Javascript & Node.js

Front-end: Browser to run

Back-end: Node.js - File access, database operations,

Fundamentals

* variables
* functions
* loops
* conditional
* arrays
* objects
* classes
* new features
* DOM manipulations
* Form validations

Front-end - Browser to run the javascript & we need some editor VSC

Operators:

++, --, \*, %, =, ==, ===, !=, <=, >=, ?:

var counter1 = 15;

var counter2 = 15;

var x = ++counter1; // x = 16, counter1 = 16

var y = counter2++; // x = 15, counter2 = 16

Functions:

These are named blocks which can have one or more statements, they can accept arguments which caller can pass as input

function function\_name(arg1, arg2) {   
 …   
}

function test1(x) {   
 // console statement  
}

test1(20); // fine  
test1( ); // fine, x will be undefined

In Javascript even if you don’t pass arguments the function will be executed

Conditional Statements in Javascript

if(condition) : You can have a boolean condition

if(a > 5) { … }

if( a > 5 || b > 5) { … }

if(a > 5 && b > 5) { .. }

if & else: You can have any one to be executed.

if(a > b) { … }

else { … }

if, else if, … else if…. else

if(marks >= 80) { …. }

else if(marks >=70 && marks < 80) {… }

else if (marks >= 60 && marks < 70) { … }  
else { …. }

switch(…): it is similar to if else if … else, but switch takes cases

switch(expression) {   
 case caseNumber [or] caseStrings: …. break;  
 … // if all the cases fail then default will be executed  
 default:   
}

num = 2;

switch(num) {   
 case 1 : … break;  
 case 2 : … break;  
 case 3: … break;

default : ….  
}

Difference between == & ===

== comparison operator it compares values but not types

=== comparison operation it is called exactly equal to operator, it compares values & types both

Javascript has below types

1. number
2. string
3. boolean
4. Date
5. Object

typeof function you can use to know the datatype of value/variable.

typeof(10): number

typeof(“10”): string

var x = 10; typeof(x) : number

var y = “10”; typeof(y): string

var z = 10;

x == y // true, checks only value but not the type

x === y // false; checks value & type both

x === z // true; as value & type are same

% operator:

It gives the reminder of 2 numbers you divide

3 % 3 will be 0

3 % 2 will be 1, because 3/2

2 % 3 will be 2, as 2 can’t be divided by 3

// taking input in javascript

There are following ways to take input

1. prompt() function
2. form element
3. terminal : using some keyboard related libraries

prompt(‘enter some value’): returns the value you enter and stores in a variable

Note: The value you enter will be in string format by default

x = prompt(“Enter x”)

y = prompt(“Enter y”);

x = 15000

y = 5000;

Result must be 20000

Loops:

To repeatedly execute the statements we can use the loops, the loop will stop when the condition becomes false

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

Arrays:

It is used to store multiple elements in a single variable

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

Activity:

1. find the sum of numbers in the array and print the result
2. find the maximum & minimum number in the array & print their values.
3. enter a number from the prompt & print if that number is present in the array

Ternary operator:

?: It is a simple form writing if - else

a = 20;

b = 30;

c = (a > b) ? a : b

// value of c will be 30

document.write((a > b) ? a : b); // prints 30

Objects:

Objects are real world entities which will have properties & functions

you can create object using function constructor

function User(name, age) {   
 this.name = name;  
 this.age = age;  
}

var u = new User(‘Raj’, 35);  
 u.name & u.age to get the properties

prototype: It can be used to add some behaviours to the object like adding functions to the object

ObjectName.prototype.functionName = function(arg1, arg2) {   
…  
}

DOM: Document Object Model, these are the objects that are created at runtime from the HTML elements

Using DOM you can add dynamic behaviours to your web page

canvas: it is an element used to create 2D drawings using Javascript

You must access the element and call getContext(‘2d’) on the element object

Day 2 Agenda

Form Validation

XMLHttpRequest

Callbacks

New features of Javascript

class, super, extends, let, const

Arrow functions

Rest & Spread operators

De-structuring

Promises

Async & Await

Callback functions:

These are the functions which are initiated but executed later based on some events like response from the server, timeout, database connections and so on.

Callbacks are mainly used to perform asynchronous operations

XMLHttpRequest object uses callbacks to process the response coming from the server.

New Features added in Javascript

* New keywords like let, const, class, super, extends,
* Template Strings
* Arrow functions
* Rest & Spread operators
* Optional Chain
* Promises
* async & await

let & const: They are used to create block scoped variables

Arrow functions:

Simpler form writing anonymous function or callbacks

Callback:

function(x, y) {  
 return x \* y;  
}

Arrow function

(x, y) => { return x \* y }

[or]

(x, y) => x \* y;

xhr.onreadystatechange = () => {   
   
}

There are callbacks that are passed as an arguments in many functions you can replace them by arrow functions

Inheritance in Javascript

class A { }

class B extends A { }

Day 3 Agenda

Other new features of Javascript

Node.js

Variables & Functions Hoisting

Hoisting is a process where javascript keeps the functions, classes, variables at the top-level of the script,

Note: Javascript keeps only the declarations at the top not the initialization

Note: let & const wouldn’t be hoisted as they are block scoped, however var, class, functions will be hoisted at the top

Function hoisting

test(); // works fine without any error

function test() { …. }

Variables hoisting

console.log(x); // works but prints undefined

var x = 10;

let & const hoisting: Doesn’t work

console.log(y); // error

let y = 30;

function can be assigned as an expression

abc(); // doesn’t work, because only variable will be declared

var abc = function() { ….. }

abc(); // works

Rest & Spread

Rest parameter is used when you want a function to accept 0 or more arguments

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

function sum(…a) { } // valid

function sum(a, …b) { } // valid

function sum(…a, b) { } // invalid

function sum(…a, …b) { } // invalid

Rest parameter accepts values in the form array so you can use all the array methods on the rest parameter variable like forEach, map, filter and so on.

Spread operator

It is used to distribute the array items to multiple arguments of the function

it is used at the time of calling the function or assigning the values

// rest parameter

function test(a, ...b) {

}

function demo(a, b, c) {

}

items = [20, 30, 40]

// spread parameter

test(...items); a = 20, b = [30, 40]

demo(...items); a = 20, b = 30, c = 40;

items = [1, 3, 2, 4, 5, 7, 8]

demo(...items); a = 1, b = 3, c = 2, the other values like 4, 5, 7, 8 will be lost

test(...items); a = 1, b = [3, 2, 4, 5, 7, 8]

Similar concept of spread can be applied at the time of value assignment which is called as destructuring

Earlier

let items = [5, 7, 8];

let x = items[0];

let y = items[1];

let z = items[2];

New feature

let [x, y, z] = […items];

Optional Chain (?.): It is used to access the property on conditional basis

Earlier:

user = {id:100, name: “Raj”, address: {state:”KA”, city:”BLR”} }

user.address.state to access any nested properties, but sometimes this leads to undefined errors

because, if you are accessing array of these objects in a loop you may need to check address is present or not

i.e., users = [ {}]

New methods in string object

padStart & padEnd functions that helps you to add some extra characters in the beginning or in the end of a string/numbers in string format.

abcxxx@gmail.com

988983xxxx

xxxx998839

456900 or 004569

“4569”.padEnd(6, ‘0’); it generates 6 digits from the number by adding 2 0’s to the end

“4569”.padStart(6, ‘0’): it adds 2 0’s in the beginning

“45678”.padEnd(6, ‘0’); it adds 0 to the end = 456780

“234567”.padEnd(6, ‘0’); nothing will be added as the number of digits itself 6.

Using undefined & null

undefined means a variable is not present or a property is not present, it is a type as well in javascript.

null is a value you can assign to a variable mentioning its empty.

obj = { a : 1 }

if(obj.b == undefined) {   
 // you can define b property

obj = { a: obj.a, b : 1 }  
}

obj2 = {a : 1, b : null }

// this below condition will be false as be is defined but its value is null

if(obj.b == undefined) { // condition will be false  
  
}

if(obj.b == null) { // condition will be true   
  
}

a = null;

typeof(a) will give object

b = undefined;

typeof(b) will give undefined

obj = { a : 10, b : undefined }

if(obj.b == undefined) { }

Closures: It is a function nested inside another function, where you can access it or call it only within the enclosed function

Note: inner function can access all the data of outside function, however inner function data wouldn’t be accessible to the outer function

function abc() {   
 function xyz() {   
 return value;  
 }   
 let a = xyz(); // this is fine  
 return a;  
}

xyz(); // doesn’t work

Promises:

It is an object that performs asynchronous operation based on the state of the promise like rejected state or success state

to handle success state the promise uses then( callback ) function

to handle rejected state the promise uses catch ( callback ) function

fetch(“ url “) returns a Promise through .then ( callback ) or .catch ( callback ) we can find out the fulfilment of the promise object

Promise was introduced to handle interdependent asynchronous operation easily

// nested callbacks

function(result, error) {   
 if(result) {   
 function(result2) {   
 ….  
 }  
 } else {   
 function(error2) {   
  
 }  
 }  
}

fetch(url)

.then( function() { … } )

.catch( function () { … } )

.then ( function () { } )

.catch (function() { } )

.then( function() { })

.catch( function() { } )

Note: fetch() function is inbuilt in browser, you can use in browser without any library

Earlier we used XMLHttpRequest to make an asynchronous request

xhr = new XMLHttpRequest();

xhr.open(“get”, url);

xhr.send();

xhr.onreadystatechange = ( ) => { …. }

Using fetch is much easier

fetch(url, { method: “POST”, body : { json data} } ): Returns Promise object

i.e.,

fetch(“http://api.com”).then( (data) => { … }).catch( (data) => { … }))

HTTP status codes :

2XX >> success codes

4XX >> failure codes

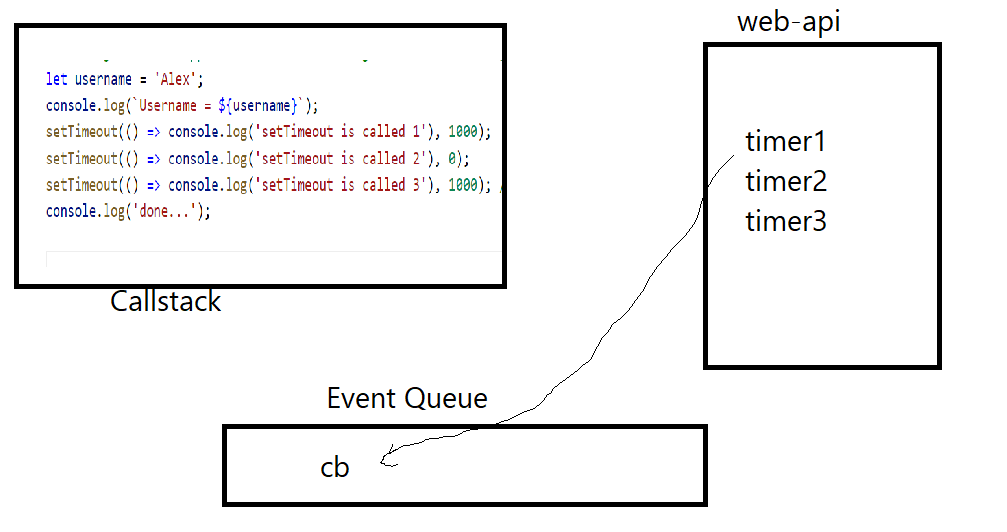
Node.js

It is a runtime environment to run the javascript at the backend so that you can perform many backend operations like file handling, database operations.

Node.js architecture

It uses event loop mechanism to run the javascript & doesn’t block any IO operations that takes time

node filename.js >> this runs the javascript file



Node.js modules

Modules are some reusable unit of code like functions, classes, variables

1. Core modules - os, fs,
2. Local modules - which we create in our project
3. Third party modules - which are downloaded from the internet like express, cors, readline-sync, …

How to use these modules

You need to import either by using require (old approach) or using import (ES6 approach)

let os = require(“os”); // old approach

import os from ‘os’; // es6 approach, but you need to use module type in package.json

package.json:

It is a main configuration file that will have project metadata

By default node.js uses old approach to import the modules, but if you want to use es6 approach then in package.json you need to mention

{   
 “type” : ”module”  
}

If you use above lines in package.json you can import & export modules with ES6 feature, else you need to import & export with old approach

os module: It gives you properties & functions that helps you to give os related information like platform, hostname, architecture and so on

Sometimes your application may want to create some resources which OS dependent then your application must have an idea about the OS in that case os module will be useful.

Mac -> a.sh

Windows -> a.bat

let os = require(‘os’); // os module is core module

os.platform()

os.architecture()

Third party module we need to install using npm install or npm i module-name

ex: we want to take input from the keyboard, we have readline-sync module

npm install readline-sync

or

npm install readline-sync express cors body-parser

The above command downloads totally 4 modules

import read from ‘readline-sync’;

read.question(“Enter some text: “): to read strings

read.questionInt(“Enter some number: “): to read integers

read.questionPassword(“Enter some character: “): to read masked characters & advantage of this function is it expects a strong password & also expects you to enter password again to confirm

Understanding the imports

import someName from ‘module’; // default import

import { someName } from ‘module’; // named import

import a, { b, c, d } from ‘module’; // a is default & b, c, d are named imports

Implementing some data-structures

Datastructures are a way to manage the data, it must have functions to manipulate the data

1. Stack : Last In First Out
2. Queue: First In First Out
3. Hashtable: Key value pairs, ex: object is a kind of key value pairs datastructure

REPL command line in node.js

Read, Evaluate, Print & Loop it’s an environment given by node.js to test simple javascript expressions

\*\* Implement a Queue that adds and removes in the following way

1. add(): Adds elements in to the array
2. poll(): Removes the top element & returns it
3. peek(): Returns the top element, but doesn’t remove

ex: queue = [3, 2, 5, 4, 8, 6], poll() : must remove 3

\*\* Implement a PriorityQueue that adds & removes in the following way

1. add() : Adds elements into the array
2. poll(): Removes the elements in sorted order

ex: queue = [3, 2, 5, 4, 8, 6], poll() : must remove 2, once again poll() is used means it must remove 3, then 4, then 5, then 6 & lastly 8

Map: it used to store data in key value pairs

map keys will be unique

map.set(key, value);

fs module

fs: file system module used to read/write files, it is part of core module.

it provides functions like

readFileSync()

writeFileSync()

to perform read & write

writeFileSync(‘abc.txt’, data, { flag : ‘a+’});

readFileSync(‘abc.txt’) : returns the data in binary format which needs to be converted to string to see the text representation

let data = fs.readFileSync(“simple.txt”);  
console.log(data)

let str = data.toString();

console.log(str)

Reading a string from the file & converting it in the form object

npm init -y

The above command creates packages.json with default values

Writing a reusable program which can write new javascript object to the JSON file without erasing old data.

Webservices:

These are online services available over the internet that can be accessed by multiple client applications written in different languages

ReSTful webservice: ReST stands for Representational State Transfer,

Rules while designing REST webservices

1. You must have URL’s for each webservice
2. There must be any one HTTP method mapped to the webservice
3. When producing the data send the status codes also along with the response like 200, 404 and etc

Rules while consuming REST webservices (Clients)

1. Use the HTTP clients to access the webservice ex: fetch/XMLHttpRequest in Javascript
2. Use the URL & Http methods as per the standard rule of HTTP like if client wants to fetch the data it must use GET method of HTTP
3. Make use the HTTP status code to understand the type of response whether error or success response

You can create webservice & consume webservice in any technology

Node.js - Express module

Java - Spring REST

C# - ASP.NET

Express module is used to create webservices in node.js

#installing express

npm install express

let app = express();

app.listen(9090, callback);

app.get(url, callback);

app.post(url, callback);

app.delete(url, callback);

app.put(url, callback);

callback of get, post, put & delete will have 2 arguments

1. request
2. response

i.e.,

app.get(“/test”, (request, response) => { … } );

app.post(“/test”, (request, response) => { … });

app.put(“/test”, (request, response) => { .. });

app.delete(“/test”, (request, response) => { … } );

POST: /employees  
Body: { … }

webservice

app.use(parser.json()); // converts JSON to Javascript

app.post(“/employee”, (request, response) {   
 let emp = request.body; // gets the Javascript object   
} );

Path parameters: It is used to parameterize the paths so that it can accept dynamic values

/employees/:user

The above path can accept value to the key :user

i.e., /employees/Sachin

/employees/Virat

cosole.log(‘….’);

console.error(‘….’);

console.warn(‘…’);