Date 22/09/2022

Mo Tu We Th Fr Sa Su

JavaScript Concepts:

1. JavaScript Introduction:

Client ——→ Search Engine ——→ Web Server

* Client request :- Google.com, Amazone.com
* Server response :- response contains the HTML + CSS + Javascript files.

Javascript Usages:

i) Javascript is a high-level programming language.
[Binary language is understood computer]

ii) Evergreen programming language as used in front-end & back-end.

iii) Event based :- hover, click, load, etc.

iv) Brain of the web page.
HTML → Is a skelton body, EMPTY Body
CSS → Is a color, size, position,
Javascript → Brain, It will suggest function.

Date

Mo Tu We Th Fr Sa Su

Javascript, where it runs:

* Every browser has a Javascript engine [It converts Javascript into Binary code].
* Engines:-
  - Firefox → Spider Monkey.
  - Chrome → Chrome-V8.

* Javascript code can be run inside the browser or in node.

* Node:

  In 2009 developers took Javascript in Chrome embeded it inside a C++ program.

Date

Mo Tu We Th Fr Sa Su

Console concepts:

1. console.log ("Hi Lucky"):
   = It will display message on console.

2. console.clear():
   = It will clear the data from the DOM.

3. console.warn("Hi created warning msg").
   = It will created warning message.

4. console.error("This is error msg");
   = Display error message.

5. console.assert(0>2, "It is not possible");
   = Assertion failed = msg.
   Basically like some of false (If-else)

6. console.table([1,2,3,4]):
   = Display table format

(index)	value
0	1
1	2
2	3
3	4

7. console.time("code"):
   }
   console.timeEnd("code");

} It display's the time for taking execution.

Mo Tu We Th Fr Sa Su

Date

Variables:-

* It is a storage or container
* For declaring variables, Following rules:

we can declare	we can't declare
var num;	var 1num
" $num;	"
" $1num;	
" _num;	

* The num & Num both are different variable.

Types of variable:

1) var:-
* global variable.
* not a scope of variable.
* we can use entire file.

2) let:-
* scope of variable ({ --- })

3) Const:-
* we can't modified value of variable.
* once we declared as const variable, we need to initialize.

Date

Mo Tu We Th Fr Sa Su

Ex:
```
var num = 10;
console.log(num) // 10
{
  let num = 20;
  console.log(num)  // 20;
} var num = 50;

console.log(num)  // 50;
```

Ex:
```
const name = "Lucky";
console.log(name);  // Lucky;
name = "Appu";      // Error.
console.log(name);
```

Difference b/w var, let, const

var — Hoisting also works

* global scope, - we can use outside the block & inside block with reinitialization, & redeclare with same variable.

let

* Block scoped :- we cannot use outside block & not redeclaration

Date

Mo Tu We Th Fr Sa Su

Data-Types

1) Primitive Data Types
   i) Numbers
   ii) String
   iii) Boolean
   iv) NULL
   v) Undefined
   vi) Symbol.

ii) non-primitive D.T
   i) Array
   ii) object.

→ var num;
console.log(num); // undefined

* There is a method to check the data type.
= typeof (variable_name);

i) var age = 20;
console.log(typeof age); // numbers.

ii) var name = "Hi Lucky";
console.log(name); // Hi Lucky;

we can use ' or " quote and \ If we use third (`) symbol, we can easy to add ' or " in value.

iii) var bool = true;

iv) NULL is a P.T but consider as object.
It means holding as a buffer value.

Date

Mo Tu We Th Fr Sa Su

Conditions:

i) If
ii) If-Else,
iii) nested If.
iv) Switch

Loop's:

① While Loop:-

* It's display repeatedly

Syntax:

```
while (condition)
{
  condition(ison/deen);
}
```

Ex:-
```
let i=1;
while (i<=10)
{
  console.log(i);
  i++;
}
```

O/P

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Date

② do-while:

At least execute once, if condition is true fall.

```
let i=10;
do
{
  console.log(i);
  i++;
}
while (i<=10);
```

O/P

11

③ for loop.

~~let i=10;~~

```
for (let i=0; i<=10; i++)
{
  console.log(i);
}
```

Date

Mo Tu We Th Fr Sa Su

Storing Methods:

① charAt(): ⇒ returns index value.

② charCodeAt():
returns ASCII value.

③ concat():
returns add multiple strings

④ endsWith():
check ends value is there or not and returns boolean value.

⑤ fromCharCode():
returns character.

⑥ replace(): includes():
check whether a storing contains the specified storing / character
str = "Hello Hi"
str.include('ll'):
true.

⑦ indexOf():-
Returns the first occurence of a specified value in a storing.

⑧ lastIndexOf():-
Returns the last occurance.

Date

Mo Tu We Th Fr Sa Su

(8) Match:-
Search a string for a match & returns the matches.
Ex- str = "Hi Welcome Hi Hello"
str.match(/Hi/gi) // Hi Hi

(9) repeat():-
returns with a no. of copies of specified string
Ex str="Kamada"
str.repeat(2) // kamadakamada

(10) replace():-
Returns a new string where the specified values are replaced
Ex let str = "Java course";
let s = str.replace("Java", "Python");

(11) search():-
Returns the position of the match
Ex let str = "Hi search engine"
let s = str.search(/search/gi); // 4

(12) slice():-
extracts a part of a string & returns a new string
Ex Let str = "HiHello")
let str1 = str.slice(3, 5) // He

(13) split():-
splits a string into an array of substrings.
= Array formate

Date

Mo Tu We Th Fr Sa Su

(14) startsWith():-

let str1 = "Javascript Course"

let s = str1.startsWith("Co", 11);

∴ checks whether strings are started with same position.

And returns boolean value.

(15) toLocaleLowerCase():- & toLocaleLowercase()

convert into lower case().

str1.toLocaleLowerCase();

(16) toUpperCase():- & toLocaleUppercase()

convert to Uppercase all letters.

(17) toString():

convert to string

Ex let s = 45627;

let str1 = s.toString(); // 45627 string format.

(18) trim():

Trim the begin & ending spaces.

let str1 = " Javascript ";

let s = str1.trim(); // Javascript.

(19) valueOf():-

Returns the primitive value

str.slice(0)

Date

Mo Tu We Th Fr Sa Su

(20) substring():

* start = = end index. empty string.
* end index > str.length consider as string length as end index.
* If the end index is not given, so it's consider as ~~length~~ print the whole string.
* If the start > end index. so returns the swap the arguments.
* If we give ~~EMPTY~~ or -ve NAN consider as 0.

slice():

* start = = end index empty string.
* same as 2nd point
* same as 3rd.
* Returns the empty
* Returns from the end.

(21) substr():

str = Lucky.
str.substr('1', '2') // UC.

It will take a part of the string from starting index to length of given -string.

Date | | | Mo Tu We Th Fr Sa Su

Arrays

* Array is a store multiple values in a single variable.
  arr = [S1, S2, S3];
* If you access array values using index.

Array methods:

arr = [2, 4, 6, 1, 3];

① Join(): Any can be = separator. Eg Array can't be changed
arr = [2, 4, 6, 1, 3];
* reduces the array as a String.

② push():
At the end insert the element
~~arr = [2, 4, 6];~~
arr.push(5); // 2, 4, 6, 1, 3, 5

③ pop():
At the ~~end~~ remove the element.
arr.pop(); // 2, 4, 6, 1, 3

④ shift():
It's a opposite to the pop(); arr.shift();
At the first remove the element. o/p 4, 6, 1, 3, 5

⑤ unshift();
① At the first place to add the value.
② opposite to the push
arr.unshift(7); 7, 2, 6, 1, 3

Date | | | | arr = [1, 2, 3, 4, 5]

Mo Tu We Th Fr Sa Su

⑤ splice():
We can add any element on the any position in the array.

Ex arr.splice(2, 0, 3);

index need to add value to the position remove array.

⑥ concat():
adding multiple array's into to new array.

let arr = [1, 2, 3];
let arr1 = [4, 5];
let res = arr.concat(arr1); // 1, 2, 3, 4, 5

⑦ slice():
We can get elements from given index to up to end given index.

Ex arr.slice(1, 3);

o/p
2, 3

⑧ map():
create a new array, by performing operation each element of a given array.

arr.map((item) =>
{
console.log(item);
});

Date | | | | | | | | Mo Tu We Th Fr Sa Su

⑨ filter():-

* create a new array, with array elements, which passes a test.
* Based on the condition, if condition is satisfied it will return.

Eg: let num = [37, 29, 8, 9, 45];

```
num.filter((item) =>
{
  return item > 29;
})
```

O/P
37, 45.

⑩ some():

* Returns boolean value.
* If the condition get true ⇒ false.
* working same as filter.

```
num.sum((item) =>
{
  return item > 29;
})
```

O/P
true.

⑪ reduce():-

* reduce is a function and it will return a single value.

```
num.reduce((sum, item) =>
{
  return sum + item;
})
```

O/P
Total sum

Date

Mo Tu We Th Fr Sa Su

(12) every():

* Returns true or false, If the all elements gets the true the result is true.
* Every elements get condition is satisfied.

```
arr = [2,4,6]
arr.every((item) =>
{
  return item % 2 == 0;
})
```

o/p
true.

(13) find():

* Returns first array element that passes the condition.

Ex:

```
arr = [2,4,6,8];
arr.find((i) =>
{
  return i > 5;
})
```

o/p
6.

(14) findIndex().

* Returns index position.
* first element.

```
arr = [2,4,6];
arr.findIndex((i) =>
{
  return i > 4;
})
```

o/p
2.

Date

Mo Tu We Th Fr Sa Su

Functions:

* It's used to for reusable code.
* Is a block of code designed to perform a particular task.

```
function myFunction(a, b)
{
    return a * b;
}

myFunction(10, 20);
```

o/p
200

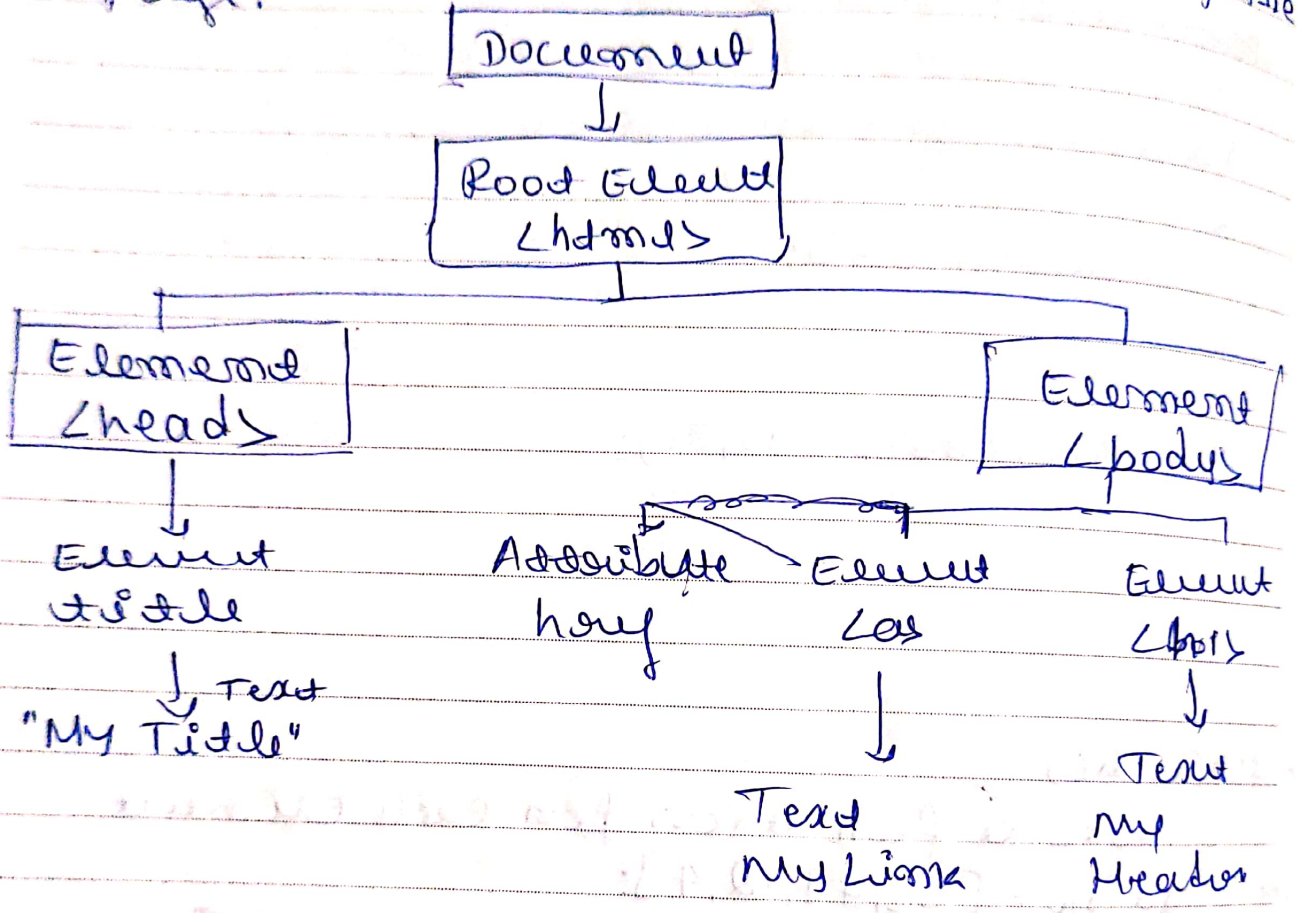
* forEach()
  calls a function for each element.

Eg
```
const no = [65, 44, 12, 4];
no.forEach(function(value, index, arr)
{
    arr[index] = value * 10;
});
console.log(no); // 650, 440, 120, 40;
```

Date

DOM :- Document Object Model

* When a web page is loaded, the browser creates a Document Object Model of the page.



* DOM Methods:

① document.getElementById(id);
Find an element by element id.

② document.getElementsByTagName(name)

③ document.getElementsByClassName(name);

Date

Mo Tu We Th Fr Sa Su

* querySelector():

* returns the first element that matches a specified CSS selector(s) in the document.

Ex
document.querySelector(CSS selectors)

* querySelectorAll():
To return all the matches, use the querySelectorAll():

Ex
document.querySelectorAll(CSS selectors)

Mo Tu We Th Fr Sa Su

Date

Timing Events:

① **setTimeout():**
executes a block of code after a spec. -fied time(only once).

Ex: function helloWorld()
{
console.log("Hi Lucky");
}
setTimeout(helloWorld, 3000);
console.log("Hello");

o/p
Hello.
Hi Lucky

② **clearTimeout():**
to clear set time out method.

Ex:

③ **setIntervals:**
repeats given function at every given time interval.
(continuously call the function)

④ **clearIntervals:**

Date | | |

Mo Tu We Th Fr Sa Su

Try & Catch:-

① **Try:-**
Statement lets you test a block of code.

② **Catch:-**
Lets handle the error.

Type's of Error:-

① **Range error:-**
Thrown when a number is outside the range.
Ex Array 1–100 :→ If we search 102, will get error.

② **ReferenceError:-**
Thrown when a reference made to a variable is broken or does not exist.
Ex a=10;
a+b // Reference Error.
∴ Because of b is not declared & initialized.

③ **Syntax Error:-**
when the code is understood by JS

④ **Type Error:-**
when operation is wrong using datatypes
let str = "abc";
str.toUpperCase();

Date

Mo Tu We Th Fr Sa Su

⑤ Ex try

```
{
    console.log("Hello coorld");
    return
}
catch(err)
{
    Console.log("There is a error in try block");
}   "     "    (err);
```

error.message = There is a error
error.name = Referred Error,
err.stack = point the all sentance

Throw :- /create

Explicitly throw the error.

Finally :-

whatever the operations of try & catch we have to peform the final operation.

Ex try {

```
} catch (err)
{
finally
{ console.log("finally error") }
}
```

Date

Mo Tu We Th Fr Sa Su

Eg: Throw example:

```
try
{
  if (x == "") throw "It is empty";
  if (isNaN(x)) throw "not a number";
  if (x > 100) throw "It is too high";
  if (x < 10) throw "It is too low";
}
catch (err)
{
  console.log (err);
}
```

Date 12 | 11 | 2022

Mo Tu We Th Fr Sa Su

Regular Expressions:

* RE is a sequence of characters that forms a search pattern.

(i) search():

(ii) exec():-

returns array or null.

Ex: let re = /JavaScript/;
let res = "This is a free JavaScript tutorial by Ramada";
let result = re.exec(res);
// console.log(result);

o/p

["JavaScript", index: 15, input: "_____"]

(iii) test():

(iv) match():

(v) replace():

Date | | |

Mo Tu We Th Fr Sa Su

AddEventListener():-

* addEventListener() is an inbuild function for JavaScript which takes the events to listen. for,

Syntax

element.addEventListener(event, listener, useCapture);

=> ex: <button onClick=" "> Add Event </button>
_
<script>
document.getElementById("sample").addEventListener("click", function()
{ id="hello">
 Hello
});

removeEventListener():

btn.removeEventListener("mouseover", function()
{
});

Date

Mo Tu We Th Fr Sa Su

ClassList Method:

* Is a DOM property of a JS. So it will do styling of CSS classes of element.

Syntax.

Elmt.

classList.add("class-names");

↓

It will add class names.

.remove()

↓

It will remove class name (CSS class)

① classlist.remove("jacky")

② classList.length

~~* conditions~~

③ classlist.contains("wow"); // true / false

④ classlist.toggle()

If you click it will take one CSS class either other class ON/OFF

Date

Mo Tu We Th Fr Sa Su

* createElement():
  Method creates h.
  If you want create any method's (HTML elements. with specified name.
  Ex:- createElement("div"):
  var a = document.
  document.body.appendChild(a);

* parentNode:
  It outputs the parent node of specified node.
  Ex:- li -> ul -> Body -> HTML -> #document.

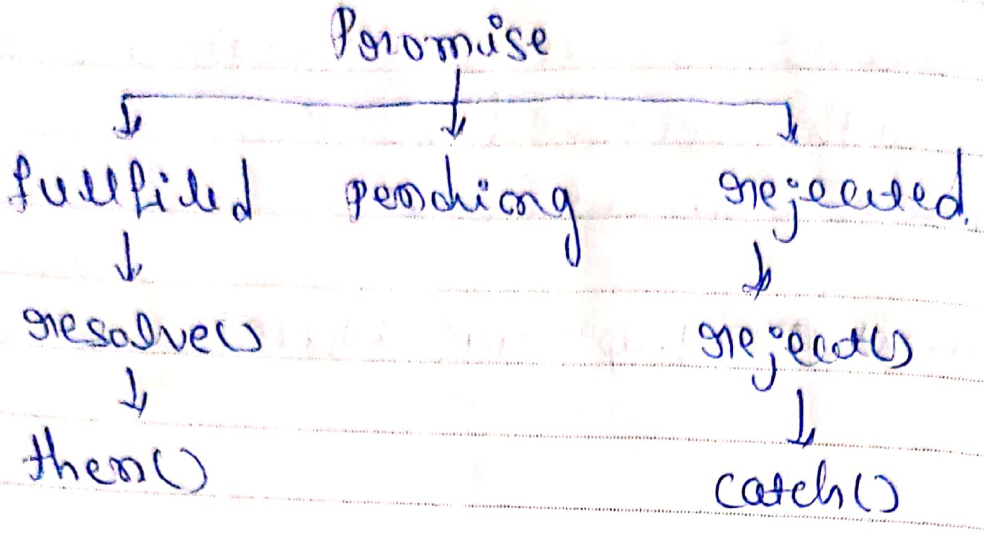
* ParentElement:
  Ex:- li -> ul -> Body -> HTML -> NULL.

add = () =>
{

}

Date

JavaScript Promises



Ex:

```
function prom(meet){
    return new promise( function (resolve, reject){
    console.log('waiting')
    setTimeOut(()=>{
      if(meet)
        → resolve('successfull')
      else(
        → reject("unsuccessfull')
    },2000)
  })
}
```

→ call()

```
prom(true).then(res => console.log(res))
.catch(err => console.log(err))
```

13/11/2022

Mo Tu We Th Fr Sa Su

Date

→ Async & Await →

To overcome promise - then, catch - Async feature.

* Synchronous - Step by step.

Sync

```
console.log('start')
myfunction()
console.log('End')
```

o/p start
Function
End.

Async → Async

```
console.log('start')
setTimeout(() => {
  console.log('my Fun')
}, 3000)
console.log('End')
```

o/p: start
End
My Fun

with Async

```
async function name()
→ 1
await 3
  4
  5
}
console.log(1)
name()
console.log(6)
```

o/p, 1, 2, 3, 6, 4, 5

Constructor: → I Blueprint to n number of objects

→ It is a function that creates an instance of class which typically called an object. It is called when you declare an object using new keyword.

```
Function User(First, last){
    this.firstname: first
    this.lastname: last
}
let user = new User('Kashi', 'Adavi')
Console.log(user)
```

Date

Mo Tu We Th

* Making Responsive using global objects (Mobile, Desktop, Tablet) views.

= navigator.UserAgent.toLowerCase();

Mobile view :

const isMobile = /Android|webOS|iPhone|iPad|iPod|BlackBerry|IEMobile|Opera Mini/i.test(navigator.UserAgent.toLowerCase());

Ex: Array.length = 4;

if (isMobile)
{ this.setState
({
Array.length: 3
});
}

Tablet view :

const isTablet = /(ipad|tablet|(android(?!.*mobile))|(windows(?!.*phone)(.*touch))|kindle|playbook|silk|(puffin(?!.*(IP|AP|WP))))/i.test(navigator.UserAgent.toLowerCase());

Ex

if (isTablet)
{ this.setState
({
});
}

Date 17/01/2023

Mo Tu We Th Fr Sa Su

Objects :

①

* we can create object following ways:

i) `const obj = {name: 'Lucky', age: 25};`

ii)
```
const obj = { };
obj.name = 'Lucky';
obj.age  = 25;
```

iii)
```
const obj = new Object();
obj.name = 'Lucky';
obj.age  = 25;
```

* Objects are mutable (Changeable).
  i) They are addressed by reference not by value.

Ex: `const x = person;`
// will not create a copy of person.

ii) Both x & person are the same object.

iii) Any changes to x will also change person, because x and person are the same object.

Ex:
```
const person = {
    name: 'Lucky'
    age: 25
}
const x = person;
x.age = 30;
```

person.age // 30

x.age // 30

↳ will change both x.age & person.age

Date

* Primitive types are immutable.

(2) Accessing Javascript objects.

objectname.property;
object_name.["property"];
Object_name[expression];

* Javascript for...in loop.

for(let variable in object)
{
  // code to be executed
}

Ex ⇒ const person = { fname: 'Lulay', lnam
'Pujara', age: 25 };
for(let i in Person)
{
  text += Person[i]; // text = text + Pers
}
console.log(text) // Lulay Pujara 25

Date

Mo Tu We Th Fr Sa Su

Javascript Generators:-

* In Javascript, Generator provide a new way to work with functions & Iterators.

Using Generator

* you can stop the execution of a function from anywhere inside the function.
* and continue executing code from a halted position.

Syntax

```
function* generator_function()
{
    ---------
}
// executing generator.
const generator_obj = generator_function();
```

* we can pause the execution of a generator function without executing the whole function body. For that, we use the yield keyword.

Ex: generator define

```
function* generator_func()
{
    console.log("1. code before the first yield")
    yield 100;
}
const generator = generator_func();
console.log(generator.next());
```

Date

Mo Tu We Th Fr Sa Su

JSON.parse():

① It is deserializes a JSON string into a JavaScript object.

JSON.stringify():

② It is a serializes a JavaScript object into a JSON string.

Ex:2 ~~var parseObj = {raw:"Lucy", age:25};~~
~~= JSON.parse stringify(parseObj)~~

① Ex: obj = "{"name": "Lucy", "age": 25}"; // JSON string
// convert JSON string JavaScript Object.

var convert-Java-obj = JSON.parse(obj);
// {name: "Lucy", age: 25}

typeof(convert-Java-obj); // Object.

② Ex: var obj1 = {name: "Lucy", age: 25};
// convert JavaScript Object to JSON str

var convert-JSON-obj = JSON.stringify(obj1)
// "{"name": "Lucy", "age": 25}"

typeof (convert-JSON-obj) // string

Date

Mo Tu We Th Fr Sa Su

Javascript Closures

What is Lexical Scope?

Function scope to access variable from the parent scope, we call the child function to be lexically baend by that of the parent function.

Eg:

```
function Outer()
{
  const a = 1;
  function Inner()
  {
    const b=2;
    console.log(a+b);
  }
  Inner();
}
Outer();
```

Date | | ES-6 features

Mo Tu We Th Fr Sa Su

① Promises:

Using for asynchronous execution.

Ex:

```
var asyncCall = new Promise((resolve, reject) =>
{
    // some logic
    resolve();
}).then(() => {
    console.log('Done!');
});
```

② modules:-

```
export var no = 50;
export function sum(a, b)
{
    return a+b;
}
```

: import {no, sum} from "module"

↳ file name where we configured.

Date | | | ES-6 features

① Let & Const :-

```
let i = 10 ; // console.log (i) = 10.
const PI = 3.142 // console.log (PI) = 3.142.
```

② Arrow Function :-

```
let sum = (a,b) => a+b;
console.log (sum (10,20) // 30.
```

③ Multi-line String :- (back-tick (`))

```
let greeting = `Hello World,
                Greeting to all!`
```

④ Template Literals :-

String templates along with placeholders for the variables

= let name = `My name is ${firstName}`

⑤ Destructuring :-

Destructuring assignment an expression that makes it easy to extract values from arrays, objects.

① Array Destructuring :-

```
let fruits = ["Apple", "Banana"];
let [a,b] = fruits;
console.log (a, b) // Apple, Banana.
```

② Object

```
let Person = {name: 'Uday', age: 20};
let {name, age} = Person;
```

Date | | | React:

Mo Tu We Th Fr Sa Su

React LifeCycle Methods:

* The three phases are;-
1. Mounting
2. Updating
3. Unmounting

1. Mounting:

* Putting the elements into the DOM.

* Whenever that component is inserted into the DOM.

* When component is mounting following methods will be called in order.
i) constructor()
ii) getDerivedStateFromProps() → we set our state
iii) render() → required.
iv) componentDidMount()

2. Updating:

* whenever component is updated, there is a change in state or props.

* Following methods will be called in order
i) getDerivedStateFromProps()

Date

Mo Tu We Th Fr Sa

ii) shouldComponentUpdate()
iii) render()
iv) getSnapshotBeforeUpdate()
v) componentDidUpdate()

3. Unmounting:
whenever destroyed component from the DOM.
= componentWillUnmount()

Date

Mo Tu We Th Fr Sa Su

Context - API

Parent
↓
Child A
↓
Child B
↓
Child C

* If we send data from parent comp to child C. Better approach to context API.

* prop drilling:
Moving props from grandparent to child.

* context-API features is added in version 16.3 of React.

* React.createContext() → It returns a consumer & a provider.

* Provider:-
i) Is a component.
ii) Provides the state to its children.
It will hold the 'store' & be the parent of all components that might need to store.

* Consumer:-
Is a component that consumes & uses the state.

Date

Mo Tu We Th Fr Sa Su

React Hooks:

* Basically Hooks are using in React Functional component.

* Hooks will work only higher than 16.8 version of React.

* Hooks can used for delivering React Class Lifecycle methods in Functional component.

Ex:
```
function App()
{
  return
  (
    <h1> Hello Functional component </h1>
  );
}
```

1. useState():

It is used for track state in a functional comp-

```
import React, {useState} from "react";
function App()
{
  const [name, setName] = useState("");
}
```

Date

Mo Tu We Th Fr Sa Su

2. useEffect():

* It will work as componentDidMount, DidUpdate and willUnMount().
* Basically it's using for API - call in functional component.

```
useEffect(() =>
{
},[])
```

1st arg → function
2nd arg → dependency
[] → call only once.
[name] → call only when state is changed.
~~[]~~ → call every render.

Date

Mo Tu We Th Fr Sa Su

Redux:

* Redux is a predictable state container for javascript application.
* Redux is a library for Javascript App.
* Redux stores the state of an application.
* Redux will store state & manage the application state.
* We can use Angular, Vue, & Vanilla JS
* Uses to manage the global state of application in a predictable way.
* Redux is easier to understand when, where, why and how the state in the application is updated.

Date

Mo Tu We Th Fr Sa

Redux Step's:

1. Install Redux toolkit & React Redux
   npm install @reduxjs/toolkit react-r
2. create Redux store.
3. Provide Redux store to react.
4. create a Redux state slice.
5. Add slice Reducer to the store.
6. Use React State & actions in React Application.

Date | | | Practice

Mo Tu We Th Fr Sa Su

1. npm install @reduxjs/toolkit react-redux

2. Create store file = store.js
import {configureStore} from "@reduxjs/toolkit";

export const store = configureStore
({
reducer: {}
})

3. Go to index.js file.
import {Provider} from "react-redux";
import {store} from "./store.js";
Then wrap with Provider with over App component (Parent Comp).
<Provider store={store}>
<App />
</Provider>

4. Creat -> redux folder -> features -> counterSlice.js
=
import {createSlice} from "@reduxjs/toolkit";

```
const initialState = {
    count: 0
};

export const counterSlice = createSlice({
    name: "counter",
    initialState,
    reducers:
    {
        increment: (state, action) => {state.count + 1},
        decrement: (state, action) => {
            state.count - 1},
    }
})

export const {increment, decrement} =
                counterSlice.actions

export default counterSlice.reducer
```

2.

5. Go to store file and configure reducer.

```
= import counterReducer from "./features/counter/counterSlice";

    reducer:
    {
        counter: counterReducer
    }
```

Date

Mo Tu We Th Fr Sa Su

6. ~~Import~~ React component

= import {useSelector, useDispatch} from "react-redux";

function Counter() → const dispatch = useDispatch();

{

const count = useSelector((state) => state.counter.count)

return

(

<>

<h3> count: {count} </h3>

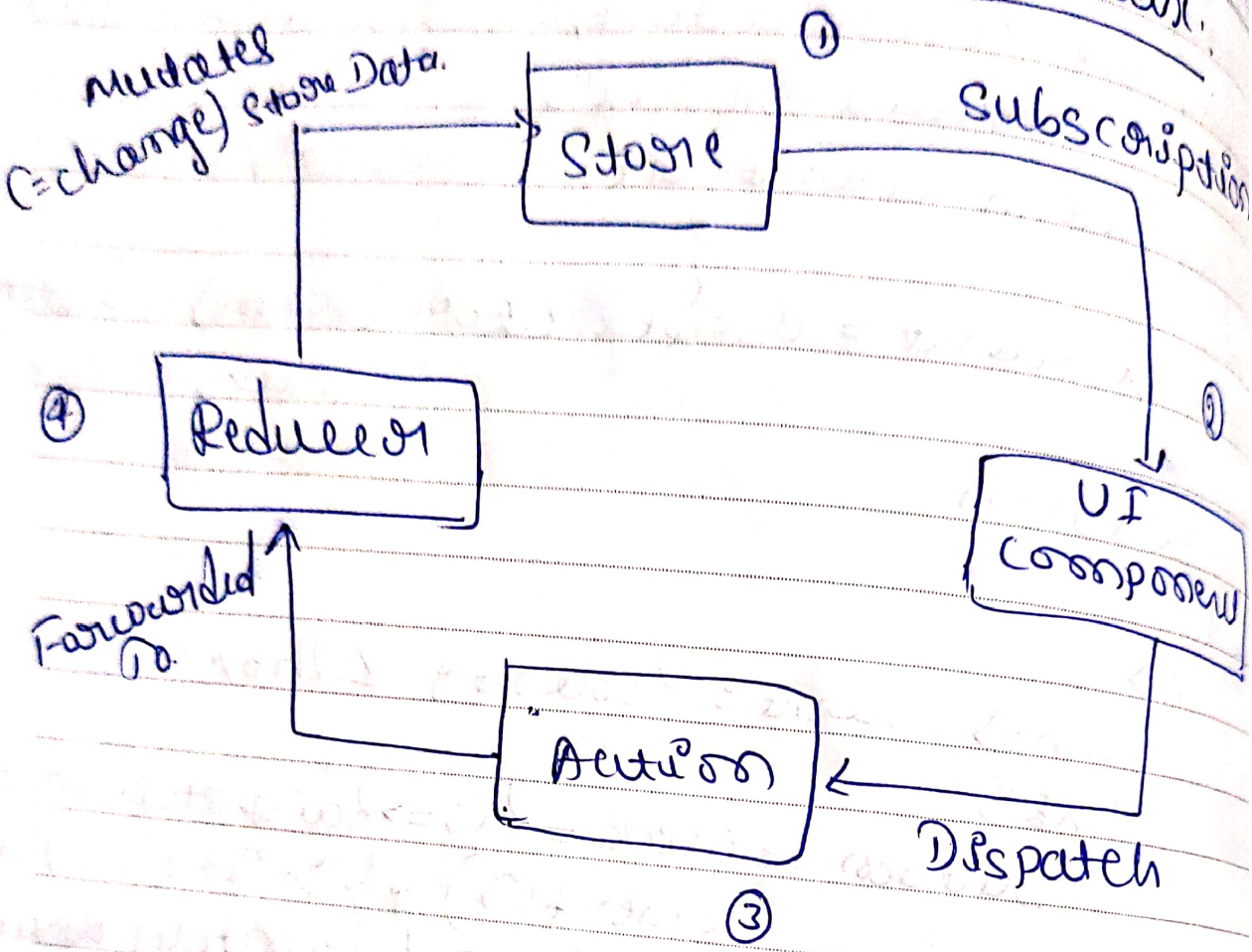
<h2>

<button onClick = {() => {dispatch(increment())}}> Inc + </button>

<button onClick = {() => {dispatch(decrement())}}> Dec - </button>

Date

Plain JavaScript Redux.



1. Create Store.

= const redux = require('redux').

= ~~const store = redux.createStore();~~

= const counterReducer = (state = { counter: 0 }, action) =>
{
return
{
counter: ~~state~~.counter + 1.
}.
};

Date

Mo Tu We Th Fr Sa Su

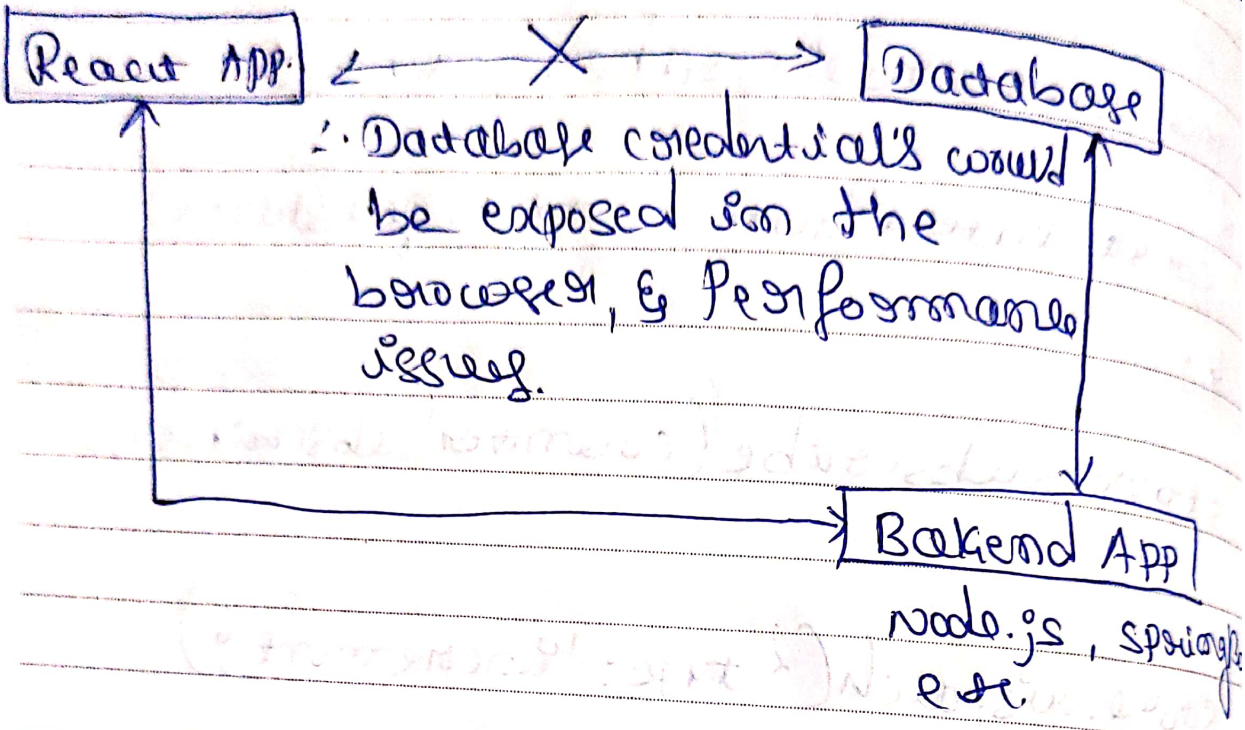
```
= const store = redux.createStore(counterRe-
                                  -ducer);

= const counterSubscriber = () =>
  {
    const latestState = store.getState();
  };

store.subscribe(counterSubscriber);

store.dispatch({type: 'increment'});
```

Jai Anjaneya

Date 01 01 2023

Mo Tu We Th

How to connect database with Re



* API (Application programming Interface)
* For using dummy api:
  https://swapi.dev/
* For api call using fetch method and Async & await.

```
func = async () =>
const res = await fetch('_____url_____')   ∴ default it will consider as GET method.
const newData = await res.json();
to
console.log(newData);
```

}

Mo Tu We Th Fr Sa Su

Date

* without Async & await, Using for Promise.

```
func = () =>
{
fetch (" ____ url ____ ") - // default GET url.
.then (response =>
{
return response.json();
});
.then ((data) =>
{
console.log(data);
});
```

req
[2+ ____]

Date

Mo Tu We Th Fr

Post method:

get → request i/p

```
async function postData()
{
  const response = await fetch('url',
  {
    method: 'POST',
    body: JSON.stringify(res);
    headers:
    {
      'Content-Type': 'application/json';
    }
  });

  const finalResponse = await response.json()
  console.log(finalResponse);
```

∴ we will get sent data in finalRespon object.

Deployment Steps

Date 09/01/2023

Mo Tu We Th Fr Sa Su

* After Developing code from your local system into production system.

Steps:

① Test code:
Test code for testing full application based diff scenarios.

② Optimize code:
Lazy Loading,

③ Build App for Production:
Based on scripts it will create a bundle for production code.

④ Upload Production code to Server

⑤ Configure server:

~~②~~ ② Lazy Loading:

* It's improve a App Performance, using React Router. {Suspense}

import React from "react";

const Homepage = React.lazy(() => import('./page.jsx'));

Date

```
function App() {
  return (
    <div>
      <Suspense fallback={<h1>Loading...</h1>}>
        <Homepage />
      </Suspense>
    </div>
  );
}
```

∴ `const Homepage = React.lazy(() => import('...'))`

* Lazy Loading is almost using for routes.
* Reason is unnecessary calling pages will be stopped.
* If we use React.lazy() it will call only when render will call (Routing page) time only.

Date

Mo Tu We Th Fr Sa Su

③ Build App for Production:

* Upon run build, we will get a build folder.

* It bundles CSS, JS for optimizing code.

① Upload Production code to Server:

* We have to choose a cloud platform for deploying our code to production like AWS console, Google console or Firebase console. (Ours).

⑤ Configure Server.

Firebase deploy

Testing React Apps

Date 03/01/2023

Mo Tu We Th Fr

* Manual Testing:
  i) write code <> Preview & Test in Browser
  ii) Very Tedious:
      You see what your user will see.

∴ Error-prone :- It's hard to test all possible combinations & scenarios.

* Automated Testing :-
  i) need to write code that tests your code.
  ii) You test the individual building blocks of your app.

∴ very technical but allows you to test All building blocks at once.

⇒ Testing Types :

- Unit Tests
  ① Test a small parts, like, component functions
- Integration Tests
  ① combination of multiple building blocks.
  * More than one component involved.
- End-to-End (E2E) Tests
  ① Test complete scenarios in your.

Mo Tu We Th Fr Sa Su

Date

* For React Testing Tools & Library.

Tool: Jest

Library: React Library

Syntax (Unit Test)

↑ description of Test

```
test('_____', () => {
    render(<App />);
    const linkElement = screen.getByText(/learn react/i);
    expect(linkElement).toBeInTheDocument();
});
```

→ arrow function.

① render → component name

② screen

→ import {render, screen} from '@testing-library/react';

: npm test (Running test)

* Writing Tests :- The Three "A"s

i) Arrange ⇒ Set up the test data, test conditions, and test environment.

ii) Act ⇒ Run logic that should be tested. (like, execute function).

iii) Assert ⇒ Compare execution result with expected result.

Date 05/01/2023

* Echo system of React. (Related to the
  i) Gatsby.js
  ii) Preact
  iii) React Native (Building for mobile app.
  or android)

Authentication works in React Apps:

Date 05/01/2023

Mo Tu We Th Fr Sa Su

* Authentication is needed if content should be protected (not accessible by everyone).

* when working with 'Authentication token', these tokens are typically created in the "JSON Web Token" Format (JWT).

* Tokens:-
  i) Tokens are just long string.
  ii) which are constructed by an algorithm that encodes data into a string (with help of private key, only server can know the data).

React Hooks

Date 09/01/2023

* React Hooks runs above 16.8.6 version only
* we can't use inside functions, or conditions (if)

① useState:

* The useState can be object, array, string, Boolean, etc.
* The class based component state always was an object.

Syntax:

- import React, {useState} from "react";
- const [demo, setDemo] = useState(' ');

demo → initial state
setDemo → Updated state.

Ex: ① const [inputState, setInputState] = useState({title:'', amount:''});

return
(
<form>
<label> Title </label>
<input type="text" value={inputState.title}
onChange={event =>
{ const title = event.target.value;
setInputState(prevState => ({
...prevState, title: title
}))
}} />
</form>
)

Date

Mo Tu We Th Fr Sa Su

=> Multiple useState

Ex ②

```
const [title, setTitle] = useState('');
const [amount, setAmount] = useState('');
```

```
return
(<form>
  <label> Title </label>
  <input type="text" value={title}
    onChange={event => {
      setTitle(event.target.value)
    }} />

  <label> Amount </label>
  <input type="Number" value={amount}
    onChange={event =>
      setAmount(event.target.value);
    } />
```

O/P

Title
i/p [ ]

Amount
i/p. [ ]

[submit]

Date

* React Hooks will declare on top only After creating the function.

* If you want to send data from one comp to other comp.

Demo.js

<Demosamp data={'newData'}

↓ component

↓ property name

↓ function to data (reference)

Demosamp.js

const handleClick = () =>

{

props.data({title: 'Book', amount: 200

});

return (<p onClick={handleClick}>

Date

Mo Tu We Th Fr Sa Su

② useEffect()

* It's a side effects.
It's equivalent to → componentDidMount(), componentDidUpdate(), componentWillUnMount().

* If we need to call API (backend_API), UseEffect() way.

* It Runs the function After Every component Update (re-rendered).

Syntax:

```
useEffect(() =>
{
    ____________
},[])
```

↳ It's Runs the only one time (componentDidMount())

,[name]
↳ It's runs only once, when name is changed.

Date

Mo Tu We Th Fr Sa Su

JS coding questions

1. Remove duplicates characters from a given string.

i/p str = adhaar

o/p adhar

2. Find maximum occurring character in a string.

i/p str = apple

o/p = p

3. Can you write nested objects.?

```
const obj = { name: 'vinay', age: 20,
              address: {
                        first_addr : " ",
                        second_addr : " "
                       }
            };

const obj1 = obj;
obj1.address.first_addr = 'bangalore';

obj1.add.print {
obj.add.print.
        (
```

(6b) How are we integrating html response with
(7) Explain Redux flow in class components
(8) diff b/w Flux & Redux?
(9) which middleware, Are using & exp
(10) what's a new features in React
(11) what is React Fiber? (which v they
(12) Diff b/w controlled & uncontrolled
component? onChange() (React) ref (Dom)
(13) what are synthetic events in Reac
↳ stopPropagation() & preventDefault()
(14) what is the diff b/w ES5 & ES6 stand
while creating the class.
(15) How are we configuring webpack file?

HTML & CSS

(1) What are the block level & Inline level El
in HTML? B-> <h1>-<h2>, <p>, <li>, In-><

(2) How you will make your web pages as respo
-sive? can write the mobile view media quer

(3) what are the semantic elems? & can
semantic elements in HTML?
S-> <form>, <table>, & <article>,
N->

(4) Can you explain me position property in CS
what is default position?

(5) diff b/w querySelector & querySelectorAll()

(6) what is z-index?

Date 1

Mo Tu We Th Fr Sa Su

React Top 20 Questions.

Tell me about yourself.

1. what is generator function? Exp with ex:
2. diff b/w call() & apply() in JS
3. what is Higher Order component in React?
4. what is diff b/w shallow copy & deep copy?
5. what is use of closure? and have you used closure in real application.
6. diff b/w slice & splice() methods in Array?
7. what is the use of Symbol data type?
8. what is use of addEventListener()?
9. Diff b/w Rest & Spread operators?

10. what is diff b/w useMemo & useCallback? explain with react example?
11. Do you worked on Redux? which version are you using?
12. How to convert componentWillmount in functional component?
13. Are you using any Testing Library? Apart from Test Lib.
14. what is the use of useReducer & useRef.
15. what is the use of JWT Token?

Date 12/01/2023

Mo Tu We Th Fr Sa Su

React Custom Hooks

* Only call React Hooks in React Functio-nal component.
  i) React component Functions
  ii) Custom Hooks

* Only call React Hooks at the Top Level
  i) Don't call them in nested functions.
  ii) Don't call them in any block statements.

Custom Hooks:-
* Outsource ~~logic~~ stateful logic into re-usable functions.
* By end it's a re-usable function.

i) creating custom Hooks steps:-
  i) creating Hooks name start with "use" keyword

Eg: const useCounter = () =>
{ ——— => function code
}

ii) use as a just function

iii) Don't use any returns and html body.

Date

Mo Tu We Th Fr Sa Su

⇒ React Props Validation:

App(component name).propTypes =

{

}

< import PropTypes from 'prop-types'

Ex: App.propTypes =

{

name: PropTypes.array.isRequired,

age : PropTypes.number.isRequired

}

App.defaultProps =

{

name : 'Lelay'

age : 10

}

Date 01/03/2023 CSS

Mo Tu We Th Fr Sa Su

* CSS -> Cascading Style sheets

* 3 types styling we can do
i) Inline style.
ii) Internal style.
iii) External style.

* CSS selectors:
we can devide CSS selectors into 5 categories.
i) Simple Selectors
(select elements based on name, id, class)

ii) Combinator selectors:
(select elements based on a specific relationship between them).

iii) Pseudo-class Selectors:
(select elements based on a certain state).

iv) Pseudo-element Selectors:
(select & style a part of an element).

v) Attribute selectors:
(Select elements based on an attribute or attribute value).

Date

* In this example only <p> elements with class = "center" will be red and center aligned.

```
<body>
  <h1 class="center"> ---- </h1>
  <p class="center"> ---- </p>
</body>
```

~~p.class~~ → p.center

{ text-align : center;
  color : red;
}

* Here the <p> tag will be styled according to class = "center" and to class = "large"

```
<p class="center large">
```

p.center large

{ }

* The CSS Universal Selector:

The universal selector (*) selects all HTML element on the page.

```
* {
    text-align : center;
    color : red;
}
```

Mo Tu We Th Fr Sa Su

Date

* Backgrounds CSS :-

i) background-color :-

```
body
{ background-color : red;
}
```

ii) Opacity / Transparency :-

* The opacity property specifies the opacity/ transparency of an element.
* It take a value from 0.0 - 1.0.
  The lower value is more transparent.

Ex:-

```
div
{ background-color : green;
  opacity : 0.3;
}
```

Date

Mo Tu We Th Fr Sa

* **display:**
  The display property specifies if/how an element is displayed.
  The default display value for most elements is block or inline.

* **Block-Level Elements:**
  Always starts on a new line & takes up the full width available.
  Ex: <h1> — <h6>, <p>, <form>, <header>, <footer> <sec

* **Inline Elements:**
  Does not start on a new line & only takes up as much width as necessary.
  Ex: , <img>, <a>

* Max-width & width diff
  ∴ Drag the browser window to smaller than 500px wide, to see the difference b/w two divs.

Mo Tu We Th Fr Sa Su

Date

* Position:-
The position property specifies the type of positioning method used for an element (static, relative, fixed, absolute or sticky).

position: static;
* HTML elements are positioned static by default.
* It is not affected by the top, bottom, left, right properties.

position: relative;
* Setting the top, right, bottom, left properties of a relatively-positioned element will cause it to be adjusted away from its normal position.

Ex div {
position: relative;
left: 30px;
border: 3px solid red;
}

position: fixed:-
It means always stays in the same place even if the page is scrolled.

Ex div {
position: fixed;
right : 0;
}

Date / /

Mo Tu We Th Fr Sa

position: absolute:

It uses the document body, & moves alo
with page scrolling.

```
div{
    position: absolute;
    top: 80px;
    border: 3px solid red;
}
```

position: sticky:-

This position based on the users scroll posi

Ex: div{

```
    position: sticky;
    position: -webkit-sticky; // Safari
    top: 0;
    background-color: green;
    border: 2px solid red;
```

Scanned with OKEN Scanner

Date

Mo Tu We Th Fr Sa Su

Higher-Order function.

The Higher order function that accepts functions as parameters and/or returns a function.

Eg: map, reduce, filter, and forEach loop methods.

Functions

① function function_name() // function definition.
{

}

function_name(); // function calling.

② passing parameters into a function.

```
function sum(a,b)
{
  return a+b;
}
sum(10,20);
```

O/P

30,

Date

Mo Tu We Th Fr Sa Su

③ Anonymous function:-

```
let res = function (a, b)
          {
            return a*b;
          };
res(10,10);                    // o/p
                               // 100.
```

④ Function constructor:-

```
const myFunction = new Function("a", "b",
                   "return a*b");
let x = myFunction (4,3);      // o/p
console.log(x);                // 12
```

⑤ Self Invoking Function:-

Ex: A self Invoking expression is invoked automatically, without being called.

It will execute automatically if the expression followed by ().

Ex:

```
(function()
{
  console.log("Hiiii");
})
();
```

Mo Tu We Th Fr Sa Su

Date

* arguments.length :-
property returns the no of arguments received when the function was invoked.

Ex:
function myFunction(a, b)
{
return arguments.length; // 2.
}

* Diff b/w Call & Apply :
i) call() method takes arguments separately.
ii) we can write a method that can be used on diff objects.
i) apply() method takes arguments as Array.

= const p1 = { firstName: "Lilay", lastName: "pujari" }
const p2 = { firstName: "Appu", lastName: "pujari" }
const Person = {
fullName: function() {
return this.firstName + " " + this.lastName;
}
}
Person.fullName.call(p1) o/p
" " .call(p2)

= Apply() :
We Person object and p1 object. only change return in person object.
+ "city + ", " + country;
}
Person.fullName.apply(p1, ["Bangalore", "India"]);

Date

* what is diff b/w Set and Map

Set

i) Set is one dimensional unique Array.

ii) Syntax:

```
const setObj = new Set(); //create set object
setObj.add("a");
setObj.add("b");
setObj.add("c");
console.log(setObj) // a,b
for (let i of setObj.values())
{
  console.log(i)          // "a"
}                         "b", "c"
let text="";
setObj.forEach(function(x)     // "a"
{                              "b"
  text += x;                   "c"
})
```

Map

i) Map is 2D and has key-value pair, where key shall be unique.

ii) Syntax:

```
const mapObj = new Map();
mapObj.set("name", "Lucky");
  "      "   ("age", 26);

let text = ""; loop
mapObj.forEach(function(val, key)
{
  text += key + "=" + val;
});
```

Mo Tu We Th Fr Sa Su

Date

Javascript Async, concepts

1. callbacks
2. Asynchronous
3. Promises
4. Async / Await.

1. callbacks:

* A callback is a function passed as argu-ments to another function.
* This technique allows a function to call another function.
* A callback function can run after another function has finished.

Ex

```
function myDisp(res)
{
  console.log(res)
}
function myCalc(num1, num2, callBack)
{
  let sum = num1 + num2;
  callBack(sum);
}
myCalc(10, 20, myDisp);
```

Mo Tu We Th Fr Sa Su

Date

Javascript Async concepts

1. callbacks
2. Asynchronous
3. Promises
4. Async / Await.

1. callbacks:

* A callback is a function passed as argu-ments to another function.
* This technique allows a function to call another function.
* A callback function can run after another function has finished.

Ex

```
function myDisp(res)
{
  console.log(res)
}
function myCalc(num1, num2, callBack)
{
  let sum = num1 + num2;
  callBack(sum);
}

myCalc(10, 20, myDisp);
```

Date

2. Asynchronous
* Functions are running in parallel with other functions are called asynchronous
* Best example
setTimeout(), & setInterval()

Ex:
setTimeout(myFunction, 1000);

function myFunction()
{
console.log("Hi Asynchronous call");
}

3. Promise:
i) "Producing code" is code that can take some time.
ii) "Consuming code" is that must wait for result.

Eg: Syntax
const promiseObj = new Promise(function
{ (resolve, reject)

resolve();
// reject();

promiseObj.then(
function(val) { ——— },
function(val) { ——— }
);

Mo Tu We Th Fr Sa Su

Date

Ex

```
function myDisp(res)
{
  console.log(res);
}

const myPromise = new Promise(function
      (resolve, reject) {
  let x=0;
  if(x==0)
  {
    resolve("Ok");
  }
  else
  { reject("Error");
  }
});

myPromise.then(
  function(val)
    {
      myDisp(val);
    },
  function(val)
    {
      myDisp(val);
    }
);
```

Date

Mo Tu We Th Fr Sa

4. Async/Await :-

* Async makes a function return a Promise.
* Await makes a function wait for a Promise.

Ex async function display() {

let promiseObj = new Promise(function (resolve, reject) {
resolve("Hi Promise");
});

await promiseObj;

display();

Mo Tu We Th Fr Sa Su

Date: 08/04/2023

Javascript BOM (Browser Object Model)

* window object is supported by all browsers.
* All global Javascript objects, functions, and variables automatically become members of the window object.
* Global variables are properties of the window object.
* Global functions are methods of the window object.
* Even the document object (HTML DOM) is a property of the window object.

Ex:

document.getElementById("header");

⇓ equivalent to

window.document.getElementById("header");

Ex: (Methods)

1. window.innerWidth;
2. " .innerHeight;
3. " .open() → open a new window.
4. " .close() → close the current window.
5. " .resizeTo() → resize the current window.

Date

1. Javascript Window Screen:

* window.screen. Object containing information about the user screen.

Properties.

i) window.screen.
ii) screen.availHeight:
iii) " .availWidth:
iv) " .width:
v) " .height:

2. JavaScript Window Location:

* The window.location object can be used to get the current page address (URL) and to redirect the browser to a new page.

Property

i) window.location.
ii) window. " .hostname → domain name
iii) " . " .href → URL of the current page.
iv) " . " .pathname → returns the path & filename of current page.
v) " . " .port → returns port no.
vi) " . " .origin → returns the load webpage.

Mo Tu We Th Fr Sa Su

3. JavaScript Window History:

* window.history object contains the browser history.

properties:-

i) window.history.back() → same clicking back in the browser.

ii) window.history.forward() → same clicking as forward in the browser.

4. JavaScript Cookies:-

* Cookies let you store user information in web page.
* cookies are small text files, data stored on your computer.
* cookies are saved in name-value pairs.
  = username = Lucky Pujari

i) create cookie

= document.cookie = "Lucky Pujari";

ii) Read cookie.

let x = document.cookie;
console.log(x);

iii) delete cookie.

document.cookie = "username=;";

Date

JSON (Javascript Object notation)

* JSON is a text format for storing and transporting data.

* JSON string
'{"name": "Uday", "age": 25, "car": null}'

* JSON.parse():
It's converting JSON string into Javascript object.

* JSON.stringify()
It will convert Javascript into a JSON string.

⇒ JSON vs XML

* {"employees": [
{"firstName": "John", "lastName": "Smith"},
{"firstName": "Uday", "lastname": "Pujari"}
]}

XML

```
<employees>
  <employees>
    <firstName>John</firstName>
    <lastName>Smith</lastName>
  </employees>
</employees>
```

Mo Tu We Th Fr Sa Su

Date

* JSON values cannot be one of the following data types.
i) function ii) data iii) undefined.

Date 20/04/2023

1. Diff b/w Rest & Spread operators.

Rest operator	Spread
Pack element.	• unpacked element.
	• shallow copy.

spread:

Ex:- const arr = [1,2,3];
console.log(...arr); // 1,2,3.

Ex 2:- const arrA = [1,2,3];
const arrB = [...arrA];
console.log(arrB); // 1,2,3.

If we do changes to either of the arr they will not affected the other.

Rest:

```
function sum(...args)
{
let res = 0;
for(let arg of args)
{
  res += arg;
}
return res;
}
sum(10,20,30,40);
```

Date 24/04/2023

Mo Tu We Th Fr Sa Su

1. Javascript prototype:

* Javascript, every function & object has a property named prototype by default.

Ex: function Person()
{ this.name = "Lucy";
  this.age = 25;
}

const p1 = new Person();
console.log(Person.prototype); // {---}
// This prototype has no value at the moment.

* A prototype can be used to add properties & methods to a constructor function.

Syntax:

objectConstructor.prototype.key(property name) = value;
name

Note:
i) Add property to a constructor function.
ii) " Methods to a " " using prototype

Ex: function Person()
{ this.name = 'Lucy';
  this.age = 26;
}

const obj1 = new Person(); // Person {name: 'Lucy', age: 26}
Person.prototype.salary = 20000;
console.log(obj1.salary) // 20000;
Person.prototype.address = function()
{ console.log('my name is' + this.name);
}

console.log(obj1.address()); // my name is Lucy

Date

2. Default Parameter value.

```
function sum (x, y=5)
{
    console.log(x+y); // 2
}
sum(5);          // x=5, y=5 = 10
sum(5,15);       // x=5 y=15 = 20.
```

∴ If don't pass the parameter for y it will take by default as 5.