React

It is a Javascript library used to develop single page applications, it follows component based approach to develop the UI.

Single page application

It is an application where you can see everything in one page, if user clicks on a button or a link or any element only part of the page will be loaded instead of loading the entire page.

ex: Facebook, Git, Twitter, Git hub and so on.

Component: These are reusable UI’s which you can see in the page and they can be nested in other components, you can develop and maintain the components independently and fit the components wherever you need.

Note: React.js is used only to develop User Interface.

Advantages of React.js

1. It is easy to learn and understand
2. It uses components which are easy to maintain and test
3. It can be used to develop UI’s for many platforms like Mobile, Web and DVR.
4. It is a library and encourages you to add any javascript libraries you wish ex: axios to call back end, bootstrap js to style or materials to style and so on.
5. It provides javascript libraries to perform unit testing.

Pre-requisites

1. HTML
2. CSS
3. Javascript
4. New Features of Javascript (ECMAScript or ES6 and latest)
5. JSX (Javascript XML, extended version of Javascript).

Softwares required

1. Visual Studio Code editor
2. Node.js (Javascript Runtime Environment)
3. Browser

React.js is developed using JSX, which is an extended form of Javascript.

Why JSX

JSX is easy to write compare to Javascript when comes to developing UI’s i.e., writing HTML code is easier in JSX compare to Javascript

Brief Overview of Contents

* React.js
* JSX
* ES6+ Features
* VDOM
* Components
* Controlled Components
* Props & States
* HOC
* Render Props
* Advanced concepts
* Router
* Axios
* Redux
* Testing
* Materials
* Javascript or Node.js libraries i.e., Express

Understanding the new features of Javascript i.e., ES6+

ES6 onwards the javascript syntax has slightly changed.

ECMAScript or ES

* It is standard that will have set of rules & regulations which is implemented by Javascript, Node.js, Browser
* It has different versions like ES6, ES7, ES8 and so on
* In each versions new changes are added.

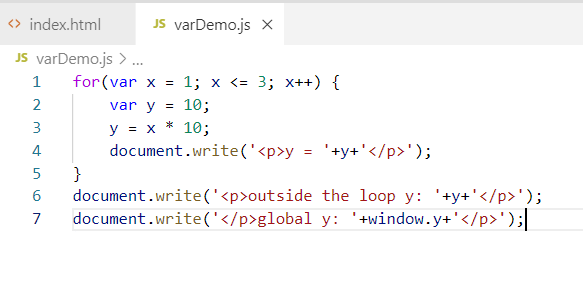
What are the main changes introduced in ES6 or later

* New keywords in javascript like let, const, class, constructor, super
* Template String literals
* Arrow functions
* Static members
* New functions in Object like entries & value
* New functions in String like padStart & padEnd
* Generators
* Object Destructuring
* Rest & Spread operators
* Default arguments
* Optional Chain
* Trailing commas
* Exponential Operator
* includes in Arrays

Understanding let & const keyword

let & const keywords are used to create block scoped variables, because earlier javascript had var keyword to create variables which would become global variable when you create.

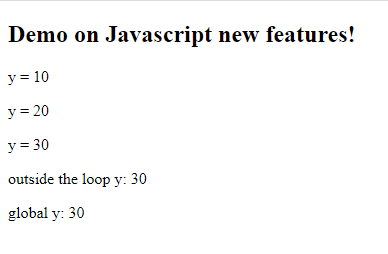
varDemo.js



index.html



Output:



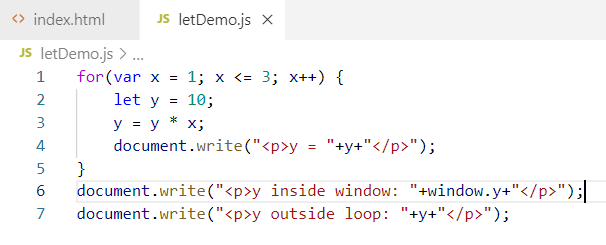
Here the problem is you can access the variable which is declared inside the loop i.e., outside the loop also they are accessible, which is not secure, hence we have let & const to declare variables.

let & const variables wouldn’t be added to window object.

let is used to create block scoped variables & you can modify it within the scope, but you can’t access outside the scope

const is used to create constants, it is also block scoped variable, but you can’t modify it.

letDemo.js

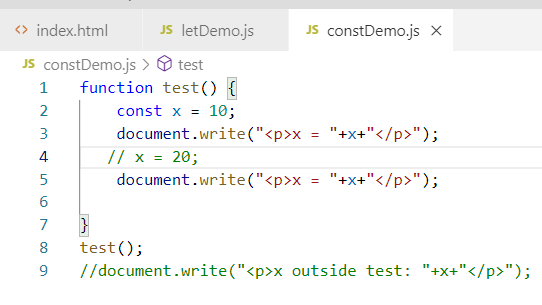


Output:



Since let keyword doesn’t add the variables inside the global object, the window.y gives undefined & also accessing the ‘y’ outside the scope gives error in the browser console.

constDemo.js

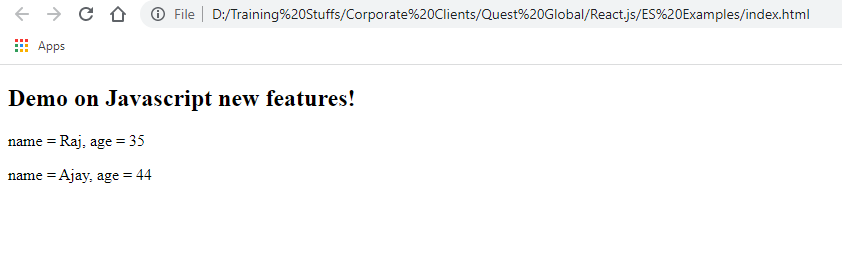


The above code gives error at the line no. 4.

But the object properties are not constants.



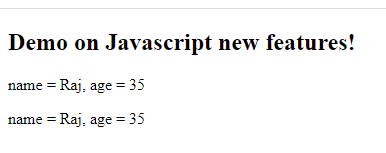
Output:



The above code lets you modify object properties though it is const, but to avoid modifying properties you need to freeze the object using Object.freeze.



Output:



The above code makes the entire object freeze so that you can’t modify this was there even before ES6.

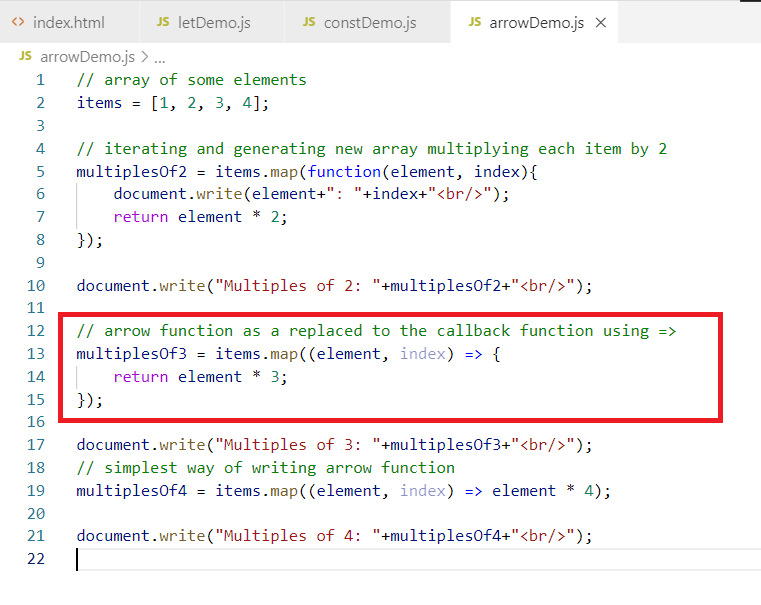
Differences in var and let

|  |  |
| --- | --- |
| var | let |
| It is global | It is block scoped |
| by default variables without any keyword takes var | by default variables doesn’t take let |
| You can redeclare same variable | You can’t redeclare same variable |



Arrow functions

These are used to simply writing callback functions with a simpler syntax that eliminates using function keyword, {} if it’s a single line statement, return keyword if it’s a single line statement.



Note: If in case the callback has only one line statement then you can avoid writing {}.

Valid arrow functions

() => “Hello World”: This takes no arguments & returns string

(x, y) => x + y: This takes 2 arguments & returns addition of 2 arguments

(x, y) => return (x + y): Not valid, use return inside { }

(x, y) => { x + y; }: This is also not valid, you must return the expression or create some valid statement, since in javascript a function need not to return always.

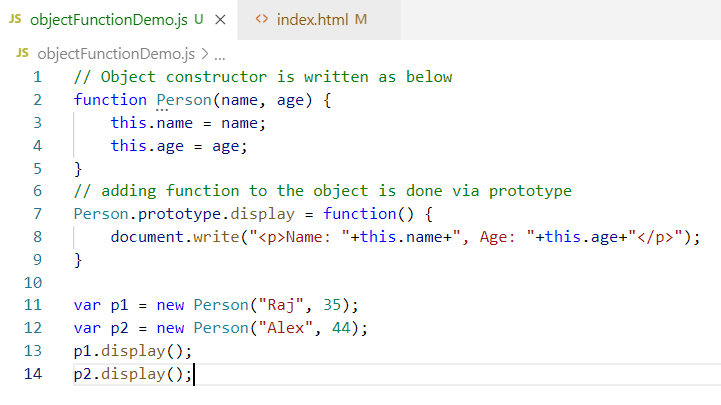
Day 2 agenda

* class, constructor, inheritance
* default parameters
* rest & spread operator
* static members
* generators
* string template literals
* object destructuring
* exponential operator
* trailing commas
* optional chain
* Object entries & values
* Arrays include

class in Javascript

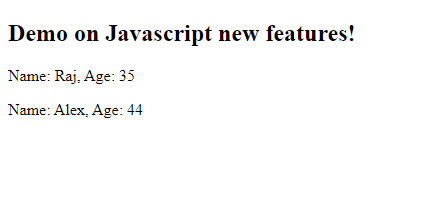
Before ES6 adding functions to the object was done using prototype and also object constructor were created using function keyword, but from ES6 onwards you can add functions in an object and constructors through class keyword.

Before ES6



Here the prototype lets you add new functionalities to the existing object and also you use function keyword to create the constructor which initializes object properties, you must use this keyword inorder to access object properties inside its function.

Output:

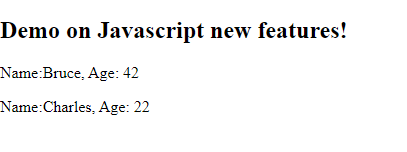


But from ES6 onwards you have a simpler syntax through class keyword you can create constructors & functions, so that you can encapsulate the functions that are part of the object

You need to use class & constructor keyword



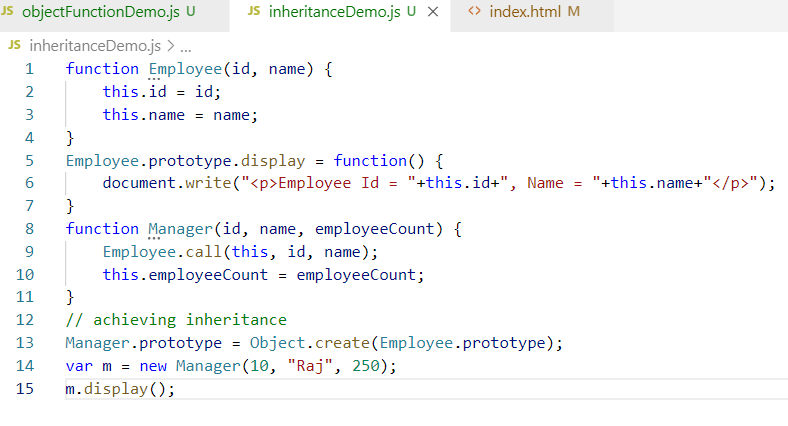
Output:



Inheritance in Javascript

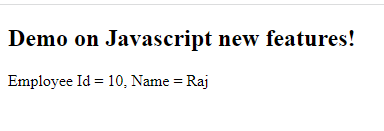
From ES6 onwards you have new keywords to achieve inheritance that is extends keyword and also you can call super class constructor using super keyword.

But before ES6 you need to use prototype to achieve inheritance



Earlier you need to use prototype to achieve inheritance

Output:

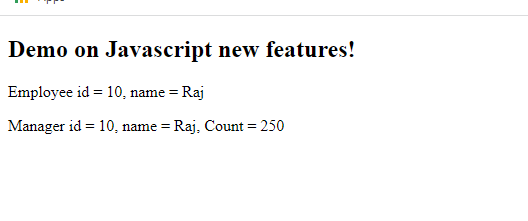


From ES6 onwards you have new keywords like extends & super using which you can achieve inheritance.



In ES6 the syntax to inherit and call the super class constructor is much simplified

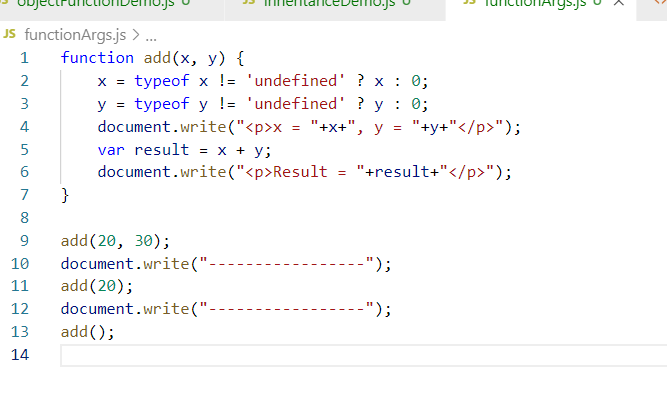
Output:



Default parameters to the function

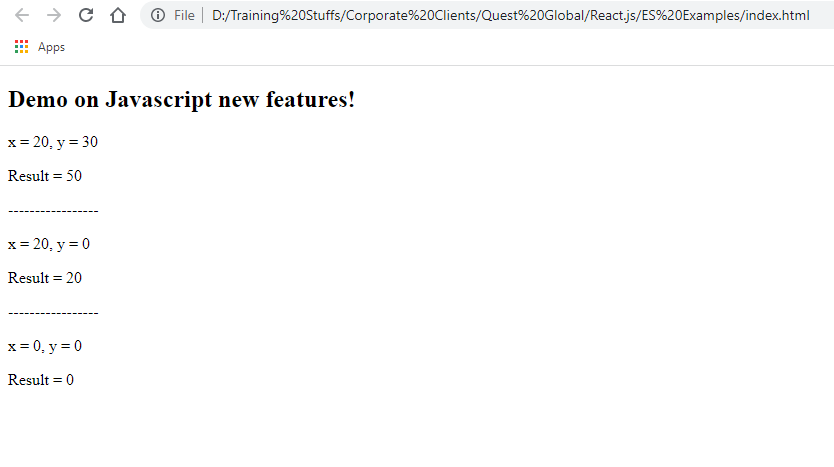
Since in Javascript you can skip passing right number of arguments when function is taking some parameters you need to use some default values when the right number of arguments are not supplied to the function, in that case you need to check for the type and assign default value.

Before ES6 developers used to check for undefined to assign the default value through some conditional constructs which is shown as below.



In the above code we are checking x & y has value or not, when x & y are undefined they get 0 by default else x & y considers value you supply to the add.

Output:



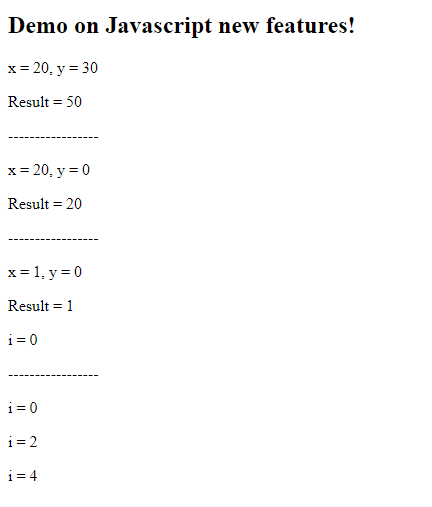
Note: If you don’t write the condition you will get x & y undefined & result will be NaN

The problem with the old approach is you need to check each parameter to assign the default value, if the number parameters are more you write more number of type checking code

Hence in ES6 they introduced default parameter function where you can assign default value to the parameter at the time defining the function



Output:

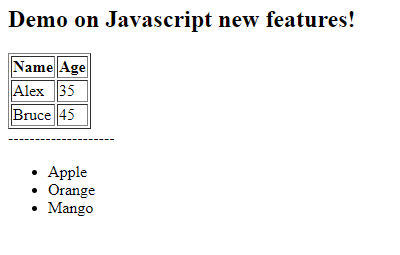


Template literals for Strings

It allows you to create strings & concatenate strings without + operator so that you don’t need to break the strings when you want to show some dynamic data, before ES6 you need to break the string to show the dynamic data and when your javascript is showing the data using HTML it is very difficult to write HTML as you need to break the strings to show the dynamic data as below:



Output:



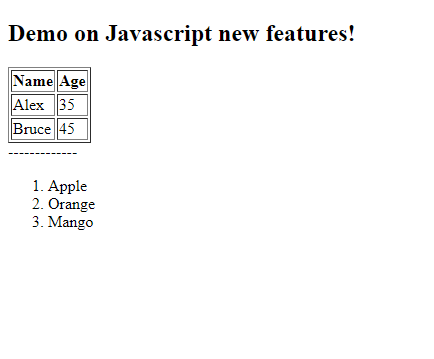
From ES6 onwards template string literals made easy to create strings with dynamic data without breaking it, it uses an expression with ${data} and string must be created using backtick `(below the ESC key)

Ex: `Hello ${name}` can be used instead of “Hello “+name

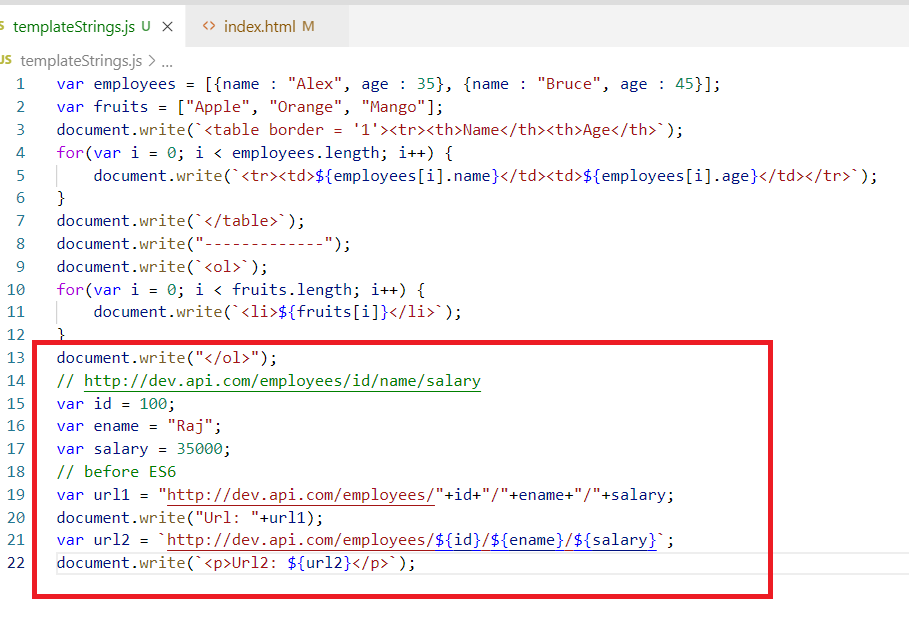
From ES6 onwards



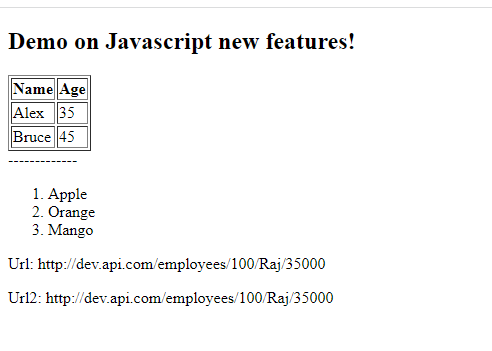
Output:



Another use of template string is you can concatenate multiple variables to prepare the webservice URL to call the back end service

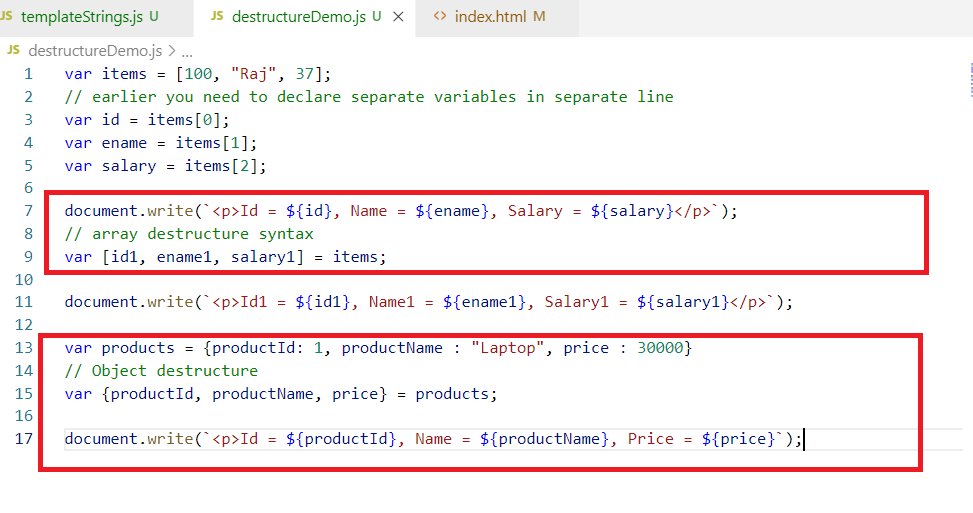


Output:

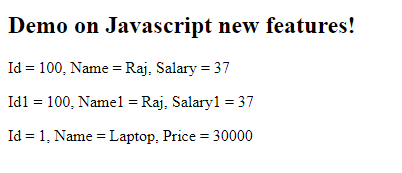


Array & Object Destructuring

It is one of the cool feature added in ES6 which allows you to unpack values from the arrays or object into distinct variables. you can extract data from arrays or object properties & assign them to the variables in a single statement



Output:



Rest & Spread operator

These are used to pass multiple inputs to the function parameters and not to loose any data

Rest: It is used in the function parameter which can accept 0 or more arguments

Ex:

function test(a, b) { } can accept maximum 2 arguments i.e., test(10, 20)

However if you pass test(10, 20, 30) then 30 will be lost because test accepts only two arguments, but in Rest operator you can avoid this data loss

i.e.,

test(a, b, …c){}

Here a & b are normal parameters but c is a rest parameter which accepts 0 or more arguments, but here in test you must pass atleast 2 arguments minimum and c is optional

you can call test by following ways

test(10, 20) : here a = 10, b = 10 & c = []

test(10, 20, 30): here a = 10, b = 20 & c = 30

test(10, 20, 30, 40): here a = 10, b = 20, c = [30, 40]

test(10): here a = 10, b = undefined, c = []

Note: You can have maximum only one rest parameter & also it must be the last parameter

test(a, b, …c){} is valid

test(…a,b,c){} is invalid

test(…a, …b, …c){ } is invalid.

test(…a) {} is valid

Spread operator:

It is used to distribute the inputs across multiple parameters of the function, if there is a function taking some arguments & you have an array then each items of the array you can spread to the function parameter

Ex: add(x, y){ ……. }

then items = [1, 3] can be spread to add by calling

add(…items), so here x = 1, y = 3

If items = [1, 4, 5] then

add(…items), assigns x = 1 & y = 4, but 5 would be passed so you can combine Rest operator mentioning in the function parameter

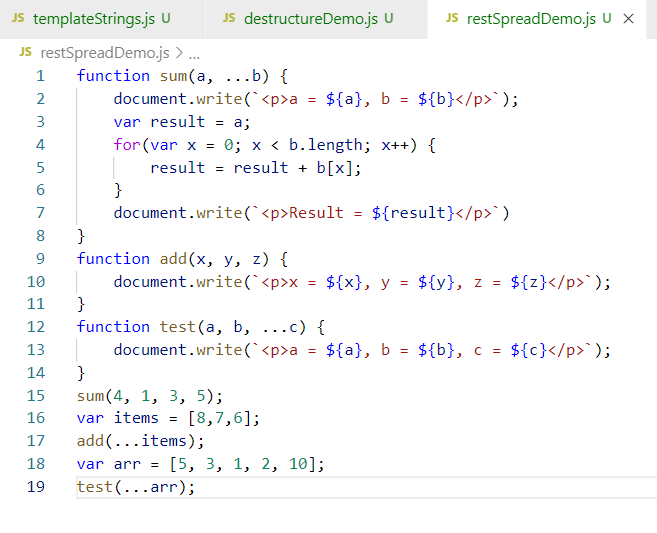
i.e., add(x, y, …z) { ……. }

Then if items = [4, 5, 1, 2, 3], calling add would be

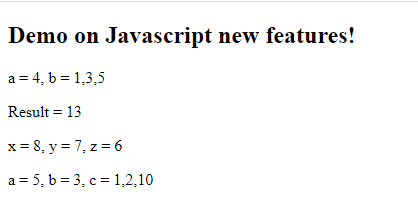
add(…items), where x = 4, y = 5, z = [1, 2, 3]

Note: Rest operator is written at the time function creation

Note: Spread operator is written at the time calling function



Output:



Generators

It is a special kind of function that can return multiple values one after the another on demand, return means yield as well in generators, in generators you can use yield keyword to return value & also you can use return keyword, but return must be the last execution statement, however yield can be anywhere in the function.

In generators you should call next() function to get the value from the yield, along with the value it will return done which will be true/false based on the status of generator completion, if function is completely executed then done will be true else done will be false.

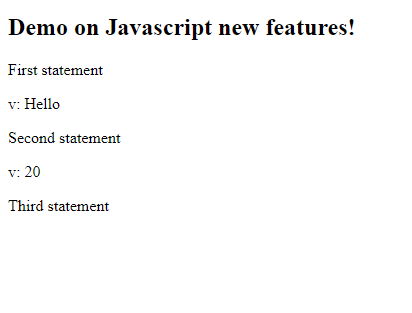
Generator would yield an object with two properties when you call next(), {value, done}, value will be the data the yield returns & done will be status like true/false.

Generators are created with \* beside the function or function name

i.e., function\* test() or function \*test()



Output:



Trailing commas

It removes extra commas if you keep in the array or function parameters without giving the error.

Earlier: items = [1, 2, 3, ] was an error

Now: items = [1, 2, 3, ] is fine, because javascript trails it



Output:



Trailing commas was a much needed feature, because most of the times developers tend to keep the extra comma because of some silly mistake either in React.js or Angular or Javascript, but now it will be removed if its extra

ex: In React.js if you create array of objects you may by mistake have extra comma as below

emp = [{name:”Alex”},{name:”Bruce”},]

ex: In Angular in the decorators you may have extra by some silly mistake

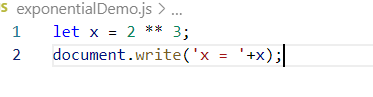
@NgModule({  
 providers : [..],   
})

Exponential Operator

It is used to write an arithmetic expression to find the powers in a simple way without using Math.pow()

Earlier: Math.pow(2, 3) returns 8

Now: 2 \*\* 3 returns 8



Output:



Event Handling in Javascript

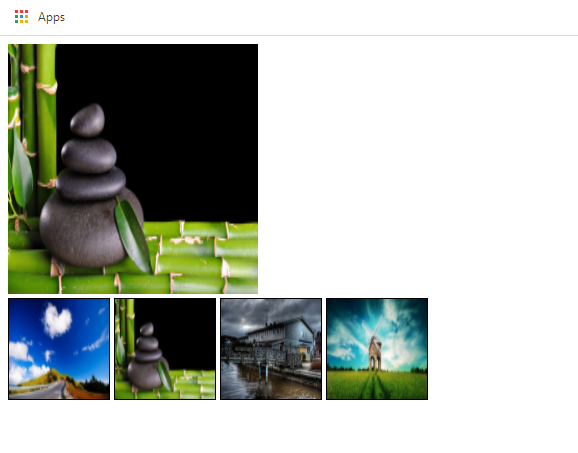
It is not part of ES6 it was there since the beginning, inorder to handle the users action we handle the events, these are generated by HTML elements

HTML elements generate variety of events like

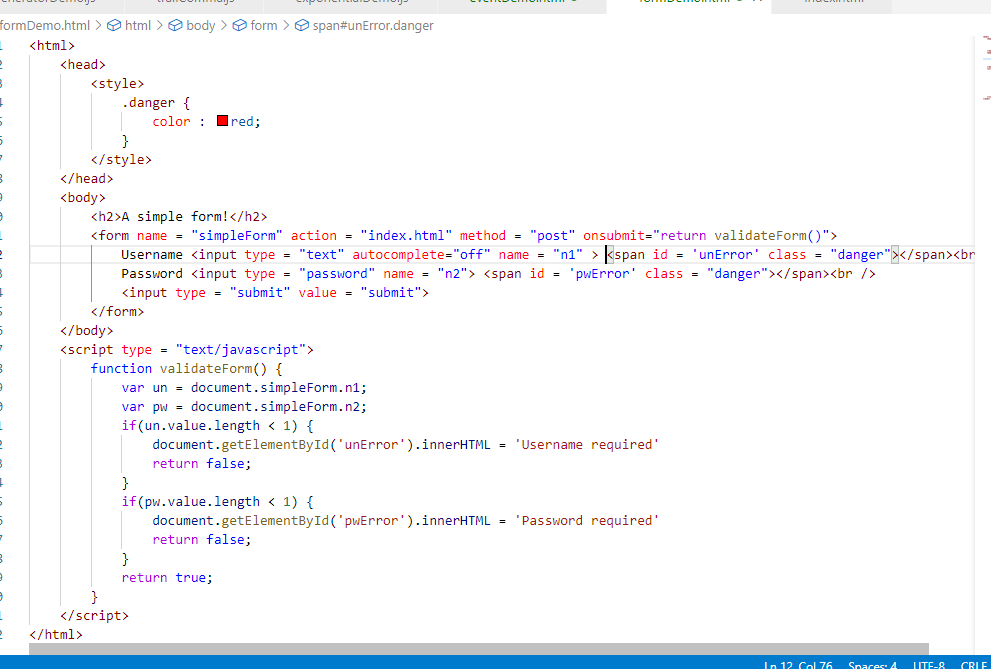
onclick, onblur, oninput, onsubmit, onmouseover, onmouseout, onsubmit and so on.



Here whenever you click on any image that is inside the div an event from that image is handled and its location is assigned to another image whose id is mainImage so that you can change the images dynamically.



Another useful feature of Javascript is form handling and validation



Output:



Activity

Create a form which will have 4 input elements

1. Firstname
2. Lastname
3. Password
4. Confirm Password

Validate each input so that firstname and lastname must have minimum 2 characters, whereas password & confirm password must have minimum 5 characters & their values must be same then only form must be submitted

Day 3 Agenda

* Object.entries() & Object.values()
* Static members
* Array.includes()
* padStart() & padEnd() functions
* Optional Chaining
* React.js

Object.entries() & Object.values()

These are new functionalities used to retrieve object properties & their values separately

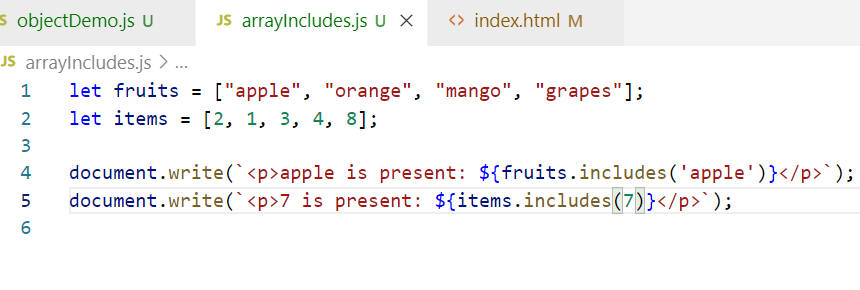


Output:

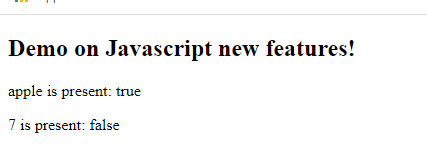


Array includes:

Earlier developers need to use index position to find the element of the array, but now you have includes() function to find the element of the array

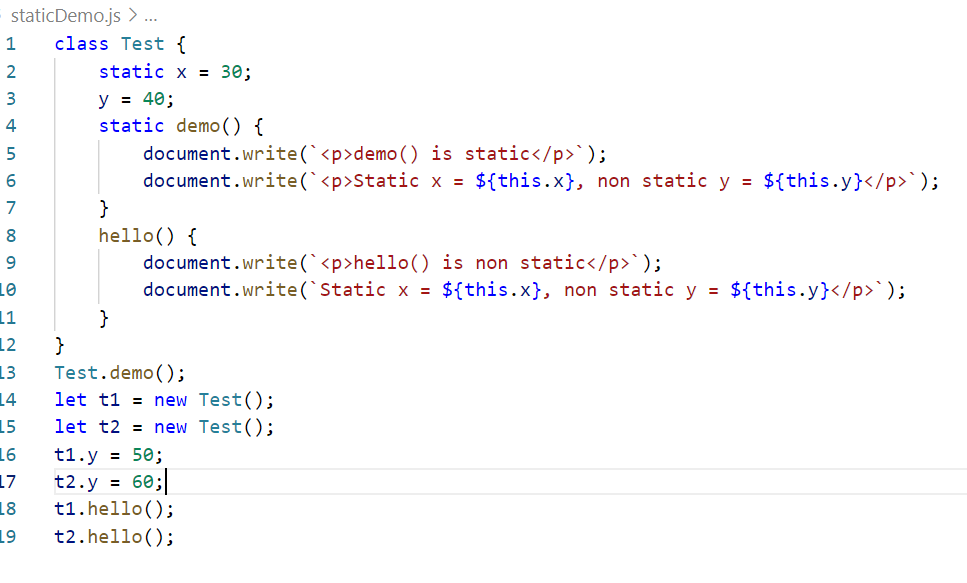


Output:

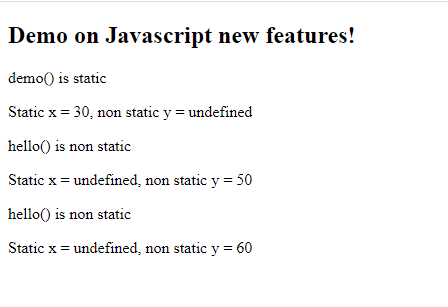


Static members:

It is used mainly to directly access the members with the class name, it is mainly used when you want fixed configuration for all the objects or to create utility functions like cloning objects and all.



Output:



New functions for String: padStart() & padEnd()

padStart(): It adds some string in the beginning of the string until the length of the padded string matches

ex: “9809”.padStart(6, ‘0’), here the 1st argument 6 is the length of the string and 2nd argument 0 is the string that should be added in the beginning of the “9809” if the length is less than 1st argument of padStart(), so the result would be 009809

ex: “44209”.padStart(6, ‘0’) >> 044209

ex: “887733”.padStart(6, ‘0’) >> 887733, because the current string length itself is string so no padding happens

padEnd(): It adds some string at the end of the string until the length of the padded string matches

ex: “998882”.padEnd(10, ‘x’) >> 998882xxxx

ex: “9988223344”.padEnd(10, ‘x’) >> 9988223344, as the current string length and the amount of strings to pad is same it doesn’t pad.

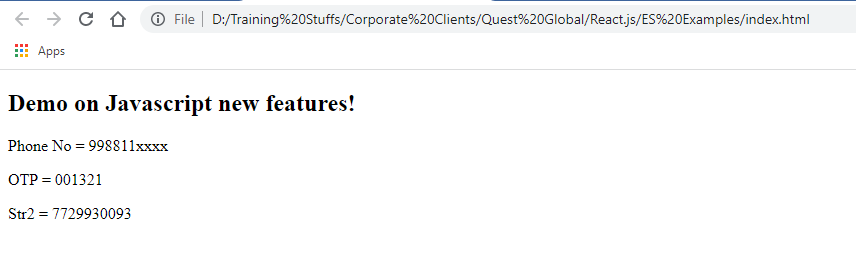
padStart() & padEnd() doesn’t pad the extra strings if the length of the string is greather or equal to the 1st argument

Note: You can apply padStart() & padEnd() only to the strings not to the numbers, if they are numbers you need to convert them to string first

padDemo.js



Output:



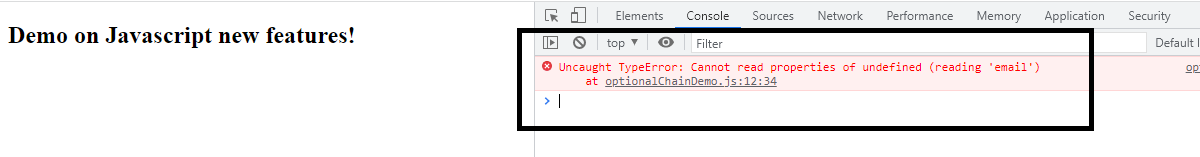
Optional Chaining(?.)

It is used to safely access the nested properties of an object to avoid errors in the javascript if in case the nested properties are not present.

Whenever there is an object with lot of intermediate properties developers tend to get errors when they access some properties which are missing in some objects to avoid that developers used to write lot of conditions to check for its existence

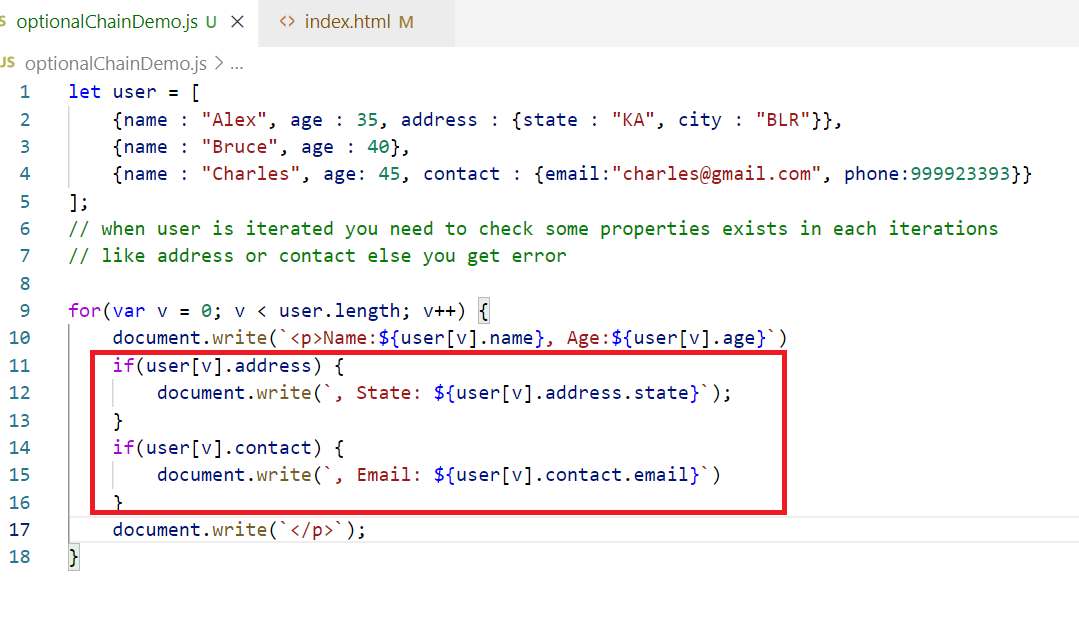


The above code gives error when iterating the item because each object iterating doesn’t have address or contact which are intermediate properties, they will be undefined & accessing their nested properties would raise the error as below:

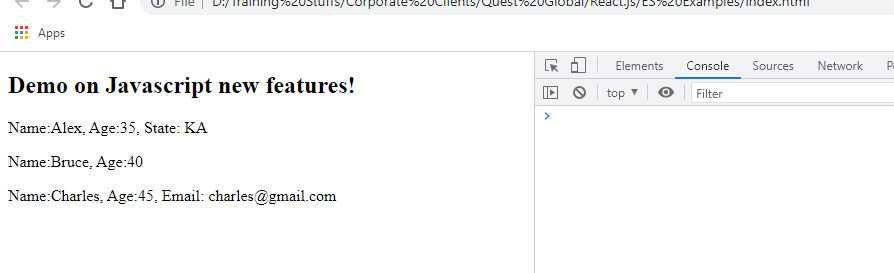


On the first iteration itself the object contact is undefined & accessing email on undefined raises error.

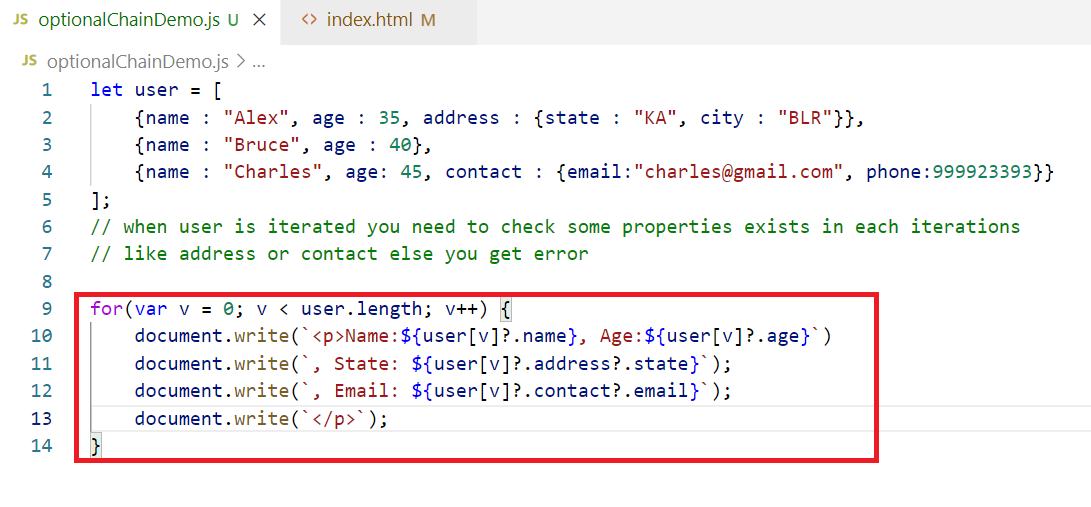
To avoid that you need to write lot of conditions to check undefined & then access



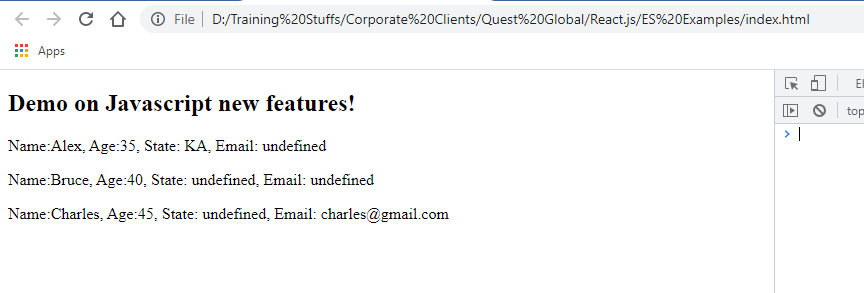
Output:



The above code may not throw error, but your application logic will be big to only check for undefined in the object, however from ES2020 onwards optional chain would allow you to safely access object properties or functions by internally checking if they exist or not using `?.`, Here `?.` means if exist then access.



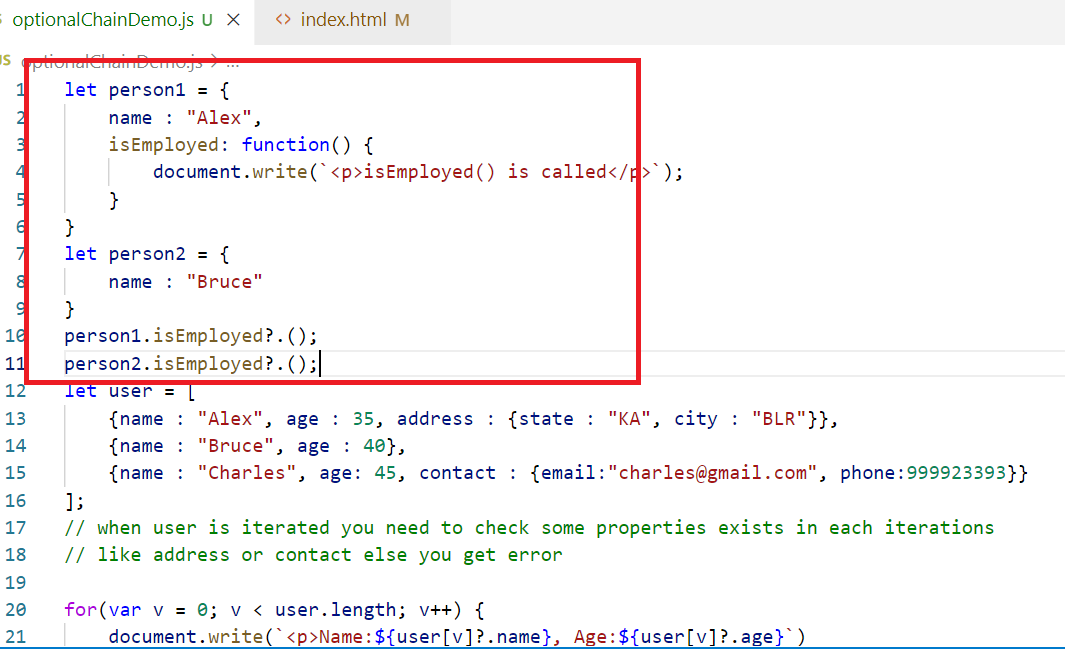
The above code stops accessing the intermediate property if it finds property doesn’t exist, i.e., if address is undefined, then it doesn’t access state from the code ‘user[v]?.address?.state’

Output  


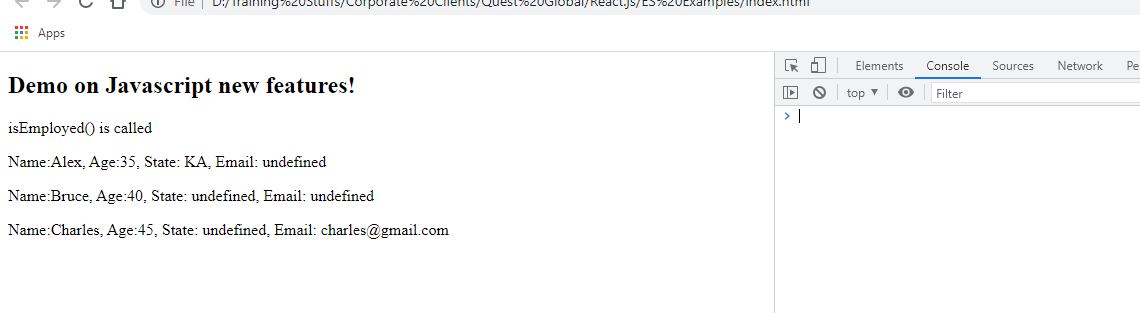
Note:

The ?. immediately stops the evaluation as soon as the property is not present which is like short-circuit, so if there are any further calls or side effects then they don’t occur.

You can also use the ?. on the functions



Output:



Summary:

* Keywords like let, const, class, super, extends
* Arrow function
* Template String literals
* Rest & Spread operators
* Generators
* Object & Array Destructuring
* Default values to the parameters
* Trailing commas
* Exponential operator
* padStart & padEnd
* Array.includes()
* Object.entries() & Object.values()
* Static

Activity

1. Try all the above examples
2. Using padStart() & padEnd() try the following requirements
   1. If the phone number is 10 digits split that to 6 digits and 4 digits and for the 6 digits you pad ‘x’ at the end, suppose if phone number is 9988445522, then divide it to get 998844 and add ‘x’ at the end so that result must be 998844xxxx
   2. Use Math.random() and generate an OTP of 6 digits, ensure you will add 0 at the beginning if the OTP generated is not 6 digits, suppose Math.random() returns 0.0235293 then converting it into 6 digits will give you 5 digits i.e., 23529, not 023529, hence you need to pad 0 in the beginning