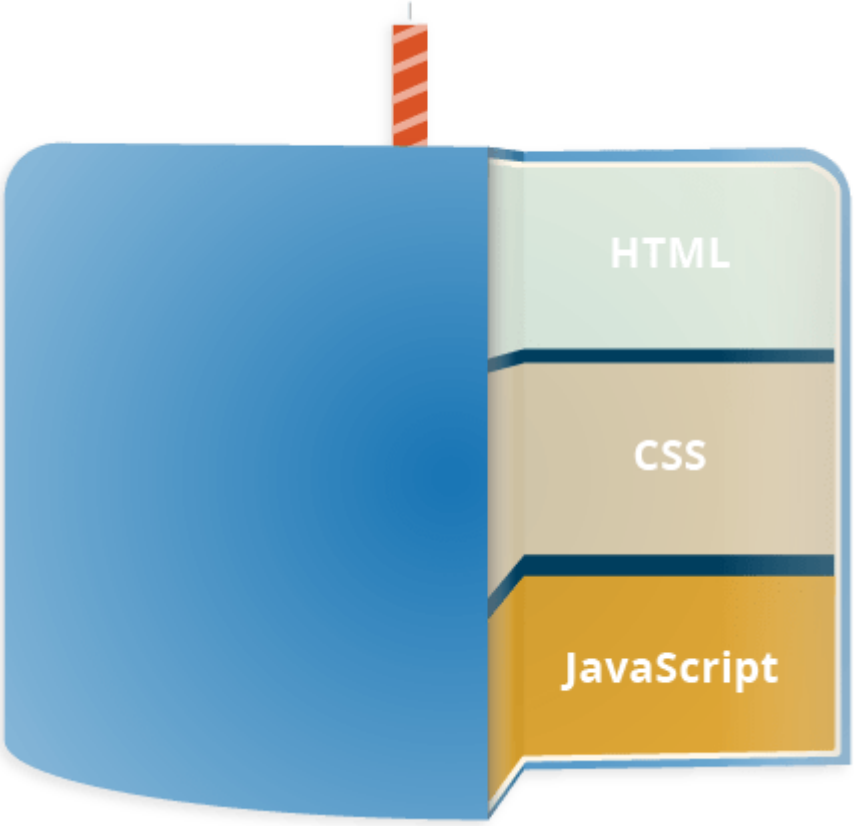
JavaScript

Apna College Web Development Course

[Type the abstract of the document here. The abstract is typically a short summary of the contents of the document. Type the abstract of the document here. The abstract is typically a short summary of the contents of the document.]

[Pick the date]



HTML is the markup language that we use to structure and give meaning to our web content, for example defining paragraphs, headings, and data tables, or embedding images and videos in the page.

CSS is a language of style rules that we use to apply styling to our HTML content, for example setting background colors and fonts, and laying out our content in multiple columns.

CHAPTER NO 1

What is JavaScript?

JavaScript is a scripting language that enables you to create dynamically updating content, control multimedia, animate images, and pretty much everything else.

JavaScript is the world's most popular programming language.

JavaScript is the programming language of the Web.

JavaScript is easy to learn.

JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997.

JavaScript running order

When the browser encounters a block of JavaScript, it generally runs it in order, from top to bottom. This means that you need to be careful what order you put things in. For example, let's return to the block of JavaScript we saw in our first example:

```
const button = document.querySelector("button");

button.addEventListener("click", updateName);

function updateName() {

  const name = prompt("Enter a new name");

  button.textContent = `Player 1: ${name}`;

}
```

If you swapped the order of the first two lines of code, it would no longer work — instead, you'd get an error returned in the browser developer console — **Uncaught ReferenceError: Cannot access 'button' before initialization.** This means that the button object has not been initialized yet, so we can't add an event listener to it.

Server-side versus client-side code

Client-side code is code that is run on the user's computer — when a web page is viewed, the page's client-side code is downloaded, then run and displayed by the browser. In this module we are explicitly talking about client-side JavaScript.

Server-side code on the other hand is run on the server, then its results are downloaded and displayed in the browser. Examples of popular server-side web languages include PHP, Python, Ruby, ASP.NET, and even JavaScript! JavaScript can also be used as a server-side language, for example in the popular Node.js environment — you can find out more about server-side JavaScript in our Dynamic Websites – Server-side programming topic.

Dynamic versus static code

The word **dynamic** is used to describe both client-side JavaScript, and server-side languages — it refers to the ability to update the display of a web page/app to show different things in different circumstances, generating new content as required. Server-side code dynamically generates new content on the server, e.g. pulling data from a database, whereas client-side JavaScript dynamically generates new content inside the browser on the client, e.g. creating a new HTML table, filling it with data requested from the server, then displaying the table in a web page shown to the user.

A web page with no dynamically updating content is referred to as **static** — it just shows the same content all the time.

Relationship between JavaScript, APIs, and other JavaScript tools

why we should create js seperate file? ans:(readability,modularity(code is divided into parts like html,css,js),browser caching(loading will be faster))

using the console

repl(read evaluate print loop)

ctrl+l=clear

console.log("string") : to display message on console

console.error("error") : to display error on console

console.warn("warn"): to display warn on console

alert display an alert message on the web browser > alert("message")

prompt displays a dialog box on the web browser >prompt("enter a number")

Our 1st JS Code

Console.log is used to log (print) a message to the console

console.log("Apna College");

Variables in JS

Variables are containers for data.

Name storage location.

Variable Rules Variable (identifier rules)

- names are case sensitive; "a" & "A" is different.
- Only letters, digits, underscore(_) and $ is allowed. (not even space)
- Only a letter, underscore(_) or $ should be 1st character.
- Reserved words cannot be variable names.

let, const & var

var : Variable can be re-declared & updated. A global scope variable.

let : Variable cannot be re-declared but can be updated. A block scope variable.

const : Variable cannot be re-declared or updated. A block scope variable.

Data Types in JS

Primitive Types(basic or fundamental data types) : Number, String, Boolean, Undefined, Null, BigInt, Symbol

Nan:not a number is special type of number(itself is number but represent the value is not a number)

In JavaScript, **dynamic typing** means that a variable's type is determined at runtime and can change as needed, allowing variables to hold values of any type without explicit type declarations.

```
let char = 'a';

document.write(typeof (char));
```

Literal: A fixed value or representation of a data type directly written in code.

Examples of literals const obj = { key: 'value' };care numbers (10), strings ("hello"), arrays ([1, 2, 3]), and objects ({ key: 'value' }).

Types of literal:

- Numeric literal: 10
- String literal: "hello"
- Boolean literal: true
- Object literal: { name: "Alice", age: 25 }
- Array literal: [1, 2, 3]
- Null literal: null
- Undefined literal: undefined

CHAPTER NO 2

Comments in JS

Part of Code which is not executed.

```
1   //This is a single line comment
2
3   /* This is a multi-line
4      comment. */
```

Operators in JS

Used to perform some operation on data

Arithmetic Operators

+, -, *, /

- Modulus %
- Exponentiation **
- Increment ++
- Decrement --

Operators in JS Assignment Operators

= += -= *= %= **=

Operators in JS Comparison Operators

Equal to == Not equal to != Equal to & type ===

Not equal to & type !==

> greater than , >= greater than or equal to , <= less than or equal to , < less than

Operators in JS Logical Operators

Logical AND &&

Logical OR ||

Logical NOT !

Ternary Operators

condition ? true output : false output

```
age > 18 ? "adult" : "not adult";
```

Operator precedence

This is the general **order** of solving an expression.

() bracket (5+2)/7+1*2

** power operator (5+2)/7+1**2

Note* for multiple powers we check from right to left

*,/,% check left to right 1+2

+,- check left to right 3

Conditional Statements

To implement some condition in the code if Statement

```
let color;
if(mode === "dark-mode") {
    color = "black";
}
```

if-else Statement

```
let color;
if(mode === "dark-mode") {
    color = "black";
} else {
   color = "white";
}
```

else-if Statement

```
if(age < 18) {
    console.log("junior");
} else if (age > 60) {
   console.log("senior");
} else {
    console.log("middle");
}
```

Switch

Used when we have some fixed values that we need to compare to.

```
let color = "red";

switch(color) {
    case "red" :
        console.log("stop");
        break;
    case "yellow" :
        console.log("slow down");
        break;
    case "green" :
        console.log("GO");
        break;
    default :
        console.log("Broken Light");

}
```

Scope

Determines the accessibility of variables, objects, and function from different parts of the code.

Function scope

Variable defined inside a function are not accessible(visible) from outside the function

Block scope:

Variable declared inside a {} block cannot be accessed from outside the block

Lexical scope:

A variable defined in outer function can be accessible inside inner function defined after the variable declaration

THE opposite is not true

Nested function concept will be used here

Global scope:

Variable defined outside of the functions and accessible everywhere.

CHAPTER NO 3

Loops in JS

Loops are used to execute a piece of code again & again

for Loop

```
for (let i = 1; i <= 5; i++)
{
console.log("apna college");
}
```

Infinite Loop : A Loop that never ends.

while Loop

```
while (condition)
{
// do some work
}
```

do-while Loop

```
do
{
// do some work
}
whil e (condition);
```

for-of Loop used for string and arrays

```
for(let val of strVar)
{
//do some work
}
```

Nested

```
let array=[["siraj","dujana"],["qureshi","israr"]]

        for (const list of array) {

            for (const hero of list) {
                console.log(hero+"<br>")
            }
        }
```

For-in Loop

we can used it for arrays and objects.

```
for (let key in objVar)
{
//do some work
}
```

Nested loops

Loop inside loop

```
For(let i=1;i<=5;i++)
{
        For(let j=1;j<=5;j++)
        {
        }
}
```

Strings in JS

String is a sequence of characters used to represent text

Create String

```
let str = "Apna College";
```

String Length

```
str.length
```

String Indices

```
Let str="ABC"
str[0], str[1], str[2]
```

Template Literals in JS

A way to have embedded expressions in strings

```
`this is a template literal`
Console.log(`this is sum of ${a+b}`);
```

String Interpolation

To create strings by doing substitution of placeholders

```
`string text ${expression} string text`
```

Truthy and falsy values

Everything in JS is true or false(in Boolean context)

This doesn't mean their value itself is false or true, but they are treated as false or true if taken in Boolean context.

False values

False,0,-0,on(bigint value),""(empty string), null, undefined, NAN

Truthy values

Everything else

String Methods in JS(methods: action performed on objects)

Strings are immutable in js (string can't be change)

These all methods will not change the original string but return a new string with changing

These are built-in functions to manipulate a string

- str.toUpperCase( ) convert text to upper case
- str.toLowerCase( ) convert text to lower case
- str.trim( ) removes whitespaces
- str.slice(start) returns part of string as a new string
- str.slice(start, end?) // returns part of string as a new string
  - Ending value is not included ex: str.slice(1,5) so it will be executed as str.slice(1,4)
- str.slice(-value)=str.slice=(length-value) returns part of string as a new string
- str1.concat(str2) joins str2 with str1
- str.replace(searchVal, newVal) search a value in the string and return the string with replace value.
- str.charAt(idx)
- str.indexpf("text"); return the first index of text or gives -1 if not found
- str.repeat(value) return a string with the number of copies of a string

Method Chaining

Using one method after another.

Order of execution will be left to right.

Ex:

Str.toUpperCase().trim();

CHAPTER NO 4

Arrays in JS

Collections of items or linear collection of things
Arrays are mutable then can be changes.

Create Array

let heroes = ["ironman", "hulk", "thor", "batman"];
let marks = [96, 75, 48, 83, 66];
let info = ["rahul", 86, "Delhi"]; mixed array

Array Indices

arr[0], arr[1], arr[2] .

Array Methods and properties

Array methods(some methods are mutable and some methods are immutable)

- length return length of an array **ex: array.length**
  - **array[0].length** rethrn lenth of first element
  - **array[0][0]** return first value of first element
- Push( ) : add to end
- Pop( ) : delete from end & return
- toString( ) : converts array to string
- Concat( ) : joins multiple arrays & returns result = array.concat(array2);
- Unshift( ) : add to start
- Shift( ) : delete from start & return
- Slice( ) : returns a piece of the array and doesn't change original
- Slice(startIdx, endIdx)
- Splice( ) : change original array (add, remove, replace)
  - Removes, replaces, add elements in place splice(start,dltcount,item0…itemn)
- Splice(startIdx, delCount, newEl1...
- Indexof() return index of value
  - If it return -1 it means the value is not found
- Includes() searched for a value
  - Return true if value is found otherwise false if value is not found
- reverse() reverse an array
- sort() sort an array
  - It doen's work well for numbers
  - It internally converts array into string format and sorting is done based on string format.

Array references : address in memory

Let array=[1,2,3,4]
Array is called here reference variable> it stores address of value

[1]==[1] both have difference memory location so it compare the memory address since they are different so it return false
[1]===[1] both have difference memory location so it compare the memory address since they are different so it return false

```
let array1=[66,65]
let array2=array1;
document.write(array1,"<br>"); 66,65
document.write(array2,"<br>"); 66,65
document.write(array1==array2);
true because both arrays stores same reference so when we compare both arrays then it returns true;
```

Constant Array

const array=[1,2,3,4] you can make changes in the value of array but you can't make new array of the same array.

array=[1,2,3,4,5] > this is new array with same name(reference) here error will occur because are trying to change the reference of array.

*Note: reference is constant not its values and you can't change its reference as well.
Array=array2; error will occur here

Nested Array

Array of Arrays

```
const array1=[[66,65],[77,78]];

console.log(array1[0][0]) first[0] is row and second [0] is column
```

CHAPTER NO 5

Functions in JS

Block of code that performs a specific task, can be invoked whenever needed.

important for interviews (what are the higher order functions/methods)

JavaScript Higher-Order Functions are functions that can accept other functions as arguments, return functions, or both.

Function Definition

```
function functionName( ) {
//do some work
}
```

Function Call

```
functionName( );
```

Function Parameters

function functionName(param1, param2 ...) {

//do some work

}

Default Parameters

Giving a default value to the arguments

Function func(a,b=2){

//do something

}

function sum(a,b=3) {

return a+b;

}

let x= sum(4)

console.log(x)

***note: argument keyword stores all the arguments.**

function sum(a,b,c) {

console.log(arguments)

}

spread

expands an iterable into multiple values

function func(...arr){

// do something

}

Ex:

Console.log(...”siraj”) > result : s i r a j

Let array=[1,2,3,4,5]

Find min value in array

Console.log(Math.min(...array)) > result: 1

Find max value in array

Console.log(Math.max(...array)) > result: 5

Spread array with array literal

let array=[1,2,3]

let array2=[...array]

let char=[...'hellow']

```
console.log(char)
```

spread with object literals

```
let student={
    name:"siraj",
    caste:"qureshi"
}

let mine={
    ...student,
     Id:1234
};
```

Convert array into object

Let array=[1,2,3,4,5]

Let object={...array} // converters array into object and index will be the key and element of particular index will be the value

Reset

Alows a function to take an indefinite number of arguments and bundle them in an array

Function sum(...args){

Return args.reduce((add,el)=> add + el;

}

Spread: single value> multiple value

Rest : multiple value > single value

Example:

```
function sum(...args){
        return args.reduce((sum,el)=>
            sum+el
        )
    }
```

Take multiple inputs and provide a single value

Higher Order function

A function that does one or both:

>takes one or multiple functions as arguments

```
function mulgreet(fun,n) {
        for(let i=1;i<=n;i++){ fun();}
}
        let greet=function(){console.log("hello")}
        mulgreet(greet,8);
```

>return a function

```
function oddevenTest(request) {
            if(request=='odd'){
                let odd=function(n){ console.log(n%2!=0) } return odd;
            }
            else if(request=='even'){
                let even=function(n){ console.log(n%2==0) }return even;
            }
            else{console.log('invalid request') }
        }
       let test=oddevenTest("even");
```

In this way you can also write.

```
function oddevenTest(request) {
```

```
        if(request=='odd'){
            return function(n){ console.log(n%2!=0) }
        }
        else if(request=='even'){
            return function(n){ console.log(n%2==0) }
        }
        else{console.log('invalid request')}
    }
    let test=oddevenTest("even");
```

Arrow Functions

Compact way of writing a function

```
const functionName = ( param1, param2 ...) => {
//do some work
}
```

Implicit return=(automatic return)

```
const mul= (a,b) => (
        a*b      it will return a*b value automatically
)
const mul= (a,b) =>
        a*b      it will return a*b value automatically
```

Methods

Actions that can be performed on an object

```
const calculator={
        add:function(a,b){return a+b;}
    }
    console.log(calculator.add(1,2))
```

Or you can write in this way (shorthand method)

```
const calculator={
        add(a,b){return a+b;}
    }     console.log(calculator.add(1,2))
```

forEach Loop in Arrays

arr.forEach(callBackFunction)

CallbackFunction : Here, it is a function to execute for each element in the array

***A callback is a function passed as an argument to another function.**

```
arr.forEach( ( val ) => {
 console.log(val);
 }
```

We can also use for for each loop for array of objects.

Some More Array Methods

Map

Creates a new array with the results of some operation. The value its callback returns are used to form new array

```
arr.map( callbackFnx( value, index, array ) )
let newArr = arr.map( ( val ) => {
return val * 2;
} )
```

Filter

Creates a new array of elements that give true for a condition/filter.

Eg: all even elements

```
let newArr = arr.filter( ( ( val ) => {
```

```
 return val % 2 === 0;
 } )
```

Reduce

Performs some operations & reduces the array to a single value. It returns that single value.

```
let arr=[1,2,3,4,5];
    let sum=arr.reduce((result,current) => {
      return result+current;
     });
```

Some method

Returns true if condition is true for some element otherwise false.

```
arr=[1,2,3,4,5];
    let new_array=arr.some ((el) => {
      return el%2==0;
     });
```

Resule is true

Every method

Returnes true if condition is true for every element otherwise false.

```
arr=[1,2,3,4,5];
    let new_array=arr.some ((el) => {
      return el%2==0;
     });
```

Result is false

Destructuring of array

Storing valus of array into multiple variables.

```
Let names=[‘siraj’,’dujana’]
Let s=names[0]
Let d=names[0]
```

Use this instead

Let [s,d]=names;

Console.log(s,d) result> siraj dujana

Destructuring of objects

Const student={

Name:’siraj dujana’,

Class:9,

Age:14,

Subjects:[‘hindhi’,’english’],

Username:’sd123’,

Password:1234

}

Let username=student.username

Let password=student.password;

Use instead:

Const {username:user,password:pass}=student;

Usernameis stored in user variable

Here key must be same and user is variable

Console.log(user)// sd123

Hoisting in javascript

Hoisting is JavaScript's default behavior of moving declarations to the top of the current scope (script or function).

It is handled by javascript engine.

It means you can use variables and functions before they are declared.

Example 1: Variable Hoisting

console.log(name); // Output: undefined

var name = "John";

Here, `name` is **hoisted**, but only the declaration (`var name`), not the assignment (`= "John"`).

So, it prints `undefined` instead of an error.

Example 2: Function Hoisting

greet(); // Output: Hello!

function greet() {

  console.log("Hello!");

}

In this case, the entire function is hoisted, so it can be called before its declaration.

Let and Const Hoisting

`let` and `const` are also hoisted, but they are not initialized. Accessing them before declaration results in a Reference Error.

console.log(age); // ReferenceError

let age = 25;

set Timeout function (window object function)

is used when we want to execute piece of code after a specified time.

Example: When I click on button it displays I am clicked after 2 seconds.

setTimeout(function,timeout)

function: here function is call back function

call back function: is a function passed as argument in another function.

Timeout: specified time

```
setTimeout(() => {
            console.log('display after 2 seconds')
        }, 2000);
```

Setinterval function

Same as settimeout function but there is a difference between both and that it executes a function after specified time and repeat it till it is not stopped.

If we call setinterval function it return id and on each call it has new id.

Clearinterval(id): to stop the setinterval function

CHAPTER NO 6

Starter Code

<style> tag connects HTML with CSS

<script> tag connects HTML with JS

Window Object

The window object represents an open window in a browser. It is browser's object (not JavaScript's)

& is automatically created by browser.

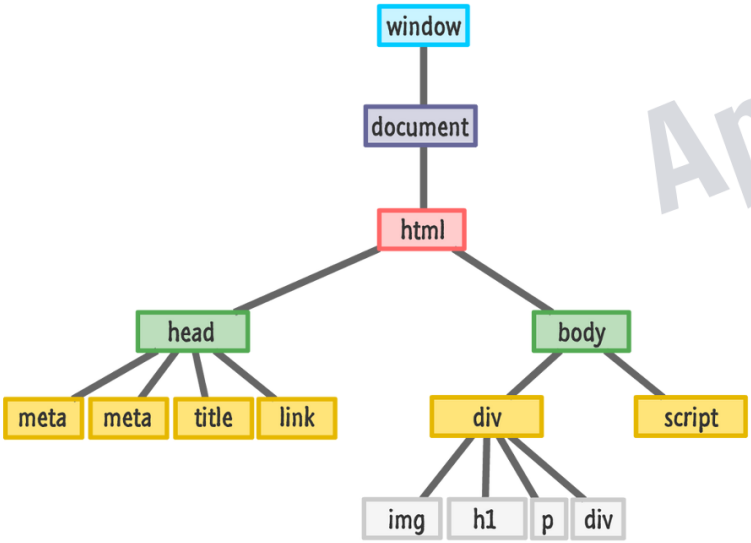
It is a global object with lots of properties & methods.

Like alert(), write() etc

What is DOM?

When a web page is loaded, the browser creates a Document Object Model (DOM) of the page.

- With DOM we can access the elements of HTML in JavaScript.
- We can change content dynamically
- We can apply style dynamically
- We can make our website interactive



DOM Manipulation

Selecting with id

document.getElementById("myId")

Selecting with class

document.getElementsByClassName("myClass")

Selecting with tag

document.getElementsByTagName("p")

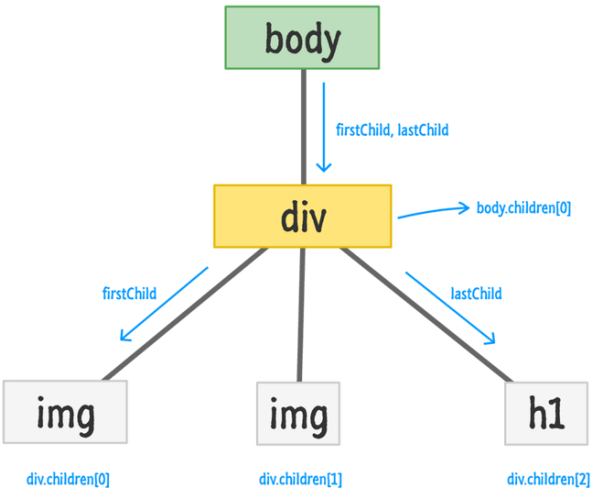
Query Selector

document.querySelector("#myId / .myClass / tag")
//returns first element

document.querySelectorAll("#myId / .myClass / tag")
//returns a NodeList

Properties

- tagName : returns tag for element nodes
- innerText : returns the text content of the element and all its children
  shows the visible text contained in a node
  take text content from web browser screen
- innerHTML : returns the plain text or HTML contents in the element
  shows the full markup
- textContent : returns textual content even for hidden elements
  shows all the full text of visible element
  takes text content from HTML File



Attributes

getAttribute(attr) //to get the attribute value

setAttribute(attr, value) //to set the attribute value

Style

node.style

Inline styling

*note: you can’t set or access the style you have applied on CSS file.

Class list

Obj.classList

- classList.add() to add new classes
- classList.remove() to remove classes
- classList.contains() to check if class exists
- classList.toogle() to toggle between add and remove
  it works like a switch (press on, press off)
  if class exists then it removes and if class doesn’t exist then it adds

Navigation

To navigate from one element to another element

Ex:

<h1></h1>

Div is parent and h1 and p are children (h1 and p are siblings)

Navigate from p to h1

Parentelement : to access element parent element of children

childnode.parentelement

Children : to access the children element of parent

Parentnode.children;

pera.children[0] : access the first children element of parent class

PreviousElementSibling : used to access the previous sibling element of children

pera.children[0].previousElementSibling

NextElementSibling : used to access the next sibling element of children

pera.children[0].nextElementSibling

childelementcount: to return the length of children inside a parent element

Insert Elements

2 steps to follow

1. create element

let el = document.createElement("div")

2. add element

node.appendChild(el) //adds el as a child in selected node.

node.append(el) //adds at the end of node (inside)

*note : It is also used for string to append text

node.prepend(el) //adds at the start of node (inside)

node.before(el) //adds before the node (outside)

node.after(el) //adds after the node (outside)

insertAdjacent(where,element) : we can define that where we have to add the element

beforebegin: before the target element itself

afterbegin: just inside the targetelement, before its first child.

beforeend: just inside the target element, after its last child

afterend: after the target element itself

Delete Element

node.remove( ) //removes the node

CHAPTER NO 7

Events in JS

The change in the state of an object is known as an Event

Events are fired to notify code of "interesting changes" that may affect code execution.

- Mouse events (click, double click etc.)
- Keyboard events (keypress, keyup, keydown)
- Form events (submit etc.)
- Print event & many more

```
node.event = ( ) => {
//handle here
}
```

example

```
btn.onclick = ( ) => {
console.log("btn was clicked");
}
```

It is a special object that has details about the event.

All event handlers have access to the Event Object's properties and methods.

```
node.event = (e) => {
//handle here
}
```

e.target, e.type, e.clientX, e.clientY

Event Listeners

```
node.addEventListener( event, callback )
```

node.removeEventListener(event, callback)

*Note : the callback reference should be same to remove

CHAPTER NO 8

Classes & Objects

object is an entity with state and behavior (properties and methods)

```
const student={
    fullname:"siraj Ahmed",
    marks:90,
    print:function(){
        console.log('marks',student.marks);//student.marks},
};
```

Student.fullname;

Prototypes in JS

A javaScript object is an entity having state and behavior (properties and method).

JS objects have a special property called prototype.

We can set prototype using _ _ proto _ _

*If object & prototype have same method, object's method will be used.

```
const employee={
    calTax(){
        console.log("10%");
    },
};
const siraj={
    salary:1200,
};
```

siraj.__proto__=employee;

Classes in JS

Class is a program-code template for creating objects.

Those objects will have some state (variables) & some behaviour (functions) inside it.

```
class MyClass {
constructor( ) { ... }
myMethod( ) { ... }
}
let myObj = new MyClass( ) ;
```

Constructor( ) method is :

- automatically invoked by new
- initializes object

```
class MyClass {
constructor( ) { ... }
myMethod( ) { ... }
```

}

Inheritance in JS

inheritance is passing down properties & methods from parent class to child class.

class Parent {

}

class Child extends Parent {

}

***If Child & Parent have same method, child's method will be used. [Method Overriding]**

super Keyword

The super keyword is used to call the constructor of its parent class to access the parent's properties and methods.

super(args) // calls Parent's constructor

super.parentMethod(args)

this keyword

This keyword refers to an object that is executing the current piece of code.

It is used to point out current object.

```
const student={
            name:"siraj",
            age:22,
            eng:50,
            math:80,
            physics:30,
            avg(){
                let average=(this.eng+this.math+this.physics)/3
                console.log(average)
            }
        }
```

Error Handling

try-catch

the try statement allow you to define a block of code to be tested for errors while it is being executed.

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

We define try-catch block where there is a probability of occurrence of an error

Like APIs errors handling

try {

... normal code

Console.log(a)

} catch (err) { //err is error object

... handling error

Console.log("var a is not defined");

}

```
try{
            console.log(a) //error occurs
        }
        catch(e){
            console.log('variable a is not declared') this block will be executed
        }

        console.log("siraj") this statement will be executed too
```

```
        console.log(a) error occurs here and the other statements will not be executed

        console.log("siraj")
```

Object Literal:

A shorthand syntax to define an object using curly braces {} with key-value pairs.

Ex: const obj = { key: 'value' };

Object: A collection of key-value pairs in JavaScript.

It can be created using either the Object constructor or an **object literal.**

Creating object literal

student={

name:"siraj",

```
    age:19,
    colors:["black","white"]
}
```

Get values:

```
console.log(student["name"]);
console.log(student["age"]);
console.log(student.name);
console.log(student.age);
```

*Notes: JS automatically converts objects keys to strings.

Even if we made the number as a key, the number will be converted to string.

Example:

```
Const random={
1:
2:
Null:
True:
Undefined:
}
```

1,2,Null,True,Undefined these keys are converted into string automatically.

Add, Update, delete Values

```
student={
    name:"siraj",
```

```
    age:19,
    colors:["black","white"]
  }
```

Student.name=”dujana” update the value

Student.id=012345 to add new pair value in the student object

Delete student.age to delete new pair value from student object

Objects of Objects

Storing information of multiple students

Ex:

```
Const classinfo=   {
   Siraj:{
     name:"siraj",
     age:19
  },
   Mubeeen:{
     name:"Mubeen ",
     age:20
  },
            };
```

Get value

```
console.log(classinfo.siraj.age)
```

Arrays of objects

Storing information of multiple students in array

```
Const classinfo=[
```

```
{
   name:"siraj",
   age:19
},
{
   name:"Mubeen ",
   age:20
},
            ];
```

Get value

Console.log(classinfo[0].name) // siraj will be printed on console screen

Update value

classinfo[0].name=”dujana” // value is being updates from siraj to dujana;

create new key

classinfo[0].gender=”Male” // new key value will be added to classinfo[0] object

Math object

Properties Math.pi= 3.14159 is constant and Math.E = 2.71828 is constant

Methods

Math.abs(n) it is absolute methods and return positive number always

Math.pow(a,b) return a^b > a is base and b is power

Math.floor(n) return rounded number(round off the number <=n)

Ex: Math.floor(5.9), Math.floor(5.5), Math.floor(5.6) return 5

Math.ceil(n) return rounded number(round off the number >=n

Ex: Math.Ceil(5.1), Math.Ceil(5.9), Math.Ceil(5.5) return 6

Math.random() //generate and return random numbers

0-1 but 1is not included

Return 0.10, 0.13, 0.81 but not one

```
Math.floor(Math.random()*100)+1; 1-100
Math.floor(Math.random()*5)+1; 1-5
20+Math.floor(Math.random()*5)+1 20-25
```

CHAPTER NO 9

What this chapter is about?

async await >> promise chains >> callback hell

Sync in JS

Synchronous

Synchronous means the code runs in a particular sequence of instructions given in the program.

Each instruction waits for the previous instruction to complete its execution.

Asynchronous

Due to synchronous programming, sometimes imp instructions get

blocked due to some previous instructions, which causes a delay in the UI.

Asynchronous code execution allows to execute next instructions

immediately and doesn't block the flow.

Callbacks

A callback is a function passed as an argument to another function.

```
setTimeout(()=>{
        console.log("Timeout");
    },6000);
```

Callback Hell

Callback Hell : Nested callbacks stacked below one another forming a pyramid structure.

(Pyramid of Doom)

This style of programming becomes difficult to understand & manage.

```
function data(data,nextdata) {
    setTimeout(() => {
      if(nextdata){
        nextdata();
      }
```

```
    console.log(data);
  }, 2000);
}



data(1,()=>{
  data(2,()=>{
    data(3)
  });
})
```

Promises

Promise is for “eventual” completion of task. It is an object in JS.

It is a solution to callback hell.

let promise = new Promise((resolve, reject) => { .... })

Function with 2 handlers

*resolve & reject are callbacks provided by JS

A JavaScript Promise object can be:(3 states)

Pending : the result is undefined

Resolved : the result is a value (fulfilled)

Rejected : the result is an error object

***Promise has state (pending, fulfilled) & some result (result for resolve & error for reject).**

resolve(result)

reject(error)

.then( ) & .catch( )

promise.then((res) => { .... })

promise.catch((err)) => { .... })

Async-Await

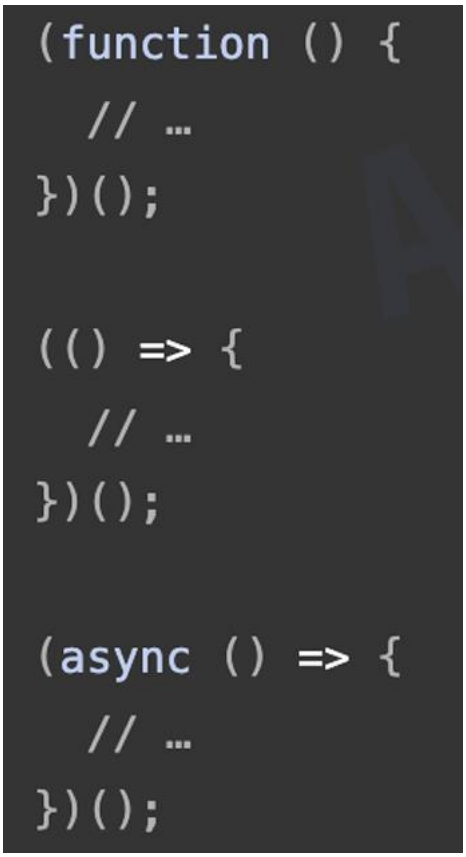
async function always returns a promise.

async function myFunc( ) { .... }

await pauses the execution of its surrounding async function until the promise is settled.

IIFE : Immediately Invoked Function Expression

IIFE is a function that is called immediately as soon as it is defined



JAVASCRIPT FETCH API

The Fetch API interface allows web browser to make HTTP requests to web servers.

No need for XMLHttpRequest anymore.

provides an interface for fetching(sending/receiving) resources.

it uses request and respose objects

the fetch() method is used to fetch a resource (data)

let promise=fetch(url,[option])

each api has its own documentation to use api

A Fetch API Example

The example below fetches a file and displays the content:

Example:

```
let file = "fetch_info.txt"

fetch(file)

.then(x => x.text())

.then(y => myDisplay(y));
```

This code does the following:

1. Fetch the file: It requests the file "fetch_info.txt" from the server.

2. Get the text: Once the file is fetched, it converts the response to text.

3. Display the text: Finally, it takes that text and inserts it into an HTML element with the ID "demo."

So, it effectively shows the contents of "fetch_info.txt" on the webpage.

Since Fetch is based on async and await, the example above might be easier to understand like this:

Use understandable names:

Example

```
async function getText(file) {
  let myObject = await fetch(file);
  let myText = await myObject.text();
  myDisplay(myText);
```

}

Understanding terms

Ajax is async js and XML

json is javascript object notation

json() async method: returns a second promise that resolves with the result of parsing the response body text as json. (input is json, output is js object)

```
let pera=document.querySelector("#demo");
        const getfacts=async()=>{
            let response=await fetch(url);
            //fetch method returns promise
            //await will pause the execution until api is fetched
            console.log(response);//json format

            //we have to convert it into javasccript object
            //we get usable data from response
            let data=await response.json();
            // console.log(data);
            // console.log(data[4].text);
            data.forEach(element => {
                pera.innerHTML+=element.text+"<br>";//

            });

        }
```

This JavaScript code fetches data from an API (specified by `url`) and displays it in the console.

1. `pera` references an HTML element with the ID `demo`.
2. The `getfacts` function uses `fetch` to request data from `url`.
3. `await fetch(url)` pauses execution until the data is fetched.
4. `response.json()` converts the JSON response into a JavaScript object.

Request and response

HTTP Verbs/methods: Define the type of request sent to a server

GET: Retrieves data

POST: Submits data to the server

DELETE: Removes specified resources

PATCH: Updates/modifies resources

HTTP Response Status Codes

200: OK - Success

201: Created - Resource made

400: Bad Request - Invalid input

401: Unauthorized - Auth needed

403: Forbidden - Access denied

404: Not Found - Resource missing

500: Internal Server Error - Server issue

HTTP response headers also contain details about the responses, such as content types, HTTP status code etc.

Introduction to web APIs

What is even more exciting however is the functionality built on top of the client-side JavaScript language. So-called **Application Programming Interfaces (APIs)** provide you with extra superpowers to use in your JavaScript code.

Application Programming Interfaces (APIs) are constructs made available in programming languages to allow developers to create complex functionality more easily. They abstract more complex code away from you, providing some easier syntax to use in its place.

They generally fall into two categories.

Browser APIs are built into your web browser, and are able to expose data from the surrounding computer environment, or do useful complex things. For example: DOM(document object model)

Third party APIs are not built into the browser by default, and you generally have to grab their code and information from somewhere on the Web. For example: Google Maps

What is a Blob in JavaScript?

Blob (Binary Large Object) is a data type to handle binary data in web applications, like images, videos, or other file types. Think of it like a container for raw data that you can use in your code.

Immutable: Once created, the data in a Blob cannot be changed. You can create new Blobs based on existing ones, but you can't modify the original Blob.

Syntax:

```
const myBlob = new Blob(['Hello, world!'], { type: 'text/plain' });
```

Here, 'Hello, world!' is the data, and { type: 'text/plain' } indicates that it's plain text.

You can use Blobs to handle files, such as uploading an image or saving data.
You can convert a Blob into a URL that can be used in your webpage. For example

```
const url = URL.createObjectURL(myBlob);
```

This creates a temporary URL for the Blob that you can use in an <img> tag or for downloading.

Reading Blobs: To read the contents of a Blob, you can use the FileReader API:

```
const reader = new FileReader();

reader.onload = function(event) {

    console.log(event.target.result); // This will log 'Hello, world!'

};

reader.readAsText(myBlob); // Reads the Blob as text
```

Why Use Blobs?

1. **Dynamic Creation:** Blobs allow you to create image, audio, or video data on the fly, especially from user inputs or generated content.

2**. File Handling:** They manage file uploads as Blob objects, enabling easy previews and processing.

3. **Binary Data:** Blobs can handle any binary data format, not just standard media types.

4. **Efficiency:** They enable chunked data processing, which is useful for large files without consuming too much memory.

Type script

What is TypeScript?

TypeScript is a **programming language** that builds on JavaScript by adding **static types**. It helps catch errors during development and makes your code easier to read and maintain.

It is developed by Microsoft company.

Difference between TypeScript and JavaScript:

JavaScript	TypeScript
Dynamic typing (no type checks)	Static typing (type checks during development)
Doesn’t need compilation	Needs to be compiled to JavaScript
More prone to runtime errors	Fewer runtime errors due to type checking
Flexible and easier to start with	Safer and better for larger projects

Javascript:

```
let message = "Hello";

message = 123; // No error
```

data type can be changed(dynamic typing)

TypeScript:

```
let message: string = "Hello";

message = 123; // Error: Type 'number' is not assignable to type 'string'
```

data type cant be changed(static typing)

TypeScript is like JavaScript with extra safety checks!