Javascript-JavaScript is versatile and can be used in both client-side and server-side development

- console.log("hello world"); =>it will print a message to the console
- To declare a variable we can use var, let, const keyword but mostly we will use let and const keyword.
  Var- can be re-declared and updated, global scope variable.
  Let- cannot be re-declared but can be updated, block scope variable.
  Const- cannot be re-declared or updated, block scope variable.
- Object creation=> player is an object here

```
const player={name:'shakib', age:36, highest_score: 136};
console.log(player["highest_score"]);
console.log(player.age);  //two different ways of accessing
```

- Difference between normal string and literal string

```
let player ={name: "shakib", age:36};
output = `the age of ${player.name} is ${player.age}`;
console.log(output);
//console.log("the age of", player.name, "is", player.age);
```

  This is the difference between normal string and literal string. Using a literal string we can print the whole statement in a single string. ${expression}=> String Interpolation

- Different string functions

```
let player ="  Shakib Al Hasan  ";
//output = player.toUpperCase();
//output = player.toLowerCase();
output = player.trim(); //trim() removes whitespaces
output = player.slice(1,7); // slice() slice a string, 1st parameter included and 2nd parameter excluded
console.log(output);
```

- Take input in js

```
let name= prompt("enter your name: "); //prompt() use for taking input
```

- push/pop from an array

```
let arr=["chips","coke","biscuits"];
//arr.push("burger");
//arr.pop();
//arr.unshift("burger"); //unshift() helps to insert an element at the first
//arr2 = arr.shift(); //shift() helps to remove an element from the first and return that element
arr.splice(2,2,101,102);
arr.splice(2,0,101,102);
```

splice() takes multiple parameters.
slice(add,remove,replace[element1,element2……elementN])

```
//console.log(arr);
console.log(arr.toString()); //return a string
```

- Practice problem

```
//create an array to store companies=>
arr=["bloomberg","microsoft","uber","google","IBM","netflix"];
//remove the first company from the array
removedFirstcompany = arr.shift();
//remove uber & Add Ola in its place
arr.splice(2,1,"Ola");
//Add amazon at the end
arr.push("Amazon");
console.log(arr);
```

- Declare function //function functionName(){//do something}

```
function area(r){
  result = 3.14*r*r; //area of a circle
  console.log(result);
}
```

```
area(5);
//primitive representation of declaring a function
function task(x, y) {
  sum = x+y;
  console.log(sum);
}
task(5,3);

//an advance way of declaring a function in javascript named
arrowfunction
const sum = (x,y) =>{
  console.log (x+y);
}
sum(5,3);
```

- forEach loop in arrays:
  arr.forEach(callbackfunction)
  -a callBack is a function passed as an argument to another function. If any method uses a function as a parameter or return a function as a value that method is known as HOF/HOM => (higher order function/ higher order method)

```
let arr = [1,2,3,4,5];
arr.forEach(function printVal(val){
  console.log(val);
}
);
```

#example of arrow function

```
let arr = [1,2,3,4,5];
arr.forEach((val) =>{
  console.log(val);
}
);
```

#uses of map()

```
let arr = [1, 2, 3, 4, 5];
arr.map((val) => { //map returns the value as an array
  console.log(val);
});
```

#uses of filter()
Creates a new array of elements that give true for a condition/filter.

```
let arr = [1, 2, 3, 4, 5, 6, 7];
let evenArr = arr.filter((val) => {
    return val % 2 == 0;
});
console.log(evenArr);
```

- Dom (document object model)
  -a hierarchical tree
  -to change any property dynamically, we use DOM
  Window -> document -> html
  html -> head || body
  head -> meta1,meta2…,title,link
  body -> div || script
  div -> img,h1,p,div

- Dom manipulation

  - ❖ document.getElementByID()
  - ❖ document.getElementsByClassName()
  - ❖ document.getElementsByTagName()
  - ❖ document.querySelector("id/class/tagName") => this will return the 1st element that matches the parameter.
  - ❖ document.querySelectorAll("id/class/tagName") => this will return all the elements/nodelist that match the parameter.
  - ❖ tagName: returns tag for element nodes
  - ❖ innerText: returns the text content of the element and all its children.
  - ❖ innerHTML: returns the plain text or HTML contents in the element.
  - ❖ textContent: return textual content even for hidden elements.

- Dom manipulation Attributes

  - ❖ getAttribute(attr) //to get the attribute value
  - ❖ setAttribute(existing attr, new value) //to set the new attribute value
  - ❖ Node.style //to change the css properties using JS.

  -attribute is something that is used as additional information inside a tag.
   this is a box Here, id="box", id is an attribute and "box" is value of attribute.

- Dom manipulation insert/remove elements
  - To insert an element into node, we have to create the element first
    - Let el = document.createElement("div")
  - ❖ node.append(element) //adds at the end of the node inside
  - ❖ node.prepend(element) //adds at the start of the node inside
  - ❖ node.before(element) //adds at the start of the node outside
  - ❖ node.after(element) //adds at the end of the node outside
  - ❖ node.remove() //removes the node

- Practice question
  -create a new button element. Give it a text “click here”, background color of red and text color of white. Insert the button as the first element inside the body tag.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
          <meta     name="viewport"     content="width=device-width,
initial-scale=1.0">
  <title>Document</title>
</head>
<body>

</body>
</html>

//-----------------------------------------------------------------
-----------

let newBtn = document.createElement("button");
newBtn.innerText = "click here";
newBtn.style.color = "white";
newBtn.style.backgroundColor = "red";

let body = document.querySelector("body");
body.prepend(newBtn);
```

  -create a <p> tag in html, give it a class and some styling. Now, create a new class in css and try to append this class to the <p> element.
  Did you notice how you overwrite the class name when you add a new one? Solve this problem using classList.

- Event handler

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta    name="viewport"    content="width=device-width,
initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <div id="div1"> this is a div </div>
    <button  onclick=  "console.log('you  clicked  the  btn');">
click me </button> //inline event handled here
</body>
</html>

//-------------------------------------------------------------
-------------------------
#div1{
  height: 100px;
  width: 100px;
  background-color: lightgreen;
}
//-------------------------------------------------------------
-------------------------
//proper way to handle event using JS
let div1 = document.querySelector("#div1");
div1.onclick = ()=>{
  console.log("you clicked on div");
}
```

- Event listener
  1. node.addEventListener (event, callback function)
  2. node.removeEventListener (event, callback function)

  Event => click,dblclick,mouseover etc. when an event occurs the callback function will execute.

- Practice question
  Create a toggle button that changes the screen to dark mode when clicked and light mode when clicked again

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
          <meta     name="viewport"     content="width=device-width,
initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <button id="btn"> switch mode </button>
</body>
</html>

//-----------------------------------------------------------------
-----------
let btn = document.querySelector("#btn");
let currentMode = "light";

btn.addEventListener("click",()=>{
  if (currentMode == "light"){
    currentMode = "dark";
    document.querySelector("body").style.backgroundColor="grey";
  }
  else{
    currentMode = "light";
    document.querySelector("body").style.backgroundColor="white";
  }
  console.log(currentMode);
});
```

- Classes in javascript

```
class Car{
  constructor(brand, mileage){
    this.brand = brand;
    this.mileage = mileage;
  }
  start(){
    console.log("car start");
  }
  stop(){
    console.log("car stop");
  }
}
let car1 = new Car("toyota", 2);
console.log(car1.brand + car1.mileage);
```

- Inheritance in javascript

```
class Parent{
  hello(){
    console.log("hello! this is parent class property");
  }
}

class Child extends Parent{
  //child class now have the access of parent class
  //child class can have its own property too
  //but if we want to invoke hello function we dont need to write a
hello function here
}
let objForChildClass = new Child();
objForChildClass.hello();
```

```
class Car{
  constructor(brand, mileage){
    this.brand = brand;
    this.mileage = mileage;
  }
  start(){
    console.log("car start");
  }
  stop(){
    console.log("car stop");
  }
}

class childCar extends Car{
  booster(){
    console.log("boosted");
  }
}
let car1 = new Car("toyota", 2);
console.log(car1.brand + car1.mileage);
let chCar = new childCar ("bmw", 10);
chCar.start()
```

- Sync in Js
  - ❖ Synchronous - means the code runs in a particular sequence of instructions given in the program. Each instruction waits for the previous instructions to complete the execution.

```
console.log("hello1");
console.log("hello2");
console.log("hello3");
```

- ❖ Asynchronous - due to Synchronous programming, sometimes important instructions get blocked due to some previous instructions, which causes a delay in the UI. Asynchronous code execution allows execution of next instructions immediately and doesn't block the flow.

```
console.log("hello1");
console.log("hello2");
//setTimeout (param1, param2);
setTimeout (()=>{
  console.log("hello3");
},4000);
console.log("hello4");
console.log("hello5");
```

- Callback in JS- is a function passed as an argument to another function
  -Async await >> promise chain >> callback hell
  -Async await is used to make asynchronous programs simple.

callback hell: nested callback stacked below one another forming a pyramid structure. Due to callback hell programming becomes difficult to understand and manage.

```
function getData(dataId, getNextData) {
    setTimeout(() => {
        console.log("data", dataId);
        if (getNextData) {
            getNextData();
        }
    },3000);
}

//getData(1, getNextData()) this is not the proper way of calling a callback function
//callback hell
getData(1, () => { //we have to use a arrow func as a result it will not execute the output immediately
    getData(2, () => {
        getData(3, () => {
            getData(4);
        });
    });
});
```

Promises: promise is for eventual completion of task. It is a solution to callback hell.

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

```
function getData(dataId, getNextData){
   return new Promise ((resolve, reject)=>{
    setTimeout(() => {
        console.log("data", dataId);
        resolve("success");
        if (getNextData) {
            getNextData();
        }
    },3000);
   });
}
```

Javascript promise object can be:

1. Pending: the result is undefined
2. Resolved: the result is a value (fulfilled/success)
3. Rejected: the result is an error object

- promise has state (pending, fulfilled) and some result (result for resolve and error for reject)
- promise.then((res)=>{...}) & promise.catch((err)=>{...})

- **Promise chaining:**

```
function asyncFunc1(){
   return new Promise((resolve, reject)=>{
  setTimeout(()=>{
    console.log("data1");
    resolve("success");
  },4000);
   });
}
function asyncFunc2(){
   return new Promise((resolve, reject)=>{
  setTimeout(()=>{
    console.log("data2");
    resolve("success");
  },4000);
   });
}

//chaining
console.log("fetching data1...");
asyncFunc1().then((res)=>{
  console.log("fetching data2...");
  asyncFunc2().then((res)=>{});
});
```

- Better way than chaining- **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. Await keyword used inside the block of async function.

```
function getData(dataId){
   return new Promise ((resolve, reject)=>{
    setTimeout(() => {
        console.log("data", dataId);
        resolve("success");
    },3000);
   });
}

async function getAllData(){
  await getData(1);
  await getData(2);
  await getData(3);
}

console.log(getAllData());
```

- Fetch API- application programming interface
  -it uses request and response object
  -fetch() method is used to fetch a resource(data)
  -let promise = fetch (url, [options]);

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
            <meta   name="viewport"   content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <button id="btn">get texts</button>
    <p id="facts"></p>
    <script src="api.js"></script>
</body>
</html>
//-------------------------------------------------------------------
---------------
```

```
const URL = "https://cat-fact.herokuapp.com/facts";
let btn = document.querySelector("#btn");
let facts = document.querySelector("#facts");

const getFacts = async ()=>{
  console.log("getting data...");
    let response = await fetch(URL);
    console.log(response);
    let data = await response.json();
    facts.innerText = data[1].text;
};
btn.addEventListener("click",getFacts);
```

- Understanding few terms
  -AJAX is asynchronous js and xml
  -json is javascript object notation
  -json() method: returns a second promise that revolves with the result of parsing the response body text as JSON.
- HTTP status code:
  1. 200 => ok/success
  2. 400 => bad request
  3. 404 => not found
  4. 500 => internal server error
  5. 502 => bad gateway