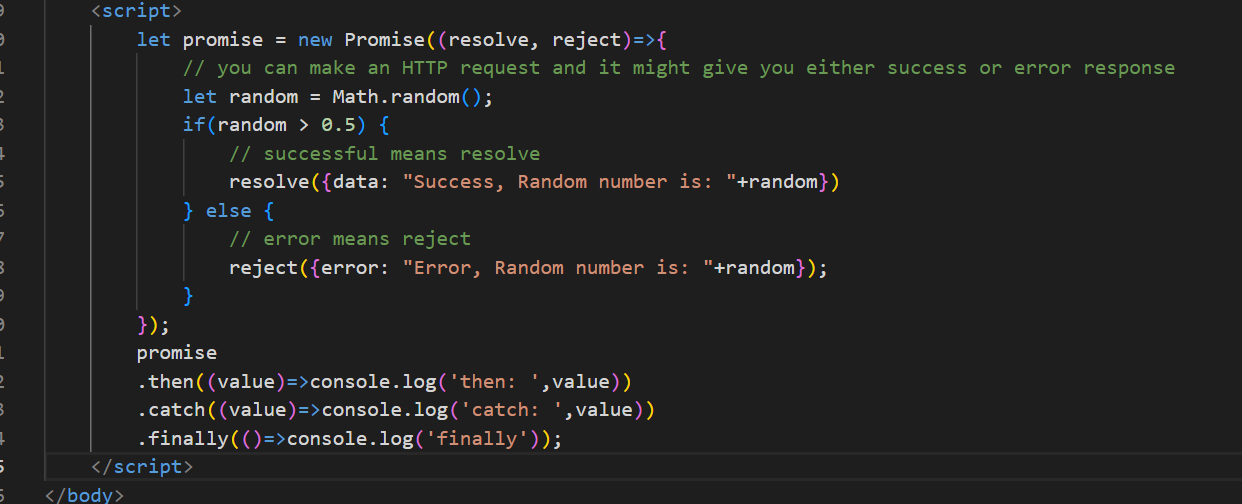
Promises:

They can perform asynchronous operations by providing separate functions to handle success and failures, it also avoids nesting the callbacks when current result is dependent on the previous result, Promise has 2 states

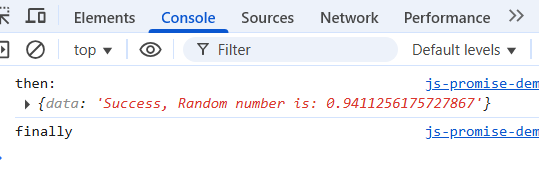
1. Pending state: Initial state, neither successful(resolved) nor failed (rejected)
2. Settled state: Resolved or Rejected

ex: axios library in React.js, fetch() which is supported in Javascript & Node.js (21 or later)

We can create our own Promise using the Promise constructor



Output:



Previously we sent the request using XMLHttpRequest, but modern javascript provides an inbuilt method called fetch() that can send the Http Request but it returns the promise



Output:

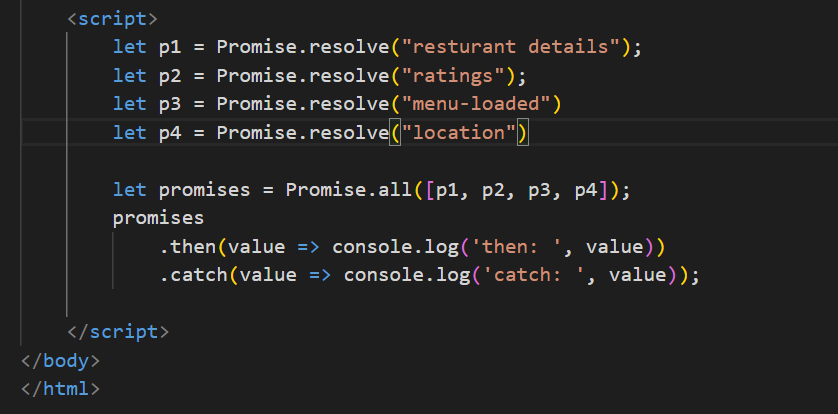


Promise can also perform multiple asynchronous operations and settle these operations in many ways, it provides various methods to take multiple promises in the form an array and settles based on the methods you use, below are the Promise methods which are static

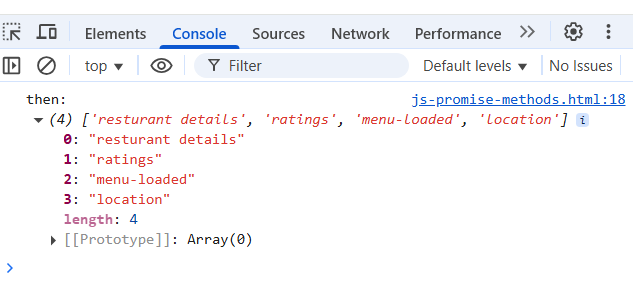
1. Promise.all([p1, p2, p3])
2. Promise.allSettled([p1, p2, p3])
3. Promise.race([p1, p2, p3])
4. Promise.any([p1, p2, p3])
5. Promise.resolve()
6. Promise.reject()

Promise.all([p1, p2, p3]): It takes multiple promises & returns a promise only if all are successful, but if any one fails it will return the failed promise and rejects the remaining promise

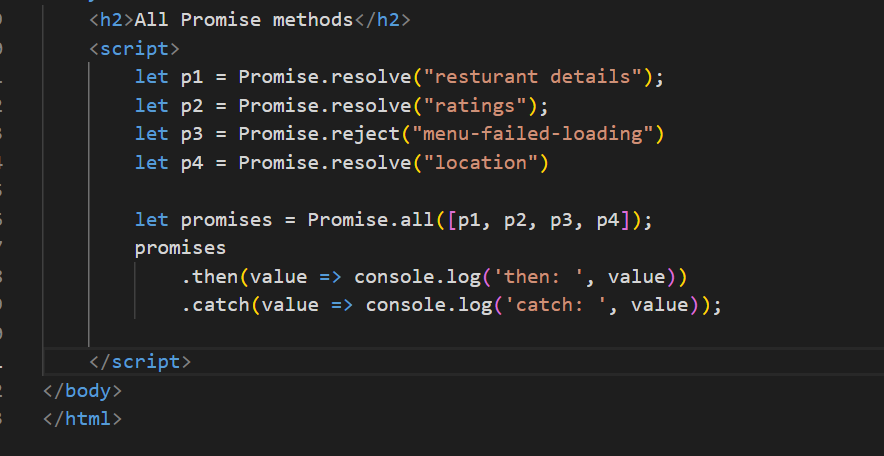
ex: Loading the restaurant details, menu details, ratings and so on



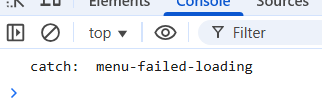
Output: Since all are successful you get all the fulfilled promise



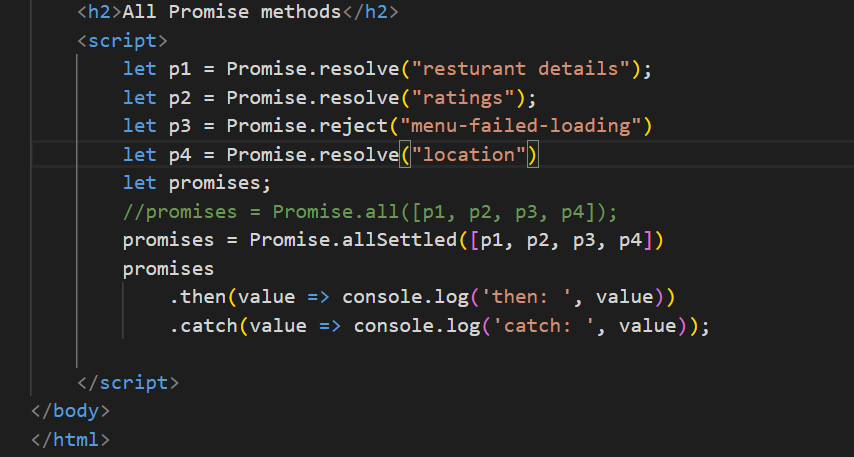
Change the Promise.resovle to Promise.reject you will see it only gives the rejected promise even though other promises are fulfilled, it gives the very first promise that is rejected & ignores the other promises



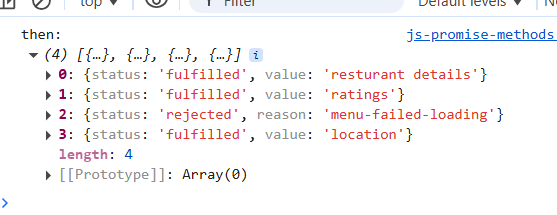
Output:



Promise.allSettled([…]): It settles all the promises both resolved & rejected one and returns a promise that will have all the promises result with which is resolved & rejected status

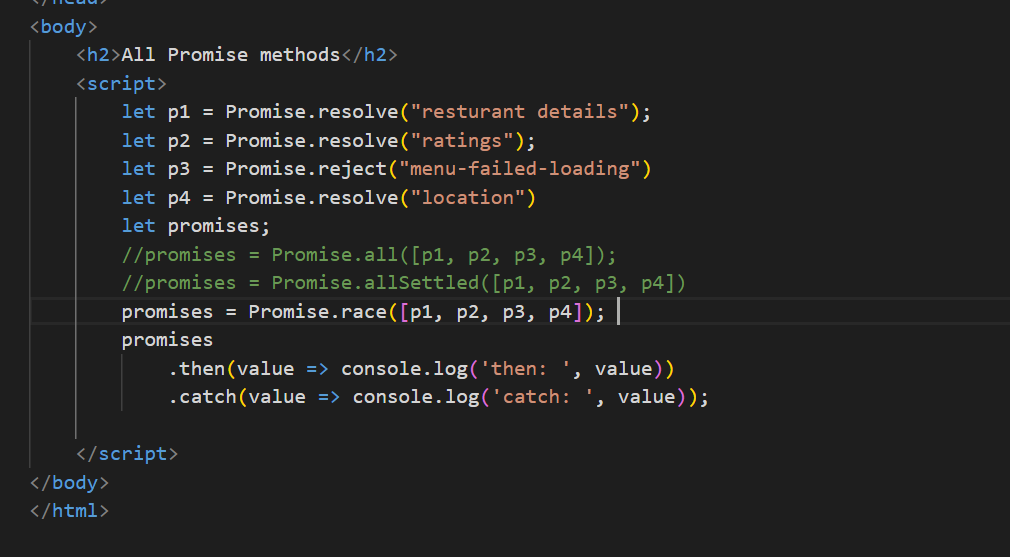


Output:

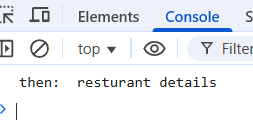


ex: You want multiple async operations to perform like rating the food, rating the delivery guy, rating restaurant

Promise.race([…]): Gives the promise whichever is settled first either rejected or resolved



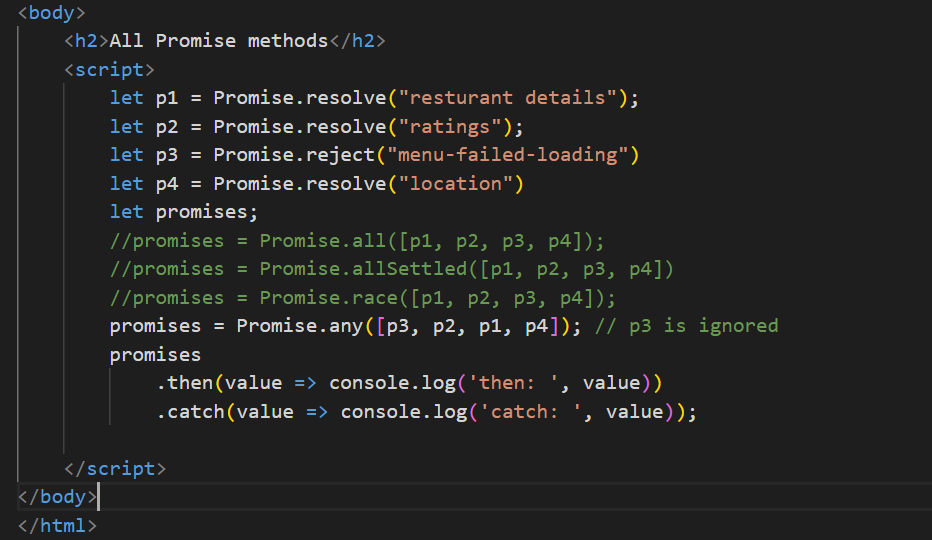
Output:



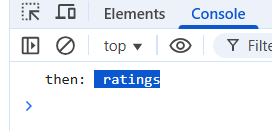
Here always the first promise itself will be the output, but in real time, when you pass multiple promises which might finish first we will not have an idea, use race() when you want the earlier one to finish first and it doesn’t matter it fails or succeeds

ex: You want to make an API call and also you want a timer to run, then whichever is finished first you want to consider

Promise.any([..]): This always gives the promise that is resolved by ignoring any failed promise



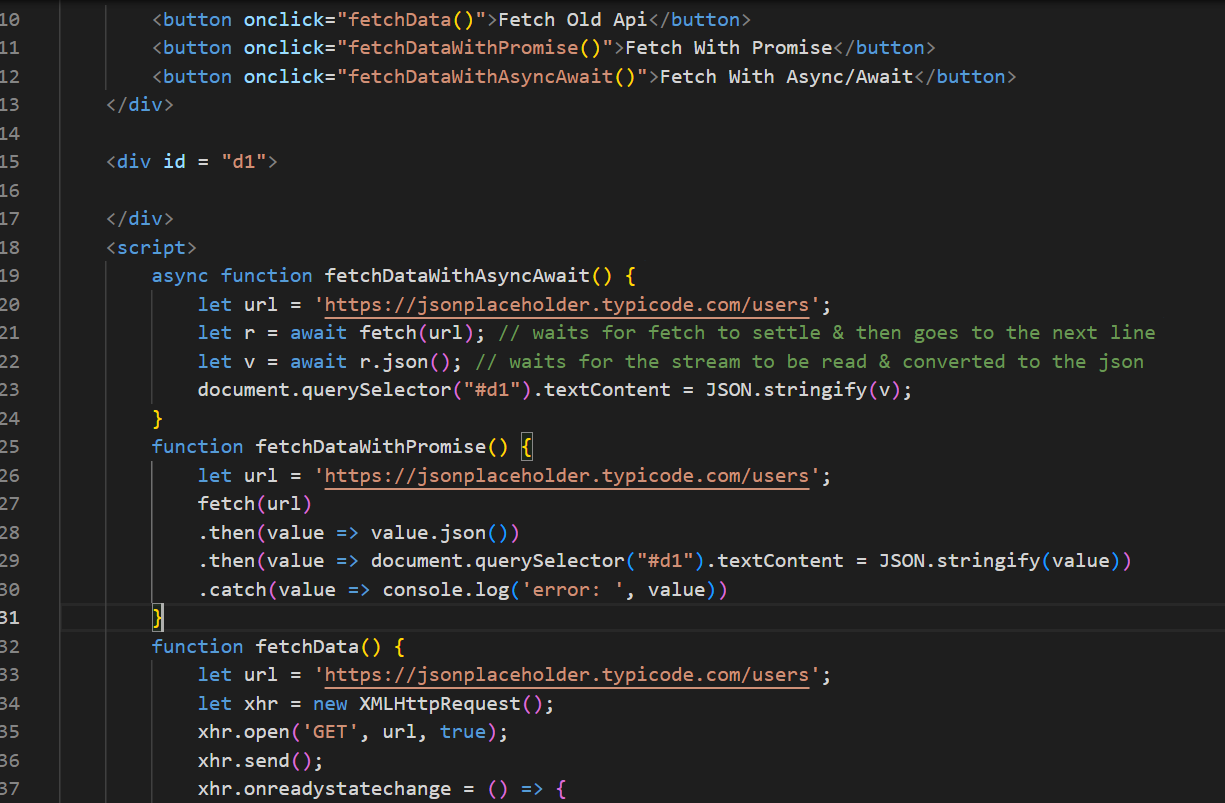
Output: Since p3 is rejected it is ignored & p2 is resolved it is considered and rest of the promise is ignored



Ex: When you have some common resource getting from multiple end points, you can consider the first promise that resolves

Async/Await:

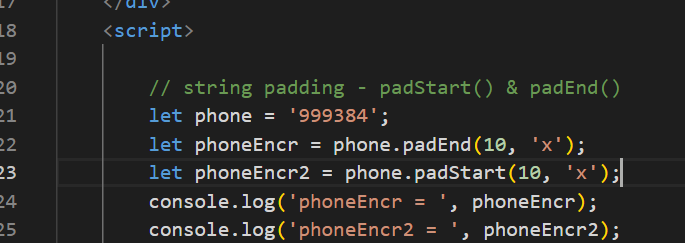
This is built on top promise to wait for the promise to settle and proceed with the next task, a function with async keyword can return a promise so that while calling you can use .then & .catch.



Output:



String padding

  
output:

