MODULE 9 : ADVANCE JAVASCRIPT → INTRODUCTION TO JAVASCRIPT (03)

9.1 Introduction to JavaScript, Javascript Features, Ways to write and Add JS code

9.2 Comments in JS, Variables in JS,

9.3 DataTypes - Primitive, Non Primitive

9.4 Operators - Arithmetic, Comparision, Bitwise, Logical, Assignment, Special JS operators

9.5 JS if/else, switch, do/while, for/in,

9.6 JS functions, function call, bind v/s apply

MODULE 10 : ADVANCE JAVASCRIPT → JAVASCRIPT OBJECT (04)

10.1 JS objects, JS Object methods, JS Arrays, JS array methods

10.2 JS string, JS string methods

10.3 JS numbers, and number constants, Number methods

10.4 JS data Objects, JS math Objects

10.1 JS objects, JS Object methods, JS Arrays, JS array methods

JS objects

- In JavaScript, almost "everything" is an object.
  - Booleans can be objects (if defined with the new keyword)
  - Numbers can be objects (if defined with the new keyword)
  - Strings can be objects (if defined with the new keyword)
  - Dates are always objects
  - Maths are always objects
  - Regular expressions are always objects
  - Arrays are always objects
  - Functions are always objects
  - Objects are always objects

JS Object methods

```
<script>
const person = {
  firstName: "Chirag",
  lastName: "Mehta",
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
};
document.write(person.fullName());
</script>
```

JS Arrays

Without Array

- let s1="chirag";
- let s2="dhiren";
- let s3="parth";

With Array

- let s=["chirag","dhiren","parth"];
- document.write(s);

- let s=[];
- s[0]="chirag";
- s[1]="dhiren";
- s[2]="parth";
- document.write(s);

new keyword

- const s=new Array("Chirag","dhiren","parth");
- document.write(s);

JS array methods

- length
- sort()
- push()
- pop()
- shift() //remove first element
- unshift() // add elements at first
- concat()

10.2 JS string, JS string methods

JS string

- let s=“Chirag”;
- let s1=new String(“Chirag”);

JS string methods

- Length
- substr()
- replace()
- toUpperCase()
- toLowerCase()
- concat()
- trim()
- trimStart()
- trimEnd()
- padStart()
- padEnd()
- charAt()
- charCodeAt()
- split()

10.3 JS numbers, and number constants, Number methods

JS numbers

- Number() converts a value to a number if possible:
- If the value cannot be converted, NaN is returned.
- Number(999);
- Number("999");
- Number("999 888");

number constants

- const keyword is used to declare constant.
- Must have to assign value when we declare a const.
- const pi=3.14;

10.4 JS date Objects, JS math Objects

- <script>
- const d = new Date();
- document.write(d);
- </script>

Different ways to create date

```
new Date()
new Date(date string)

new Date(year,month)
new Date(year,month,day)
new Date(year,month,day,hours)
new Date(year,month,day,hours,minutes)
new Date(year,month,day,hours,minutes,seconds)
new Date(year,month,day,hours,minutes,seconds,ms)

new Date(milliseconds)
```

Method	Description
getFullYear()	Get **year** as a four digit number (yyyy)
getMonth()	Get **month** as a number (0-11)
getDate()	Get **day** as a number (1-31)
getDay()	Get **weekday** as a number (0-6)
getHours()	Get **hour** (0-23)
getMinutes()	Get **minute** (0-59)
getSeconds()	Get **second** (0-59)
getMilliseconds()	Get **millisecond** (0-999)
getTime()	Get **time** (milliseconds since January 1, 1970)

JS math Objects

Math.round(x)	Returns x rounded to its nearest integer
Math.ceil(x)	Returns x rounded up to its nearest integer
Math.floor(x)	Returns x rounded down to its nearest integer
Math.trunc(x)	Returns the integer part of x (new in ES6)

```
Math.E        // returns Euler's number
Math.PI       // returns PI
Math.SQRT2    // returns the square root of 2
Math.SQRT1_2  // returns the square root of 1/2
Math.LN2      // returns the natural logarithm of 2
Math.LN10     // returns the natural logarithm of 10
Math.LOG2E    // returns base 2 logarithm of E
Math.LOG10E   // returns base 10 logarithm of E
```

MODULE 11 : ADVANCE JAVASCRIPT → JAVASCRIPT BOM & DOM (04)

11.1 BOM - Window object, history object, Navigtator Object, screen Object,

11.2 DOM, Model, Methods

11.3 JS Validations, Form Validations, Email Validations, OOP

11.1 BOM - Window object, history object, Navigtator Object, screen Object,

BOM - Window object

- window.innerHeight - the inner height of the browser window (in pixels)
- window.innerWidth - the inner width of the browser window (in pixels)

history object

- history.back() - same as clicking back in the browser
- history.forward() - same as clicking forward in the browser

Navigator Object

- navigator.cookieEnabled
- navigator.appName
- navigator.platform

screen Object

- screen.width
- screen.height
- screen.availWidth
- screen.availHeight
- screen.colorDepth (The colorDepth property returns the depth in bits per pixel.)
- screen.pixelDepth

11.2 DOM - Model, Methods

Methods

- getElementById
- getElementsByName
- innerHTML

11.3 JS Validations, Form Validations, Email Validations, OOP

MODULE 12 : JAVASCRIPT ES6 OBJECT ORIENTED CONCEPTS (06)

12.1 JS classes

12.2 JS Prototypes

12.3 Construtors

12.4 Static Methods

12.5 OOP Encapsulation

12.6 Inheritance

12.7 Polymorphism

12.8 Abstraction

12.1 JS classes

- ECMAScript 2015, also known as ES6, introduced JavaScript Classes.
- JavaScript Classes are templates for JavaScript Objects

12.2 JS Prototypes

```
class ClassName
{
  constructor() { ... }
}
```

12.3 Constructors

- Special method
- It has the same name as constructor()
- Executed automatically when we create object of class.
- If you do not define a constructor method, JavaScript will add an empty constructor method.

Example:

```
class demo
  {
      constructor()
      {
          console.log("Constructor Called");
      }
  }
  const d=new demo();
```

12.4 Static Methods

Without Static

```
class demo
   {
       constructor()
       {
           console.log("Constructor Called");
       }
       add(a,b)
       {
           return a+b;
       }
   }
   const d=new demo();
   console.log(d.add(8,7));
```

With static

```
class demo
{
    static add(a,b)
    {
        return a+b;
    }
}
console.log(demo.add(8,7));
```

12.5 OOP Encapsulation

- The meaning of **Encapsulation**, is to make sure that "sensitive" data is hidden from users.
- Use var keyword to make data members private.
- Use setter methods to set the data and getter methods to get that data.

```
class demo
    {
        constructor()
        {
            var name;
            var per;
        }
       getName()
       {
            return this.name;
       }
       setName(name)
       {
        this.name=name;
       }
    }
    var d= new demo();
    d.setName("Chirag");
    console.log(d.getName());
```

12.6 Inheritance

- Acquiring property and methods from one class to another class is known as Inheritance.
- extends keyword is used to perform inheritance in javascript.

```
<script>
    class parent{
        square(a){
            return a*a;
        }

    }
    class child extends parent{
        constructor(){
            super();
            console.log(this.square(5));
        }
    }
    const c=new child();
</script>
```

12.7 Polymorphism

```
 class car{
        avg(){
            console.log("car Avg")
        }
    }
    class hyundai extends car{
        avg(){
            console.log("Hundai Avg")
        }
    }
    class tata extends car{
        avg(){
            console.log("tata Avg")
        }
    }
const c=new car();
const h=new hyundai();
const t=new tata();
c.avg();
h.avg();
t.avg();
```

MODULE 13 : ADVANCE JAVASCRIPT (10)

13.1 JS cookies, Cookie Attributes, Multiple Names, deleting a cookie, examples

13.2 JS events, ADDevent listner, Click Events, Double Click event, onload event, OnResize event,

13.3 Exceptional Handling, JS Throw statement

13.4 Promise in JS, Promises Chaining, Error Handling with Promises, Promise API, Async wait in JS

13.5 JS Async - Callbacks - Promises - Async/Await

13.6 JS Typed Array Methods, SETS in JS, JS Map, Weakset, WeakMap,

13.7 JS Callbacks, Closures, Date Difference, Date Formats, Date PArse, Defer, Redirect

13.8 Scope, scroll, Sleep, Void, Form events, Type conversions

JS cookies

- Cookies are data, stored in small text files, on your computer.
- You can store up to 20+ cookies in a web browser.
- It is dependent on web browser.

Cookie Attributes

Attributes	Description
expires	It maintains the state of a cookie up to the specified date and time.
max-age	It maintains the state of a cookie up to the specified time. Here, time is given in seconds.
path	It expands the scope of the cookie to all the pages of a website.
domain	It is used to specify the domain for which the cookie is valid.

```
document.cookie="fname=chirag";
    document.cookie="lname=mehta";
    console.log(document.cookie);
```

deleting a cookie

- If we don't specify a value then it will delete a cookie.

examples

- document.cookie="uname=;expires=thu,23 feb 2028";
- console.log(document.cookie);

JS events

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

AddEventListner

```
<script>
    document.addEventListener("click",myFun);
    function myFun()
    {
        alert("Clicked")
    }
</script>
```

Click Events

```
<script>
    window.addEventListener("click",myFun);
    function myFun()
    {
        alert("Clicked")
    }
</script>
```

Double Click event

```
<script>
    window.addEventListener("dblclick",myFu
n);
    function myFun()
    {
        alert("Clicked")
    }
</script>
```

onload event

```
<script>
    window.addEventListener("load",myFun);
    function myFun()
    {
        alert("Clicked")
    }
</script>
```

OnResize event

```
<script>
    window.addEventListener("resize",myFun)
;
    function myFun()
    {
        alert("Clicked")
    }
</script>
```

```
<script>
    window.addEventListener("click",()=>{
        alert("Clicked")
    });
</script>
```

13.3 Exceptional Handling, JS Throw statement

What is Exception?

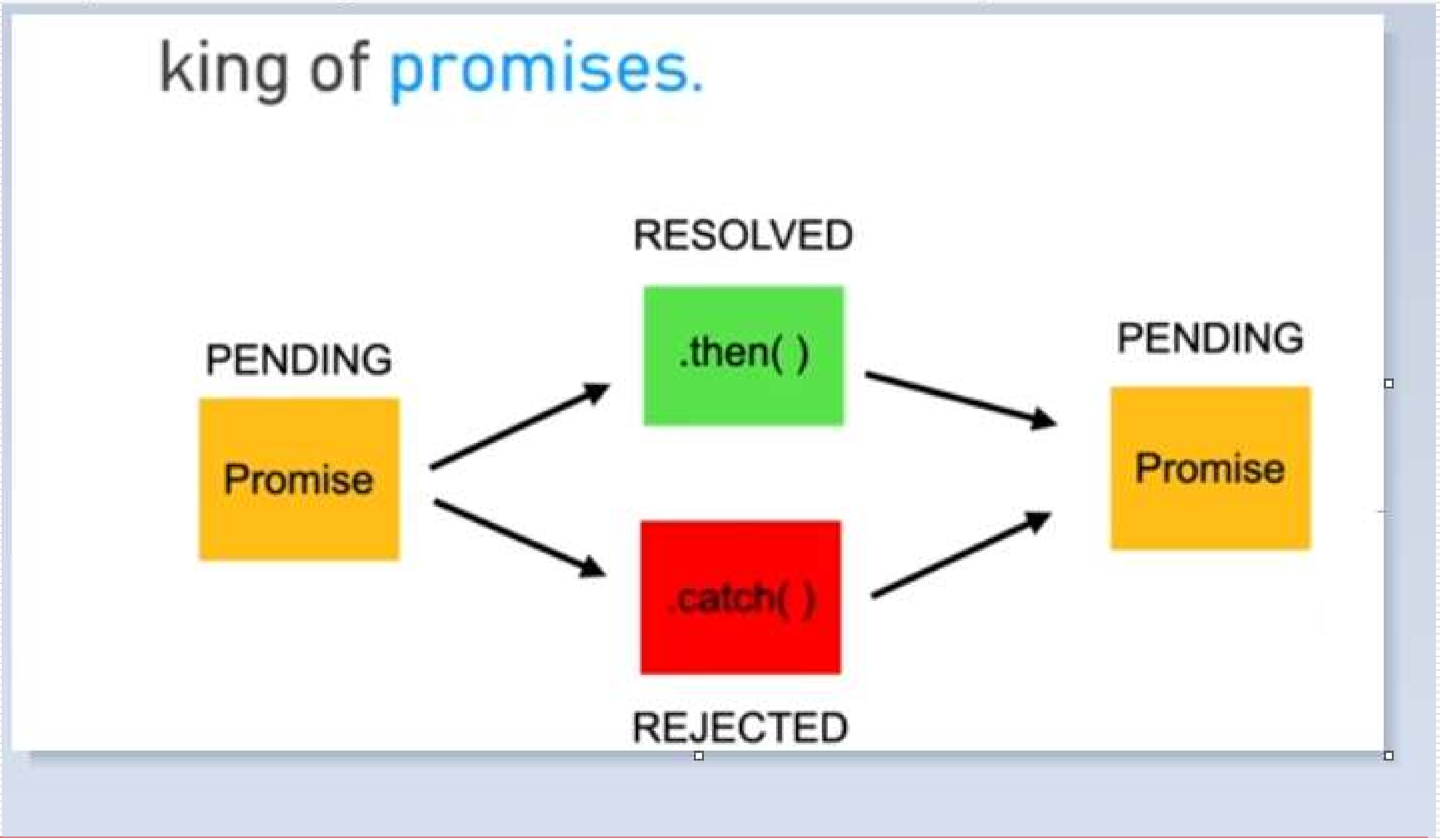
- Types of Error:
  - Syntax Error
  - Runtime Error
  - Logical Error

```
try{
    const a=10;
    let b=a/0;
    console.log(b)
}
catch(err){
    console.log("Error --> "+err);
}
finally{
    console.log("This will always Executes...");
}
```

```
  <script>
      function valid(){
          var n=document.getElementById("txtn").value;
          try{
              if(n<36) throw "Below Limit ";
              else if(n>100 ) throw "More than Upper Limit"
          }
          catch(Err){
              console.log("Error in Data "+Err);
          }
      }
  </script>
</head>
<body>
  Marks [36-100] : <input type="number" id="txtn">
  <button onclick="valid()">Click Me</button>
</body>
```

13.4 Promise in JS, Promises Chaining, Error Handling with Promises, Promise API, Async wait in JS

Promise in JS



- A JavaScript Promise object can be:
  - Pending
  - Fulfilled
  - Rejected
- The Promise object supports two properties: **state** and **result**.
- While a Promise object is "pending" (working), the result is undefined.
- When a Promise object is "fulfilled", the result is a value.
- When a Promise object is "rejected", the result is an error object.

```
const obj=new Promise((resolve,reject)=>{

    })
```

```
let promise=new Promise(function (resolve,reject){
    let n=10;
    if(n==0)
    resolve("done");
    else
    reject("Problem");
});
promise.then(
    function(value){console.log(value);},
    function(error){console.log(error);}
);
```

```
<script>
const obj=new Promise((resolve,reject)=>{
setTimeout(() => {
    let r_no=[1,2,3,4,5];
    // resolve(r_no);
    reject("Error While Fetching...")
}, 2000);
});
obj.then((r_no)=>{
    console.log(r_no);
}).catch((err)=>{
console.log(err);
})
</script>
```

Error Handling with Promises

Async wait in JS

- *"async and await make promises easier to write"*
- **async** makes a function return a Promise
- **await** makes a function wait for a Promise

```
async  function getData(){
   let prms=new Promise((resolve,reject)=>{
       resolve("Waiting for React...")
   });
   alert(await prms);
   }
   getData();
```

13.5 JS Async - Callbacks - Promises - Async/Await

Async/Await

13.6 JS Typed Array Methods, SETS in JS, JS Map, Weakset, WeakMap,

JS Typed Array Methods

- In Javascript, a typed array is an array-like buffer of binary data.

- There is no JavaScript property or object named TypedArray, but properties and methods can be used with typed array objects:

- const arr=new Int8Array(4);
- console.log(arr);

Object	Data Type	Range
Int8Array	Signed integer (byte)	-128/127
Uint8Array	Unsigned integer (octet)	0/255
Uint8ClampedArray	Unsigned integer (octet)	0/255
Int16Array	Short integer	-32768/32767
Uint16Array	Unsigned short integer	0/65535
Int32Array	Signed long integer	$-2^{31}/2^{31}-1$
Uint32Array	Unsigned long integer	$0/2^{32}$
Float32Array	Float - 7 significant digits	$1.2x10^{-38}/3.4x10^{38}$
Float64Array	Double - 16 significant digits	$5.0x10^{-324}/1.8x10^{308}$
BigInt64Array	Big signed integer	$-2^{63}/2^{63}-1$
BigUint64Array	Big unsigned integer	$0/2^{64}$

- Typed arrays provide a way to handle binary data as efficiently as arrays work in C.
- Typed arrays are raw memory, so JavaScript can pass them directly to any function without converting the data to another representation.
- Typed arrays are faster than normal arrays, for passing data to functions that can use raw binary data (Computer Games, Canvas, File APIs, Media APIs).

SETS in JS

- A set is a collection of items that are unique i.e no element can be repeated. Set in ES6 are ordered: elements of the set can be iterated in the insertion order. Set can store any type of value whether primitive or objects.

- // [“PHP",“ASP",“JAVA",“REACT"]
- var set1 = new Set([“PHP",“ASP",“JAVA",“REACT","anish"]);
-
- // it contains 'f', 'o', 'd'
- var set2 = new Set("fooooooood");
-
- // it contains [10, 20, 30, 40]
- var set3 = new Set([10, 20, 30, 30, 40, 40]);
-
- // it is an empty set
- var set4 = new Set();

- Add()
- Delete()
- Clear()

JS Map

- let map = new Map();

- map.set('1', 'str1');   // a string key
- map.set(1, 'num1');   // a numeric key
- map.set(true, 'bool1'); // a boolean key

- console.log(map.get(1));

Weakset

WeakMap

13.7 JS Callbacks, Closures, Date Difference, Date Formats, Date PArse, Defer, Redirect

Callback function

- A callback is a function called when the task finishes, and a callback function **allows other code to run in the meantime**. Using the Callback concept, Node. js can process many requests without waiting for any function to return the result, making Node. js highly scalable.

```
function add(a,b,callback){
    console.log(a+b);
    callback();
}

function callme()
{
    console.log("Call me function...");
}

add(5,7,callme);
```

```
function printName(){
    console.log("Chirag");
}
function printsname(){
    console.log("Mehta");
}

setTimeout(printName,2000);
// printName();
printsname();
```

```
function printfname(callback){
    console.log("Chirag");
    callback();
}
function printlname(){
    console.log("Mehta");
}
setTimeout(() => {
    printfname(printlname);
}, 5000);
```

Date Difference

```
var d1=new Date("03/01/2023");
var d2=new Date("04/01/2023");
var diff=d2.getTime()-d1.getTime();
var at=diff/(1000*60*60*24);
console.log(at);
```

Date Formats

Type	Example
ISO Date	"2023-02-25" (The International Standard)
Short Date	"03/25/2023"
Long Date	"Mar 25 2023" or "25 Mar 2023"

Date Parse

- parse() parses a date string and returns the time difference since January 1, 1970.
- let ms = Date.parse("March 02, 2023");

Defer

- A script that will be downloaded in parallel to parsing the page, and executed after the page has finished parsing:
- <script src="demo_defer.js" defer></script>
- The defer attribute is a boolean attribute.
- If the defer attribute is set, it specifies that the script is downloaded in parallel to parsing the page, and executed after the page has finished parsing.

Redirect

- location.href
- location.replace()
- The difference between href and replace, is that replace() removes the URL of the current document from the document history, meaning that it is not possible to use the "back" button to navigate back to the original document.

13.8 Scope, scroll, Sleep, Void, Form events, Type conversions

Scope

- Block scope
- Function scope
- Global scope

scroll

```
function scrollWin() {
  window.scrollTo(400, 400);
}
```

Sleep

```
sleep(Time in ms).then(() => {
//// code
})
```

Used with async and await

Void

- void is an important keyword in JavaScript which can be used as a unary operator that appears before its single operand, which may be of any type. This operator specifies an expression to be evaluated without returning a value.

Form events

Type conversions

- Number()
- String()