JavaScript.

- **Programming language.**
- **Client-side script language.**
  1. HTML
  2. CSS
  3. JavaScript
- **Server-side script language.**
  1. PHP
  2. Java
- **What is java script.**
- Java script is a scripting language for creating dynamic web page content. It creates elements for improving site visitor's interaction with web page, such as dropdown menus, animated graphics, and dynamic background colors.
- **What is the need of java script.**
- It is used to capture user actions on a web page.
- It gives us flexibility to edit the HTML elements after they have been loaded by the browser.
- **Use of java script files.**
- **Add to the body tags.**
- **<script type="text/javascript"scr="js/script/js"></scritp>**
- **Executing a JavaScript file.**
- Script tag ko encounter karat Hai basically JavaScript file ka liya hota hai ya accosted JavaScript ko fetch kar na ka liya one new http request instate karat Hai one fetch the JavaScript file browser ka pass chala jata hai to browser use render karta hai in short JavaScript files HTML ka sath fetch ni hote hai une render karta waqt fetch karta hai.
- **Print the Hello World Shaan in java script.**
  console.log("hello world Shaan");
- **Print the user input in Java Script.**
  const prompt = require('prompt-sync')();
  var firstname = prompt("enter the first name:");
  var lastname = prompt("enter the last name:");
  console.log("your full name is:",firstname,lastname);

Variable.

- **Variable.**
- A JavaScript variable is simply a name of storage location.
- **There are two types of variable in Javascript.**
  1. Local variable.
  2. Global variable.
- **Defining/ Declaring a variable.**
  1. Let = newly introduction in 2015.
  2. Var= old way of declaring a variable.
- **Let.**
- let keyword is also used for variable declaration.
- **JS code.**
- let a = 10
  let b = 20
  let c = a + b;
  console.log(c);
- **We cannot declare the same variable twice in the code.**
- let abc = 5;
  console.log(abc);
- let abc = "Hello world!";
  console.log(ab);
- **Var.**
- Var keyword is also used for variable declaration.
- **JS code.**
- var a = 10
  var b = 20
  var c = a + b;
  console.log(c);
- **Different between var and let.**

let	var
It has a block level scope.	It has a function or global scope.
Same variable cannot be declared twice.	Same variable can be declared twice.
The variable needs to be initialized.	No need to initialize the variable.

Data Types.

- **Data Types.**
- **There are six types of data types primitives.**
  1. Undefined.
  2. Boolean.
  3. Number.
  4. String.
  5. Bigint.
  6. Symbol.
- **Undefined**: it was not declared, declared but never assigned a value or declared and assigned the value of undefined.
- **JS code**.
  var myVar;
  console.log(myVar);
- **Booleans:** Boolean data type can have only two values: true or false. It is used to represent a logical state or a yes/no answer.
- **JS code.**
  let x = true;
  let y = false;
  console.log(typeof(x,y));
- **Numbers:** JavaScript has only one type of number. Numbers can be written with or without decimals. This type can store both integer and floating-point numbers.
- **JS code.**
  var a = 10
  var b = 20
  var c = a + b;
  console.log(c);
  console.log(typeof(a,b));
- **Strings:** A string is a sequence of one or more characters that may consist of letters, numbers, or symbols.
- **JS code.**
  let Name = "Shanelhai";
  console.log(typeof(Name));

- **Bigint:** BigInt is a special numeric type that provides support for integers of arbitrary length. A bigint is created by appending n to the end of an integer literal or by calling the function BigInt that creates bigints from strings, numbers etc.
- **JS code.**
  let x = 10000000000012345566778899987765543332n;
  console.log(x);
- **Symbol:** The JavaScript Symbol is a primitive data type, just like Number, String, Boolean, etc. It represents a unique identifier and can be used in various ways. Symbols are used to create object properties, for example, when you want to assign a unique identifier to an object.
- **JS code.**
  const mySymbol = Symbol();
  const myObject = {[mySymbol]: 'Hello World'};
  console.log(myObject[mySymbol]);
- **No-primitives data types.**
  1. An object.
  2. An array.
  3. A date.
- **Object: An object is a complex data type that allows us to store collections of data.**
- **JS code.**
  const person = {firstName:"John", lastName:"Doe"};
  console.log(typeof(person));
- **Array object:** The Array object is used to store multiple values in a single variable.
- **JS code.**
  const cars = ["Saab", "Volvo", "BMW"];
  console.log(typeof(cars));
- **Date object:**
  const date = new Date("2022-03-25");
  console.log(typeof(date));

Operators.

- **Operators.**
- Operators are the symbol. Operators are used to performing specific mathematical and logical computations on operands.
- **There are many types of operators.**
- **Arithmetic operators:** Arithmetic operators are used to perform mathematical operations in programming.
    1. Addition = (+).
    2. Subtraction = (-).
    3. Multiplication = (*).
    4. Divide = (/).
    5. Modulo = (%).
    6. Exponential = (**).
- **JS code.**

```
const prompt = require('prompt-sync')();
var number1 = parseInt (prompt ("Enter the number a:"));
var number2 = parseInt (prompt ("Enter the number b:"));
var sum = number1 + number2;
console.log("addition",sum);
var sum = number1 - number2;
console.log("subtraction",sum);
var sum = number1 * number2;
console.log("muntiplication",sum);
var sum = number1 / number2;
console.log("division",sum);
var sum = number1 % number2;
console.log("modulo",sum);
var sum = number1 ** number2;
console.log("exponential",sum);
```

- **Assignment operators:** Assignment operators are used to assign values to variables:
    1. Addition equal to = (+=).
    2. Subtraction equal to = (-=).
    3. Multiplication equal to = (*=).
    4. Divide equal to = (/=)
    5. Modulo equal to = (%=).
    6. Equal to = (=).
    7. Exponential equal to = (**=).
- **JS code.**

```
const prompt = require('prompt-sync')();
var x = parseInt (prompt ("Enter the number x:"));
var y = parseInt (prompt ("Enter the number y:"));
console.log("Addition:",x+=y);
console.log("Subtraction:",x-=y);
console.log("Muntiplication:",x*=y);
console.log("Division:",x/=y);
console.log("Modul:",x%=y);
console.log("exponential:",x**=y);
```

- **Increment / Decrement operators.**
- **Increment operators**: (++)= it is used to increase the value of a variable by one.
- **JS code.**

```
let abc = 5;
abc++;
console.log(abc);
```

output = 5+1 = 6;

- **Decrement operators:** (--) = it is used to decrement the value of a variable by one.
- **JS code.**

```
let abc = 5;
abc--;
console.log(abc);
```

output = 5-1 = 4;

- **Comparison operators.**
- Used to compare two operands and return the value ‘true’ and ‘false’.
- **Type of comparison operators.**
  1. Equal to = (==).
  2. Not equal to = (!=).
  3. Less than = (<).
  4. Less equal to = (<=).
  5. Greater then = (>).
  6. Greater then equal to = (>=).
- **JS code equal to.**

```
let x = 5;
let y = 5;
if(x == y)
console.log("it is a equal x and y:");
else
console.log("it is not equal x and y:");
```

- **JS code greater than and less than.**

```
const prompt = require('prompt-sync')();
let a = parseInt(prompt("Enter the number a:"));
let b = parseInt(prompt("Enter the number b:"));
let c = parseInt(prompt("Enter the number c:"));
if(a >= b && a >= c)
{
console.log("A is a greatest number:");
}
else
if(b >= a && b >= c)
{
console.log("B is a greatest number:");
}
else
if(c >= a && c >= b)
{
console.log("C is a greatest number:");
}
```

- **Logical operators.**
- Logical operators are used to combine conditional statements.
- **There are three types of logical operators.**
  1. And = &&
  2. Or = ||
  3. Not =!
  4. Typeof = it tells us the operands type.
- **JS code.**

```
const prompt = require('prompt-sync')();
let a = parseInt(prompt("Enter the number a:"));
let b = parseInt(prompt("Enter the number b:"));
let c = parseInt(prompt("Enter the number c:"));
if(a > b && a > c)
{
console.log("A is a greatest number:");
}
else
if(b > a && b > c)
{
console.log("B is a greatest number:");
}
else
if(c > a && c > b)
{
console.log("C is a greatest number:");
}
```

- **JS code Not =! .**

```
const prompt = require('prompt-sync')();
let num = parseInt(prompt("Enter the number:"));
if(num%2==0){
console.log("it's an even number:");
}else
if(num%2!=0){
console.log("it is not a even number:");
}
```

- **String operators.**
- String operators are used to perform operations on strings and manipulate variable values.
- **JS code string.**

```
let a ="Shaan";
let b ="Saifi";
let c = a + " "+ b;
console.log(c);
```

- **Bitwise operators.**
- Bitwise operators are used to compare (binary) numbers.
- **JS code.**

```
const prompt = require('prompt-sync')();
let a = parseInt(prompt("Enter the number a:"));
let b = parseInt(prompt("Enter the number b:"));
a = a^b
b = a^b
a = a^b
console.log("After the swapping");
console.log("a number",a);
console.log("b number",b);
```

- **Conditional statements operators in java script.**
- Conditional statements are used to perform different actions based on different conditions is true or false.
- **Types of conditional statement.**
  1. If.
  2. If-else.
  3. If-elseif-else.
  4. Switch.
- **If statement.**
- **JS code.**

```
let num = 10;
if(num % 2 == 0)
{
console.log("it is a even number:");     }
console.log("thank you");
```

output = it's an even number thank you!

- **If-else statement.**
- It executes some code even if the condition is false.
- **JS code.**

```
const prompt = require('prompt-sync')();
let num = parseInt(prompt("Enter the number:"));
if(num%2==0)
{
   console.log("it is a even number:");
}else{
   console.log("it is a odd nummber:");
}
   console.log("thank you:");
```

output = it's an even number thank you!

- **If-elseif-else statement.**
- If-elseif-else statement executes different codes for are than two conditions.
- **JS code.**

```
const prompt = require('prompt-sync')();
let num = parseInt(prompt("Enter the number:"));
if(num%2==0)
{
   console.log("it is a even number:");
}else
if(num%2!=0)
{
   console.log("it is a odd nummber:");
}
   console.log("thank you:");
```

output = it's an even number thank you!

Switch.

- **Switch statement.**
- The switch case statement allow variable to be tested again in the list value each value is called case. The switch statement in checked in every value in case.
- **JS code.**

```
const prompt = require('prompt-sync')();
let a = parseInt(prompt("Enter the first number: "));
let b = parseInt(prompt("Enter the second number: "));
let operator = prompt("Enter the operator (+, -, *, /): ");
switch(operator)
{
case "+":
   result = a + b;
console.log("Addition:",result);
break;
case "-":
   result = a - b;
console.log("Subtraction",result);
break;
case "*":
   result = a * b;
console.log("Muntiplication",result);
break;
case "/":
   result = a / b;
console.log("Division",result);
break;
default:
console.log("This operation is not yet supported.");
}
```

Loop.

- **Loop in JS.**
- They are used to run a block of code multiple times.
  1. While loop.
  2. Do-while loop.
  3. For loop.
  4. Foreach loop.
- **While loop.**
- While(condition) {
- // do something}
- While loop is a entry control loop.
- While loop is a pre- test loop.
- The condition is specified begin of the loop.
- **JS code.**

```
const prompt = require('prompt-sync')();
let table = parseInt(prompt("Enter the table number:"));
i = 1;
while (i <= 10)      {
    r = table*i;
   console.log(" ",r);
i = i+1                          }
```

output = 2 4 6 8 10 12 14 16 18 20

- **Do-while loop.**
- Do { // do something }
- While(condition);
- Do while loop is a exit control loop.
- Do while loop is a past- test loop.
- The condition is specified end of the loop.
- **JS code.**

```
const prompt = require('prompt-sync')();
let table = parseInt(prompt("Enter the table number:"));
i = 1;
do{
   r = table*i;
   console.log(" ",r);
i = i+1
}while (i <= 10)
output = 2 4 6 8 10 12 14 16 18 20
```

- **For loop.**
- For loop(initialisation; condition; updation) {
- // do something}
- For loop is a entry control loop.
- **JS code.**

```
const prompt = require('prompt-sync')();
for (var i = 0; i < 4; i++) {
let firstname = parseInt(prompt("Enter the firstname:"));
let lastname  = parseInt(prompt("Enter the lastname:"));
let age       = parseInt(prompt("Enter the age:"));
let salary    = parseInt(prompt("Enter the salary:"));
console.log(i);
}
```

- **Foreach loop.**
- It is used as a loop but not a loop.
- It is a function.
- It is used to iterate through arrays.
- **JS code.**

```
var names = ["Pawan","Aman","Ujjwal","Shaan"];
names.forEach(function(value,index){
   console.log(index,value);
});
```

- **Nested loop.**
- **JS code.**

```
const n = 5;
let str = '';
for (let i = 1; i <= n; i++) {
   let row = '';
   for (let j = 1; j <= n - i; j++) {
      row += ' ';                    }
   for (let j = 1; j <= 2 * i - 1; j++) {
      row += '*';                    }
   str += row + '\n';                }
console.log(str);
```

String.

- **String in JS.**
- A java script is zero or more characters written inside quotes.
- Strings can be created as primitives, from string literals, or as objects, using the string () constructors.
- It is always enclosed within quotes single and double (‘,’)(“,”).
- EXAMPLE
- char name[ ] = {'S', 'H', 'R', 'A', 'D', 'H', 'A','\0'};
- char class[ ] = {'A', 'P', 'N', 'A', ' ', 'C', 'O', 'L', 'L', 'E', 'G', 'E', '\0'};

- Initialising Strings
- char name[ ] = {'S', 'H', 'R', 'A', 'D', 'H', 'A','\0'};
- char name[ ] = "SHRADHA";
- char class[ ] = {'A', 'P', 'N', 'A', ' ', 'C', 'O', 'L', 'L', 'E', 'G', 'E', '\0'};
- char name[ ] = "SHRADHA";
- **JS code.**
  let a = "Shaan";
  console.log("Hello!\t");
  console.log("Hope you are having a great time");
  output = hello a!
  Hope you are having a great time.
- **Escape sequence.**
- Denoted by a backward slash (\) and escapes the character that follows it. Example - \n, \t, etc.
- \n and \t = both add a special meaning to the character.
- \n = adds a new line.
- \t = adds a tab space.
- Adds a special meaning to the character.
- Removes the special meaning from the character.
- **JS code.**
  let s = "Hello\"joe\"!";
  console.log(s);
  console.log("Hope you have a great time!");
  output = Hello"joe"! Hope you have a great time!

- **Concatenation.**
- It means to link or join together.
- ‘+’ to concatenation strings.
- **JS code.**
  let a = "India won";
  let b = " the match.";
  let c = a + b;
  console.log(c);
  output = India won the match.
- **Template literals.**
- It is a string enclosed by a backtick (~ `).
- It allows us to print a variable within a string.
- **JS code.**
  "use strict";
  let name = "joe";
  let s = `Hello ${name}!`;
  console.log(s);
  output = Hello joe!
- **String as objects.**
- Let str = new String (“Hello world!”);
- **What we will do.**
- Finding a string in a string.
- **Searching for a string in a string.**
- **JS code.**
  const s = "My Name is Shanelhai";
  console.log(s.length);
  console.log(s.indexOf("Name"));
  console.log(s.lastIndexOf("Name"));
  console.log(s.search("Shanelhai"));

- **Extracting string parts.**
- Three methods for extracting a part of a string.
- Slice (start, end).
- The slice extracts a part of a string and returns the extracted part in a new string.
- Substring (start, end).
- The substring is the similar to slice. but difference is that substring cannot accept negative indexes.
- Substr (start, length).
- The substr is similar to slice. but the difference is that the second parameter specifies the length of the extracted part.
- **JS code.**
- const s = "Apple, Banana, kiwi, Mango";
- console.log(s.slice(0, 4));
- console.log(s.slice(7, -2));
- console.log(s.substring(0, 4));
- console.log(s.substring(7, -2));
- **Replacing string content.**
- The string.prototype.replace(searchfor, replaceWith)
- The replace() method replaces a specified value with another value in a sting.
- point to remember.
- The replace method does not change the string it is called on. it returns a new string.
- By default the replace method replaces only the first match.
- By default, the replace method is case sensitive.
- writing SHANELHAI (with upper case) will not work.
- **JS code.**
- var mydata = "My Name is shanelhai.";
- console.log(mydata);
- console.log(mydata.replace("shanelhai","Shanelhai"));

- **Extracting string characters.**
- There are 3 methods for exracting string characters.
- charArt(position).
- charCodeAt(position).
- The charcodeAt method returns the unicodes of the character at specified index in string.
- The method returns a UTF-16 code.
- (an integer between 0 and 65535).
- The unicodes standard provides a unique number for every character, no matter the platfrom, device, application, or language. UTF-8 is a popular uncodes encodes encoding which has 88-bit code units.
- **property access[].**
- ECMAScript 5 (2009) allows property access [] on strings.
- **JS code.**
- var str = "HELLO WORLD.";
- console.log(str.charAt(1));
- console.log(str.charCodeAt(0));
- console.log(str[0]);
- **Other useful methods.**
- **JS code.**
- var str = "Shanelhai Saifi";
- console.log(str.toUpperCase());
- console.log(str.toLowerCase());
- trim removes whitespace from both sides.
- console.log(str.trim());
- **how to convert a string an array.**
- A string can be converted to an array with the slipt() method.
- **JS code.**
- var text = "a, b, c, d";
- console.log(text.split(",")); //split on commas.
- console.log(text.split(" ")); //split on spaces.
- console.log(text.split(" | ")); //split on pipe.

Array.

- **Array.**
- The array is a collection of same data types.
- **Array types.**
  1. Numeric indexed.
  2. Object.
- **Numeric indexed array.**
- It has a numeric index that starts from zero.
- **JS code how to create an arrray.**
  let array = ["Pawan","Aman","Ujjwal","Shaan"];
  console.log(array);
- **Arrays are objects.**
- We create a new object like same define the Array and all Array element ko hum argument pass karat hain.
- Objects keyboard place we have a Array keyword hain.
- This place Array name class and constructor hona chahiye.
- **JS code.**
  "use strict";
  let a = new Array("HTML","CSS","Bootstrap","DBMS","PHP");
  console.log(a[0],a[1],a[2],a[3]);
- **JS code.**
  var students = [{Name: "Pawan", age: 18},
  {Name: "Aman", age: 19},
  {Name: "Ujjwal", age: 22},
  {Name: "Shaan", age: 21},
  {Name: "Suryansh", age: 21}];
  console.log(students);
  for(var i=0;i<students.length;i++){
  console.log(students[i].Name,students[i].age); }
  output: [{ Name: 'Pawan', age: 18 }, { Name: 'Aman', age: 19 }, { Name: 'Ujjwal', age: 22 }, { Name: 'Shaan', age: 21 }, { Name: 'Suryansh', age: 21 }]
  Pawan 18
  Aman 19
  Ujjwal 22
  Shaan 21
  Suryansh 21

- **Array methods.**
- **Push () method** = it is used to add an element at the end of an array.
- **JS code.**

```
var a = ["Pawan","Aman","Ujjwal","Shaan"];
console.log(a[0],a[1],a[2],a[3])
a.push("Suryansh");
console.log(a[0],a[1],a[2],a[3],a[4]);
```

output: Pawan Aman Ujjwal Shaan
Pawan Aman Ujjwal Shaan Suryansh

- **Unshift method** = it is used to add an element at the start of an array.
- **JS code.**

```
const number = [1, 2, 3, 5, 7];
number.unshift(4, 6);
console.log(number);
```

- **Pop () method** = it fetches the last element of the array and them removes it form the array.
- **JS code.**

```
var a = ["Pawan","Aman","Ujjwal","Shaan"];
console.log(a[0],a[1],a[2],a[3])
a.pop("Shaan");
console.log(a[0],a[1],a[2],a[3]);
```

output: Pawan Aman Ujjwal Shaan
Pawan Aman Ujjwal

- **For-of loop method.**
- It is exclusively used for arrays.
- It iterates through all the elements of an array.
- **JS code.**

```
"use strict";
let a = ["HTML","CSS","Bootstrap","DBMS","PHP"];
for(let element of a)        {
console.log(element + "");                }
```

output: HTML CSS Bootstrap DBMS PHP

- **Foreach loop method.**
- It is used as a loop but not a loop.
- It is a function.
- It is used to iterate through arrays.
- **JS code.**

```
var names = ["Pawan","Aman","Ujjwal","Shaan"];
names.forEach(function(value,index){
   console.log(index,value);
});
```

Output: 0 Pawan 1 Aman 2 Ujjwal 3 Shaan

- **Map () method.**
- It is specific to array objects.
- It runs on an array and creates a new array.
- **JS code.**

```
var arr = [1,2,3,4,5];
var b = arr.map(test);
console.log(b);
function test(x){
   return x * 10;
}
```

Output: 10 20 30 40 50

- **Filter method.**
- It is used to create a filter on the elements of an array based on the condition provided by the function.
- It creates a new array and assigns the returned values by the filter () to this array.
- **JS code.**

```
let ages = [10, 12, 16, 18, 19, 20];
console.log(ages);
let b = ages.filter(checkadult);
console.log(b);
function checkadult(age){
   return age>=18;
}
```

Output: [10, 12, 16, 18, 19, 20]
[18, 19, 20

- **Reduce method in array.**
- The reduce method executed a reduce function that you provide on each element of the array resulting in single output value.
- The reduce method function takes four arguments.
  1. Accumulators.
  2. Current value.
  3. Current index.
  4. Source array.
- **JS code Sum and Product *.**
- let array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
  let sum = array.reduce((accum,curr)=>{
  return accum + curr;
  },2) // initial value add.
  console.log(sum);
- **JS code avg.**
  let array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
  let avg = array.reduce((accum, currval, index, array) => {
  return accum += currval;
  }) // initial value add.
  console.log(avg);
- **Traversal in array.**
- Traversal in an array is the process of visiting each element of the array exactly once. This can be done in a variety of ways, but the most common is to use a loop.
- **JS code.**
  array = ["Pawan","Aman","Ujjwal","Shaan"];
  if we want to check the length of elements of an array.
  console.log(array[array.length - 1]);
  console.log(array.length);
  for (var i= 0; i<array.length;i++)
  {
  console.log(array[i]);
  }
  for in loop.
  for(let elements of array){
  console.log(elements);
  }

- **Searching in array.**
- **Method.**
- **Find.**
- The find() method is a very powerful tool for searching arrays, and it is often the simplest way to find an element in an array.
- **JS code.**
- //how to get a name on search only one letter.
  var array = ["Pawan","Aman","Ujjwal","Shaan"];
  var result = array.find(array => array.startsWith("P"));
  console.log(result);
- **Includes.**
- The includes() method takes two parameters: the string to search for and the index at which to start the search. The method returns true if the substring is found, or false if not.
- **How to check data present or not in array.**
  const colors = ['red','green','blue','white','pink'];
  const ispresent = colors.includes('red');
  console.log(ispresent);
- **Indextof().**
- The JavaScript indexOf() method returns the first index at which a given element can be found in the array, or -1 if it is not present.
- **lastIndexOf().**
- The lastIndexOf() method returns the last index (position) of a specified value. The lastIndexOf() method returns -1 if the value is not found.
- **JS code.**
- **how to get a name on search only one letter.**
  var array = ["Pawan","Aman","Ujjwal","Shaan"];
  var result = array.find(array => array.startsWith("P"));
  console.log(result);
  var arrays = array.includes("Pawan");
  console.log(arrays);
  var arrays = array.indexOf("Shaan");
  console.log(arrays);
  var arrays = array.lastIndexOf("Ujjwal");
  console.log(arrays);

Function.

- **Function in JS.**
- A java script function is a block of code designed to perform a particular task.
- It takes data to performs tasks called ‘function arguments.’
- It can take any number of arguments.
- It returns only one value at a time.
- **JS code simple function.**

```
function sum(){
    const prompt = require('prompt-sync')(); // definition a function.
    let a = parseInt(prompt("Enter the number a:"));
    let b = parseInt(prompt("Enter the number b:"));
    let sum = a + b;
    console.log("The sum of two number:", sum);          }
sum();                  // calling a function.
```

- **Function parameter vs function arguments.**
- Function parameter are the names listed in the function’s definition.
- Function arguments are the real values passed to the function.
- **JS code function parameter and function arguments.**

```
function sum(a,b){ // function parameter.
    let sum = a + b;
    console.log("The sum of two number:",sum);
}
sum(20,20);  // function arguments.
sum(50,50);
```

- **Function expressions.**
- Function expressions simply means.
- Create a function and put it into the variable.
- **Function return keyword.**
- When java script reaches a return statement the function will stop executing. Function often computes a return value. The return value is returned back to the caller.
- **JS code.**

```
function sum(a,b){ // function parameter.
    return sum = a + b; // function return keyword.          }
var functionexpression = sum(20,20);
console.log("the sum of two number",+ functionexpression);
```

- **Summary.**
- For arrays and objects, if we change the argument variable itself, the original variable remains unaffected.
- If we change the internals of the argument variable, the original variable also gets changed.
- The concept of call by value is the same for objects as well.
- **Anonymous functions.**
- These are the functions that do not have any name.
- **JS code.**

```
var  funexp = function (a,b){ // that is anonymous functions expressions.
   return sum = a + b;                    }
var sum = funexp(20,20);
console.log("the sum of two number",+ sum);
```

- **Question = why do we need anonymous functions even when we are storing it in a variable to call it?**
- **Answer =** they are mainly used in object.
- **Arrow function.**
- It is an alternate way to define a function.
- **JS code.**

```
let multiply = (x, y)=> {
let p = x * y;
return p;
};
let p = multiply(5, 20);
console.log(p);
```

output = 100

- It can have alphanumeric characters or hyphen or underscore.
- **JS code.**

```
"use strict";
let multiply = (x, y)=> x * y;{
};
let p = multiply(5, 20);
console.log(p);
```

output = 100

- **When value return.**

```
"use strict";
let multiply = (x, y)=> {
return x * y;
};
let p = multiply(5, 20);
console.log(p);
```

output = 100

- Function has only one argument.
- **JS code.**

```
"use strict";
let multiply = x => x * x;{
};
let p = multiply(20);
console.log(p);
```

output = 400

- **Call by value / reference.**
- Call by value is a method of passing arguments to a function by copying variables.
- Call by reference is a method of passing a address.
- **JS code.**

```
function add(a,b){
    return a + b;
}
const prompt = require('prompt-sync')();
let a = parseInt(prompt("Enter the number a:"));
let b = parseInt(prompt("Enter the number b:"));
var sum = add(a,b);
console.log(sum);
```

Date and Time.

- **Date and Time.**
- javascript date objects represent a single moment in time in a platfrom-independent format.
- Date objects contain a number that represent milliseconds since 1 january 1970 UTC.
- **Creating date objects.**
- There are 4 ways to create a new date object.
- **new Date().**
- Date objects are created with the new Date() constructor.
- **JS code.**

```
var currDate = new Date();
console.log(currDate);
console.log(new Date().toLocaleString());
console.log(new Date().toString());
```

- **Date.now().**
- Returns the numeric value corresponding to the current time-the number of milliseconds elapsed since january 1, 1970 00:00:00 UTC.
- **JS code.**

```
console.log(Date.now());
```

- **new Date(year, months....)**
- 7 numbers specify year, month, day, hour, minute, seconde, and millisecond (in that order).
- Note: javascript counts months from 0 to 11.
- January is 0. December is 11.
- **JS code.**

```
var d = new Date(2024, 1, 29, 21, 59, 30, 0);
console.log(d.toLocaleString());
```

- **new Date(DateSharing).**
- new Date(DateSharing) creates a new date objects from a date string.
- **JS code.**

```
var d = new Date("January 1, 2024 11:13:00");
console.log(d.toLocaleString());
```

- **new Date(milliseconds).**
- new Date(milliseconds) creates a new date object as zero time plus milliseconds.
- **JS code.**
  var d = new Date(0);
  var d = new Date(1706546938461);
  var d = new Date(86400000*2);
  console.log(d.toLocaleString());
- **Date Method.**
- **JS code.**
  const CurrDate = new Date();
- **How to get the indivisual date.**
  console.log(CurrDate.toLocaleString());
  console.log(CurrDate.getFullYear());
  console.log(CurrDate.getMonth());
  console.log(CurrDate.getDate());
  console.log(CurrDate.getDay());
- **How to set the indivisual date.**
  const CurrDate = new Date();
  console.log(CurrDate.getFullYear(2024));
- The setFullYear()method can optionally set month and day.
  console.log(CurrDate.getFullYear(2024, 10, 5));
  console.log(CurrDate.setMonth(10));
  console.log(CurrDate.setDate(5));
  console.log(CurrDate.toLocaleString());

- **Time method.**
- **JS code.**
  const CurrTime = new Date();
- How to get the indivisual Time.
  console.log(CurrTime.getTime());
- The getTime() method returns the numbers of milliseconds since january 1, 1970.
  console.log(CurrTime.getHours());
- The getHours() method returns the hours of a date as a numbers(0-23).
  console.log(CurrTime.getMinutes());
  console.log(CurrTime.getSeconds());
  console.log(CurrTime.getMilliseconds());
- **How to set the indivisual Time.**
- **JS code.**
  const CurrTime = new Date();
  console.log(CurrTime.setTime());
  console.log(CurrTime.setHours(5));
  console.log(CurrTime.setMinutes(5));
  console.log(CurrTime.setSeconds(5));
  console.log(CurrTime.setMilliseconds(5));

ECMA Script.

- **European Computer Manufacturers Association.**
- ECMAScript (ES) is a standard for scripting languages, including JavaScript, JScript, and ActionScript. It's also known as JavaScript.
- **Here are some of the features of ECMAScript:**
- It is a high-level language, which means that it is easy to read and write.
- It is an interpreted language, which means that it does not need to be compiled before it can be run.
- It is a dynamic language, which means that its behaviour can change at runtime.
- It is a prototype-based language, which means that objects are created by cloning existing objects.
- It is a functional language, which means that it supports functions as first-class citizens.
- **Let vs Const vs Var.**
- **JS code.**

```
let myname = "Shanelhai Saifi";
console.log(myname);
myname = "Shaan";
console.log(myname);
const myname = "Shanelhai Saifi";
console.log(myname);
myname = "Shaan";
console.log(myname);
```

- **Var =>** Function Scope.
- **Let and Const =>** Block Scope.
- **JS code.**
-

```
function biodata(){
    var myfirstname = "Shanelhai";
    console.log(myfirstname);
if(true){    // let and const block scope
    var mylastname = "Saifi";
    console.log("inner"+mylastname);
console.log("inner"+myfirstname);              }
console.log("innerouter"+mylastname);                }
biodata();
```

- **template literals (Template strings).**
- Template literals (template strings) allow you to use strings or embedded expressions in the form of a string. They are enclosed in backticks.
- Java script program to print table for given number.
- **JS code.**

```
const prompt = require('prompt-sync')();
let table = parseInt(prompt("Enter the table number:"));
for (let i = 1; i < 11; i++) {
   // console.log('${table} * ${i} = ${table * i} ');
   console.log(table+"*"+i+"="+table * i);
}
```

- **Rest Parameter Operator.**
- **Rest Parameter.**
- It allows us to collect an indefinite number of arguments passed to a function in the form of an array.
- It is denoted by three dots(…)before a variable name.
- Rest parameter help convert the simple array.
- **JS code.**

```
function sum(name, ...args){
   console.log(arguments);
   console.log(`Hello ${name}:`);
   let sum = 0;
   for(let i in args){
      sum += args[i];
   }
   console.log(sum);
}
sum("Shaan",20,30,40);
sum("Shanelhai",20,40);
```

output: Hello Shaan:90.

- **We have special argument a and b that is used rest Parmenter.**
- **JS code.**

```
"use strict";
function calculateSum(a,b,...args){ // cs define karna hai a and b
let sum = a+b;
args.forEach(function(element){
sum += element;
});
return sum;
}
let s = calculateSum(3,5,7,9);
console.log(s+"");
s = calculateSum(3,5,7,9,11);
console.log(s+"");
```

output: 24 35.

- **Spread operator.**
- It spreads or expands a variable into more than one.
- Also, it spreads an array into its elements.
- It is denoted by three dots(…) before a variable name.
- **JS code.**

```
"use strict";
let odd = [1,3,5,7,9];
let even = [2,4,6,8,10];
// array define odd and even array store the numbers.
let numbers = [...odd,...even];
//spared operator convert the even and odd number one element
numbers.forEach(function(element){
//one new array make. we have put both this array store variable numbers.
//forEach method used call function and print the array elements.
console.log(element+"");
});
```

Output = 1 3 5 7 9 2 4 6 8 10.

- **Destructuring in ES6.**
- The destructuring assignment syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.
- **Array Destructuring.**

```
const myData = ["Shanelahi","Saifi",26,"BCA"];
let = [myFirstName,myLastName,myAge,mycourse] = myData;
console.log(myFirstName);
console.log(myLastName);
console.log(myAge);
console.log(mycourse);
```

- **Object Destructuring.**
-

```
const mybioData = {
   myFirstName : 'Shanelhai',
   myLastName :  'Saifi',
   myAge      :   21,
   mycourse   :    'BCA'
}
let  {myFirstName,myLastName,myAge,mycourse} = mybioData;
console.log(myFirstName);
console.log(myLastName);
console.log(myAge);
console.log(mycourse);
```

- **Object Properties.**
- **We can now use Dynamic Properties.**
- **how to get a Dynamic Data.**
- **JS code.**

```
let myName = "Shanelhai";
const mybio = {
   [myName] : "hello how are you",
   [20 + 1] : "is my age"                          }
console.log(mybio);
```

- **no need to write key and value, if both are same.**

```
let myName = "Shanelhai";
let myAge  = 21;
const myBio = {myName,myAge}
console.log(myBio);
```

- **Exponentiation Operator.**
- The exponentiation operator (**) in JavaScript is used to raise a number to a power. It is a right-associative operator,
- **JS code.**
  const result = 2 ** 3;
  console.log(result);
- **Features of ES8 ECMA Script.**
- **String padding.**
- JS code.
  let myName ="Shanelhai".padStart(7);
  let myAge = "26".padEnd(10);
  output: '26 '
- **Object.value and entries.**
  const person = {name: 'Shanelhai', age: 87};
  console.log(Object.values(person));
  console.log(Object.entries(person));
  output: [['name', 'Shanelhai'], ['age', 87]]
- **Back to Normal in array.**
- **JS code.**
  const person = {name: 'Shanelhai', age: 87};
  const arrObj = Object.entries(person);
  console.log(Object.fromEntries(arrObj));
  output: { name: 'Shanelhai', age: 87 }
- **How to flat an array.**
- **converting 2d and 3d array into one dimensional array.**
  const arr = [
  ['zone_1','zone_2'],
  ['zone_3','zone_4'],
  ['zone_5','zone_6'],
  ['zone_7','zone_8',['zone_9','zone_10']]];
  console.log(arr.flat(2)); , Infinity
- **JS code BigInt.**
  let oldNum = Number.MAX_SAFE_INTEGER;
  const newNum = 9007199254740991n+ 15n;
  console.log(newNum);
- console.log(typeof newNum);
  output: 9007199254741006n bigint

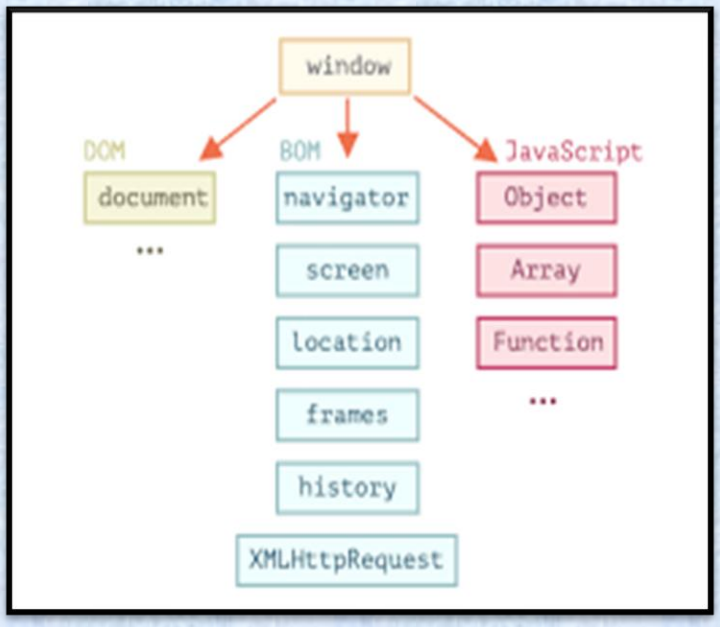
Math object in javascript.

- **Math object.**
- The JavaScript Math object allows you to perform math mathematical tasks on numbers.
- **console.log(Math.PI);**
- **Math.round()**
- returns the value of x rounded to its nearest integer.
  let num = 10.5565;
  console.log(Math.round(num));
- **Math.pow()**
- Math.pow(x,y)returns the value of x to the power of y.
  console.log(Math.pow(2,3));
  console.log(2**3);
- **How to find sqrt in javascript.**
- **Math.sqrt()**
- Math.sqrt(x) returns the square root of x.
  console.log(Math.sqrt(25));
  console.log(Math.sqrt(81));
  console.log(Math.sqrt(66));
- **Math.abs()**
- Math.abs(x) returns the absolute (positive) value of x.
  console.log(Math.abs(-55));
  console.log(Math.abs(-55.5));
  console.log(Math.abs(-955));
- **Math.ceil()**
- Math.ceil(x) returns the value x rounded up its nearest integer value increment.
  console.log(Math.ceil(4.4));
  console.log(Math.ceil(99.1));
- **Math.floor()**
- Math.floor(x) returns the value of x rounded down to its nearest integer value.
  console.log(Math.floor(4.7));
  console.log(Math.floor(99.1));
- **Math.mim()**
- Math.min() find the lowest number.
  console.log(Math.min(0, 15, -8, -200));
- **Math.max()**
- Math.max() find the largest number.
  console.log(Math.max(0, 15, 18, 20));

- **Math.random()**
- Math.random() returns a random number between 0 (inclusive), and 1(10).
  console.log(Math.floor(Math.random()*10)); // 0 to 9.
- **Math.trunc()**
- The trunc() method returns the integer part of a number.
  console.log(Math.trunc(4.6));
  console.log(Math.trunc(-99.1));

DOM IN JavaScript.

- **Window.**
- The window object represents the browser window and serves as the global object in client-side JavaScript. It encapsulates the entire browser window, including tabs, frames, and frames.
- **Document Object Methods.**
- DOM stands for Document Object Model. It is a programming interface that allows us to create, change, or remove elements from the document. We can also add events to these elements to make our page more dynamic.
- **Browser Object Model.**
- The Browser Object Model (BOM) is a collection of objects exposed by the browser that allow JavaScript to interact with the browser window, document, and other browser-specific functionalities. The BOM includes objects such as Window, Navigator, Location, History, and Screen.
- **Features of JavaScript.**
- It allows us to capture user actions in the form of java script events.
- It allows us to make changes to the existing HTML elements on a web page.



- **JS code.**
- Function goback(){ window.history.back();}

- **Show current URL.**
- alert(location.href);
  if (confirm("want to vist ThapaTechnical")){
  location.href="https://www.youtube.com/watch?v=KGkiIBTq0y0&t=30596s";
  }
- **DOM Navigation.**
- Document.head, Document.body
- Document.body.children
- Document.body.children.length
- **Accessing HTML elements.**
- DOM = Document Object Methods.
- It is a programming interface.
- It helps to access the HTML elements of a web page in the form of java script objects.
- **HTML code.**
  <html>
  <head>
  <title>Browser</title>
  </head>
  <body>
  <h1>my header</h1>
  this is a paragraph.
  <script type=”text/javascript”scr=”js/script.js”></script>
  </body>
  </html>
- **Jave Script.js code.**
  let el = document.getElementById("abc");
- **Method.**
  1. getElementsByClassName.
  2. getElementsByTagName.
  3. QuerySelectorAll.

- **Get Elements By Class Name.**
- **HTML code.**

```
<html>
<head>
  <title>Browser</title>
</head>
<body>
  <h1 class="xyz">my header</h1>
  <p class="xyz">this is a paragraph.</p>
<script type=”text/javascript”scr=”js/script.js”></script>
</body>
</html>
```

- **Java Script.js.**

```
let els = document.getElementsByClassName("xyz");
// all class element return. Array ko return kar ga.
```

- **Get Element By Tag Name.**
- **HTML code.**

```
<html>
<head>
  <title>Browser</title>
</head>
<body>
  <h1 class="xyz">my header</h1>
  <p class="xyz">this is a paragraph.</p>
  <p>this is a another paragraph.</p>
  <script type=”text/javascript”scr=”js/script.js”></script>
</body>
</html>
```

- **Java Script.js.**

```
let els = document.getElementsByClassName("p");
// all tag element return. Array ko return kar ga.
```

- **Query Selector All.**
- **HTML code.**

```
<html>
<head>
 <title>Browser</title>
</head>
<body>
 <h1 class="xyz">my header</h1>
 <p class="xyz">this is a paragraph.</p>
 <p>this is a another paragraph.</p>
 <script src="script.js"></script>
</body>
</html>
```

- **Java Script.js.**
- let els = document.querySelectorAll(".xyz");
  // all HTML element return matching the xyz. Queryselector all method be array ko return karta hai.
- **how get element value.**
- console.log(document.getElementById('web-id').innerHTML);
- console.log(document.querySelector('#web-id').innerHTML);
- **Document object properties.**
- **Inner HTML property.**
- It helps to fetch the content of the element.
- **HTML code.**

```
<html>
<head>
 <title>Browser</title>
</head>
<body>
 <h1>my header</h1>
 <p id="abc">this is a paragraph.</p>
<script type=”text/javascript”scr=”js/script.js”></script>
</body>
</html>
```

- **Java Script.js.**
  let el = document.getElementById("abc");
  //el element paragraph use a represent kar ra hai jiski id abc equal hai.
  / changing element content using inner HTML property.
  alert(el.innerHTML);
  el.innerHTML = "This is a some one new content.";
  //changing CSS property using inner HTML property.
  el.style.color="red";
- **Event handler on click property.**
- It is used to capture user click.
- **HTML code same Java Script.js.**
  let el = document.getElementById("abc");
  el.onclick = function(){
  /*this code will be executed when the element is clicked.*/
  alert("Elements is clicked.")
  }
  //wo paragraph jo abc id ka equal hai when click hota hai to function executed.
  // when user kise or element per click kar ga to no impact.
- **When both properties combine inner HTML and on click properties.**
- Inner HTML and on click properties = when create highly interactive web page.
- **Creating interactive web page.**
- **HTML code same Java Script.js.**
  let el = document.getElementById("abc");
  el.onclick = function(){
  el.innerHTML = "This is some new content.";
  el.style.color = "red";
  }
  // when user click the paragraph jiski id abc ka equal hai to hum calling the function.

- **Alternate way of using event handlers.**
- **HTML code.**

```
<html>
<head>
 <title>Browser</title>
</head>
<body>
  <h1>my header</h1>
  <p id="abc" onclick=”changeContent()”>this is a paragraph.</p>
<script type=”text/javascript”scr=”js/script.js”></script>
</body>
</html>
```

- **Debugging JS Using Inspect Elements.**
  1. Step = click the right button and select the inspect element. and the open the panel we are concern only sources tag debugging JS.
  2. Step = click the sources tag. We can see the one panel. Three parts divided.

Events in JavaScript

- **Event.**
- HTML event are "things" that happen to HTML elements.
- When JavaScript is used in HTML page, JavaScript can "react" on these events.
- When the user clicks the button, the myFunction() function will be called, which will display an alert box with the message "Hello World!".
- **Click(onclick).**
- **JS code.**

```
  <script>
      function hello(){
         alert("hello everyone");
      }
   </script>
</head>
<body>
   <button onclick="hello()">Click Me</button>
</body>
</html>
```

- **double Click(ondblclick).**
- **JS code.**

```
<script>
      function hello(){
         alert("hello everyone");
      }
   </script>
<button ondblclick="hello()">Click Me</button>
```

- **Right Click(oncontextmenu).**
- <button oncontextmenu="hello()">Click Me</button>
- **Mouse Hover(onmouseenter).**
- <button onmouseenter="hello()">Click Me</button>
- **Mouse Out(onmouseout).**
- <button onmouseout="hello()">Click Me</button>
- **Mouse Down(onmousedown).**
- <button onmousedown="hello()">Click Me</button>
- **Mouse Up(onmouseup).**
- <button onmouseup="hello()">Click Me</button>

- **Key Press(onkeypress).**
- button onkeypress="hello()">Click Me</button>
- **Key Up(onkeyup).**
- **<**body onkeyup="hello()"> ye froms main.
- **Load(onload).**
- <body onload="hello()">
- **Unload(onunload).**
- <body onunload="hello()">
- **Resize(onresize).**
- <body onresize="hello()">
- **scroll(onscroll).**
- <p onscroll="hello()">

Timing Events in JavaScript.

- **Timing Events.**
- The window object allows execution of code at specified time intervals.
- These time intervals are called timing events.
- The two key methods to use with JavaScript are.
- There are four methods.
- **setTimeout().**
- The setTimeout() method sets a timer which executes a function or specified piece of code once the timer expires.
- **JS code.**
- ```
  <script>
        setTimeout(myfunction, 5000);
        function myfunction(){
           alert("Welcome to my show");                }
   </script>
  ```
- **clearTimeout().**
- **JS code.**
- ```
  let setTimeId =  setTimeout(myfunction, 5000);
  ```
- ```
  clearTimeout(setTimeId);
  ```
- **setInterval().**
- The setInterval () method repeatedly calls a function or executes a code snippet, with a fixed time delay between each call.
- **JS code.**
  ```
  <script>
       setInterval(myfunction, 5000);
        function myfunction(){
           console.log("Welcome to my show");                }
    </script>
  ```
- **clearInterval().**
- **JS code.**
  ```
  <script>
        var t1 = setInterval(myfunction, 2000);
        function myfunction(){
           console.log("Welcome to my show");                }
        let btn = document.getElementById("btn");
        btn.addEventListener('click',function(){
           clearInterval(t1);                });     </script>
  ```
  ```

Objects.

- **Objects.**
- objects are used to store data and represent real-world entities. They are a fundamental part of the language and are used in almost every JavaScript program.
- **Dot (.) operator.**
- It is used to access the property or method of an object.
- It helps to assign a new value to an existing property.
- It helps to define a new property.
- **JS code.**

```
let a = {
   fistname: "Shanelhai",
   lastname: "Saifi",
   age     : 21,
   email   : "Shanelhai7@gmail.com"                    }
console.log(a.fistname,a.lastname,a.age,a.email);
```

output = Golden retriever 4ft 2 32kg

- **Square [] brackets.**
- It is used to access the properties of an object.
- It helps us to edit an existing property.

```
Dog['breed'] = 'Dalmatian';
```

- It helps us to define a new property.

```
Dog['color'] = 'white with black spots';
```

- **JS code.**

```
let a = {
   fistname: "Shanelhai",
   lastname: "Saifi",
   age     : 21,
   weight  : 55,
   display:function(){
   console.log(this.fistname+this.lastname+this.age+this.weight);
   }            }
console.log(a['fistname'],a['lastname'],a['age'],a['weight']);
a.email = "Shanelhai7@gmail.com";
console.log(a['email']);
```

output = Shanelhai Saifi 21 55 Shanelhai7@gmail.com

- **Different between Dot (.) operator and Square [ ] brackets.**
- **Dot (.) operator =** it is used to access the property or method of an object.
- It helps to assign a new value to an existing property.
- It helps to define a new property.
- **Square [] brackets =** it is used to access the properties of an object.
- It helps us to edit an existing property.
- It helps us to define a new property.
- **New object.**
- **JS code.**

```
let a = new Object();
    a.fistname = "Shanelhai";
    a.lastname = "Saifi";
    a.age      =  21;
    a.weight   = 5;
    a.email    = "Shanelhai&@gmail.com";
    console.log(a.fistname,a.lastname,a.age,a.weight,a.email);
```

output: Shanelhai Saifi 21 5 Shanelhai&@gmail.com

- **Function constructor.**
- It is a normal function used to create an object.
- **JS code.**

```
function Student(firstname,lastname,age,phone,nationality){
    this.firstname = firstname;
    this.lastname  = lastname;
    this.age       = age;
    this.phone     = phone;
    this.nationality = nationality;          }
var Student1 = new Student("Pawan","Singh",18,1289768951,"Indian");
var Student2 = new
Student("Shanelhai","Saifi",21,8976564563,"Indian");
var Student3 = new Student("Ujjwal","Singh",22,9878654563,"Indian");
var Student4 = new
Student("Suryansh","Verma",21,7865674325,"Indian");
console.log(Student1);
console.log(Student2);
console.log(Student3);
console.log(Student4)
```

- **Class.**
- Classes are a template for creating objects.
- **JS code.**

```
class Student {
    constructor(firstname,lastname,age,phone,nationality){
    this.firstname = firstname;
    this.lastname  = lastname;
    this.age       = age;
    this.phone     = phone;
    this.nationality = nationality;
    }
}
    var Student1 = new
Student("Pawan","Singh",18,1289768951,"Indian");
    var Student2 = new
Student("Shanelhai","Saifi",21,8976564563,"Indian");
    var Student3 = new
Student("Ujjwal","Singh",22,9878654563,"Indian");
    var Student4 = new
Student("Suryansh","Verma",21,7865674325,"Indian");
    console.log(Student1);
    console.log(Student2);
    console.log(Student3);
    console.log(Student4);
```

- **Why do we need classes to create an object if the syntax is similar to that of a function constructor?**
- It offers a convenient syntax.
- It allows us to declare the function outside the constructor.
- **JS code.**

```
class Student {
    constructor(firstname,lastname,age,phone,nationality){
    this.firstname = firstname;
    this.lastname  = lastname;
    this.age       = age;
    this.phone     = phone;
    this.nationality = nationality;
    }
    display(){

console.log(this.firstname+this.lastname+this.age+this.phone+this.nation
ality);
    }
}
    var Student1 = new
Student("Pawan","Singh",18,1289768951,"Indian");
    var Student2 = new
Student("Shanelhai","Saifi",21,8976564563,"Indian");
    var Student3 = new
Student("Ujjwal","Singh",22,9878654563,"Indian");
    var Student4 = new
Student("Suryansh","Verma",21,7865674325,"Indian");
    console.log(Student1);
    console.log(Student2);
    console.log(Student3);
    console.log(Student4);
```

- **How to create object and function add.**
- **JS code.**
- let bioData = {
  myname: "Shanelhai",
  myage : 21,
  getData: function(){
  console.log(`my name is ${bioData.myname} and my age is ${bioData.myage}`);
  }
  }
  bioData.getData();
- **output:** my name is Shanelhai and my age is 21
- **no need to write function as well as after es6.**
- **JS code.**
- let bioData = {
  myname: "Shanelhai",
  myage : 21,
  getData(){
  console.log(`my name is ${bioData.myname} and my age is ${bioData.myage}`);
  }
  }
  bioData.getData();
- **output:** my name is Shanelhai and my age is 21
- **What if we want object as a value inside an object.**
- **JS code.**
- let bioData = {
  myname: {
  realName : "shanelhai",
  course : "BCA"
  },
  myage : 21,
  getData(){
  console.log(`my name is ${bioData.myname} and my age is ${bioData.myage}`);
  } }
  console.log(bioData.myname.realName);
  console.log(bioData.myname.course);
  bioData.getData();

- **What is this object?**
- The definition of “this” object is that it contains the current context.
- This object can have different values depending on where it is placed.
- **JS code.**

```
console.log(this);
console.log(this.alert('Awesome'));
```

- **It refers to the current context and that is window global object.**
- **JS code.**
-

```
const obj ={
    myAge: 26,
    myName(){
        console.log(this.myAge);
    }
}
obj.myName();
```

Cookies.

- **Cookies.**
- Cookies let you store user information in web pages.
- Cookies are data, stored in small text files, on your computer.
- **Create a Cookie with JavaScript.**
- document.cookie = "username=John Doe";
- **Delete a Cookie with JavaScript.**
- If you want to delete a cookie. Set Cookies expiry date from the past.
- document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;";
- **JS code.**

```
console.log(document.cookie)
document.cookie = "name1 = Shaan1234561"
document.cookie = "name2 = Shaan1234562"
document.cookie = "name2 = Shaan" // update
let key = prompt("Enter your key:")
let value = prompt("Enter your value:")
document.cookie = `${encodeURIcomponent(key)} =
${encodeURIcomponent(value)}`
console.log(document.cookie)
```

- **local Storage & related methods.**
- local Storage is a powerful tool for storing data locally on the user's computer. It is easy to use, and it can be used to store a variety of data, such as user preferences, login information, and application state.
- **methods.**
- setItem() - takes a key-value pair and adds it to local Storage.
- getItem() - takes a key and returns the corresponding value.
- removeItem() - takes a key and removes the corresponding key-value pair.
- clear() - clears local Storage (for the domain).
- Key (index) – get the key on a given position.

- **JS code.**

```
let key = prompt("Enter key you want to see")
let value = prompt("Enter value you want to see")
localStorage.setItem(key,value)
console.log(`The value at ${key} is ${localStorage.getItem(key)}`)
if (key == "red" || key == "blue"){
   localStorage.removeItem(key)        }
if (key === 0){
   localStorage.clear()                }
```

- **session Storage & related methods.**
- Session storage is a web storage mechanism that allows you to store data for a particular session in the browser. The data is stored in key-value pairs, and it is available until the browser tab or window is closed.
- **methods.**
- setItem(): Adds a key-value pair to session Storage.
- getItem(): Gets the value of a key.
- removeItem(): Removes a key-value pair.
- clear(): Deletes all key-value pairs.
- key(index): Gets the key number index.
- length: This property returns the number of key-value pairs in the session storage.
- **Storage event.**
- When the data gets update in local storage or session storage event triggers with the properties.
- **JS code.**

```
// sessionStorage.getItem("name")
// sessionStorage.clear()
// sessionStorage.removeItem("name")
// sessionStorage.setItem("name","Shaan")
window.onstorage = (e)=>{
   alert("changed")
   console.log(e)
}
```

Advanced JavaScript

- **Event Propagation (Event Bubbling and Event Capturing).**
- **What is event propagation.**
- The Event Propagation mode determines in which order the elements receive the event.
- **Bottom to top** = bubble phase.
- **Top to bottom** = capture phase.
- **Event Bubbling :** Event Bubbling the event is first captured and handled by the innermost elements and then propagated to outer elements.
- **Event Capturing :** Event Capturing the event is captured by the outermost elements and propagated to the inner element. capturing is also called “tricking”, which helps remember the propagation order.
- **JS code.**

```
<div class="parentDiv" id="parentId">
  <div class="chilDiv" id="childId"></div>
</div>
<script>
  const parentId = document.getElementById("parentId");
  const childId = document.getElementById("childId");
  const parentCall = () => {
    alert("Parent Div Call");
    console.log("Parent Div Call");
  };
  const childCall = () => {
    alert("Child Div Call");
    // console.log("Child Div Call");
    // event.stopPropagation();
  };
  parentId.addEventListener("click", parentCall,true);
  childId.addEventListener("click", childCall,true);
</script>
```

- **Higher Order Function.**
- Function which takes another function as an argument is called HOF wo function jo dusre function ko as a argument accept krta hai use HOF.
- **Callback Function.**
- Function which gets passed as an argument to another function is called Callback Function. A callback function is a function that is passed as an argument to another function, to be “called back” at a letter time.
- **JS code.**

```
const add = (a,b)=> {
   return a+b;
}
const subs = (a,b)=> {
   return Math.abs(a-b);
}
const mult = (a,b)=> {
   return a*b;
}
const calculator = (num1,num2, operator)=>{
   return operator(num1,num2);
}
// calculator(5,2,subs); that is higher order function.
console.log(calculator(5,2,subs));
```

- **How JavaScript work and asynchronous JavaScript.**
- **Hoisting in JavaScript.**
- We have a creating phase and execution phase.
- Hoisting in JavaScript is a mechanism where variable and function declarations are moved to the top of their scope before the code execute.
- **What is Scope Chain and Lexical Scoping in JavaScript.**
- The scope chain is used to resolve the value of variable names in JS.
- Scope chain in js is lexically defined, which means that we can see what the scope chain will be by looking at the code.
- At the top, we have the global scope, which is the window object in the browser.
- Lexical scoping means Now, this inner function can get access to their parent function variables but the vice-versa is not true.

- **JS code.**

```
let a = "Hello guys. "; //Global scope.
const first = () =>{ // outTer function can not get the inner function.
    let b = "How are you";
    const second = () =>{ // inner function get the data of outside.
        let c = "Hii I am fine thank you";
        console.log(a+b+c);
    }
    second();
}
first();
```

- **What is Closures in JavaScript.**
- A closures is the combination of a function bundled together (esclosed) with reference to its surrounding state (the lexical environment).
- In other words a closure gives you access to an outer function scope from an inner funciton.
- In JavaScript, closure are created every time a function is created, at function function creation time.
- **JS code.**

```
const outerfun = (a) =>{
    let b = 10;
    const innerfun = () =>{
        let sum = a+b;
        console.log(`the sum of two number:${sum}`);
    }
    return innerfun;
}
let checkClousure = outerfun(5);
checkClousure();
```

- **Synchronous JavaScript prog.**
- JavaScript is synchronous, blocking and single-threaded. This means that the JavaScript engine executes our program sequentially, one line at a time from top to bottom in the exact order of the statements.
- **JS code.**

```
const fun2 = ()=>{
    let a = 10;
    let b = 5;
    a = a^b
    b = a^b
    a = a^b
    console.log("After the swapping");
    console.log("a number",a);
    console.log("b number",b);
}
    const fun1 = () =>{
    var a = 10;
    var b = 20;
    var sum = a +b;
    fun2();
    console.log(sum);
}
fun1();
```

- **Asynchronous Javascript.**
  Asynchronous programming is a technique that enables your program to start a potentially long-running task and still be able to be responsive to other events while that task runs, rather than having to wait until that task has finished.

- **JS code.**

```
const fun2 = ()=>{
   setTimeout(()=>{
      let a = 10;
      let b = 5;
      a = a^b
      b = a^b
      a = a^b
      console.log("After the swapping");
      console.log("a number",a);
      console.log("b number",b);
   },2000);                    }
   const fun1 = () =>{
   var a = 10;
   var b = 20;
   var sum = a +b;
   fun2();
   console.log(sum);                    }
fun1();
```

- **What is Event loop in JavaScript.**
- An event loop is something that pulls stuff out of the queue and places it onto the function execution stack whenever the function stack becomes empty. The event loop is the secret by which JavaScript gives us an illusion of being multithreaded even though it is single-threaded.
- **Function Currying (We will see after Async JS section).**
- Function Currying is a technique of evaluation function with multiple arguments, into sequence of function with single arguments.
- **JS code.**

```
function sum(num1){
    return function(num2){
       return function(num3){
          console.log(num1+num2+num3);     }       }       }
sum(5)(3)(8);
using fatArrow function.
const sum = (num1) => (num2) => (num3)=>
console.log(num1+num2+num3);
sum(5)(3)(8);
```

- **Callback Hell.**
- Callback hell is a phenomenon where a Callback is called inside another Callback. It is the nesting of multiple Callbacks inside a function.
- **JS code.**

```
setTimeout(()=>{
   console.log(`1 works is done.`);
   setTimeout(()=>{
      console.log(`2 works is done.`);
      setTimeout(()=>{
         console.log(`3 works is done.`);
         setTimeout(()=>{
            console.log(`4 works is done.`);
            setTimeout(()=>{
               console.log(`5 works is done.`);
               setTimeout(()=>{
                  console.log(`6 works is done.`);
               },1000);
            },1000);
         },1000);
      },1000);
   },1000);
},1000);
```

- **AJAX call using XMLHttprequest.**
- **Introduction to AJAX.**
- **AJAX** = Asynchronous Java script and XML.
- AJAX is a technique for creating fast and dynamic web pages.
- **Initiating and HTTP Request Using AJAX.**
- **Significance of AJAX.**
- It is a technique to initiate an HTTP request through java script without reloading a web page.
- **Summary.**
- If on any user action we want to make changes in the loaded web page, we can do it by using java script by changing the HTML elements.
- If we also want to update the database or to get some information, we need to use AJAX.

- **JS code.**

```
<script>
        function loadData(){
            var xhttp = new XMLHttpRequest();
            xhttp.onreadystatechange = function(){
                if(this.readyState == 4 && this.status == 200){

document.getElementById("demo").innerHTML.this.responsiveText;
                }                               };
xhttp.open('GET',"D:\software\DATA\JAVASCRIPT\XMLHttprequest\Sh
aan.txt",true);
            xhttp.send();
        }
    </script>
```

- **Java Script.js code.**

```
"use strict";
httpRequest = new XMLHttpRequest();
let el = document.getElementById("abc");
el.onclik = function(){
el.innerHTML = "This is some new content.";
el.style.color = "red";
httpRequest.onreadystatechange = postAjaxFunction;
httpRequest.open('POST','updata_database.php');
httpRequest.setRequestHeader('Content-type','application/x-www-form-
urlencoded');
httpRequest.send('email=aditya@internshala.com');
}
function postAjaxFunction(){
if(httpRequest.readyState===XMLHttpRequest.DONE){
 if(httpRequest.status===200){
var response=JSON.parse(httpRequest.responseText);
alert(response.name+"_"+response.message);
}  else{
alert('Something went wrong!');
}       }       }
```

- **AJAX is executed in 3 layers.**
  1. The first layer is the JS code which initiates the AJAX request.
  2. The second layer is when the PHP code gets executed.
  3. And, once the PHP code is executed it comes back to java script i. e. inside the function which is the layer three.
- **BONUS Section JSON.**
- **JSON = Java Script Object Notation.**
- JSON (JavaScript Object Notation) is a file format used to store and communicate data objects between applications. It uses human-readable language, text, and syntax.
- JSON.stringify turns a JavaScript object into JSON text and stores that JSON text in a string, eg.
- **JS code.**
  var my_object = {key_1: "some Text", key_2: true, key_3: 5};
  var object_as_string = JSON.stringify(my_object);
  console.log(object_as_string);
  typeof(object_as_string);
  JSON.parse turns a string of JSON text into a JavaScript object, eg;
- **JS code.**
  var object_as_string = JSON.parse(object_as_string);
  typeof(object_as_string_as_object);
- **Promises.**
- A Promise in JavaScript is an object that represents the eventual completion (or failure) of an asynchronous operation and its resulting value. A Promise is in one of three possible states: fulfilled, rejected, or pending.
- **Parameters of promises.**
- **Resolve:** successful.
- **Rejects:** failure.
- **Function of promises.**
- **Then :** when a promise is successful, you can then use the resolved data.
- **catch :** when a promise fails, you catch the error, and do something with the error information.
- **Promise Object Properties.**
- 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.

- **Create a Promise**
- To create a promise object, we use the Promise() constructor.
- let promise = new Promise(function(resolve, reject){
- //do something
  });
- **JS code.**

```
let myPromise = new Promise(function(resolve, reject){
    console.log("your order is pending");
    setTimeout(()=>{
        console.log("I am a promise and I am resolve");
        resolve(true);
    },2000)
})
let ImPromise = new Promise(function(resolve, reject){
    console.log("your order is pending");
    setTimeout(()=>{
        console.log("I am a promise and I am rejected");
        reject(new Error("I am a Error"));// error  create.
    },2000)
})
//To get the value.
myPromise.then((value)=>{
    console.log(value)
})
//To catch the errors.
ImPromise.catch((error)=>{
    console.log("some error occurred in Impromise");
})
```

- **Promise Chaining.**
- Promises are useful when you have to handle more than one asynchronous task, one after another. For that, we use promise chaining. You can perform an operation after a promise is resolved using methods then() , catch() and finally() .

- **JS code.**

```
let p1 =  new Promise((resolve, reject)=>{
   setTimeout(()=>{
      console.log("Resolve after two seconds");
      resolve(56)
   },2000);
})
p1.then((value)=>{
   console.log(value)
   let p2 = new Promise((resolve, reject)=>{
      resolve("promise 2")
   })
   return p2
}).then((value)=>{
   console.log("we are done")
   return 2
}).then((value)=>{
   console.log("Now we are pakka done");
})
```

- **Attaching multiple handlers.**
- We can attach multiple handlers. they don't pass the result to each other. they process it independents.
- **JS code.**

```
let p1 = new Promise((resolve, reject)=>{
   // alert("Hey I am not resolved")
   setTimeout(()=>{
      resolve(1);
   },2000)
})
p1.then(()=>{
   console.log("Hurray")
})
p1.then(()=>{
   console.log("Congratulations this promise is now resolved")
})
```

- **Promise API.**
- There are 6 static methods of promise class.
- Promise.all.
- Promise.allSettled.
- Promise.race.
- Promise.any.
- Promise.resolve.
- Promise.reject.
- **JS code.**

```
let p1 = new Promise((resolve, reject)=>{
    setTimeout(()=>{
        resolve("value 1");
    },1000);
})
let p2 = new Promise((resolve, reject)=>{
    setTimeout(()=>{
        resolve("value 2");
        // reject(new Error("Error"));
    },2000);
})
let p3 = new Promise((resolve, reject)=>{
    setTimeout(()=>{
        resolve("value 3");
    },3000);
})
// let Promise_all = Promise.all([p1, p2, p3])
// let Promise_all = Promise.allSettled([p1, p2, p3])
// let Promise_all = Promise.race([p1, p2, p3])
// let Promise_all = Promise.any([p1, p2, p3])
let Promise_all = Promise.resolve(6)
let Promise_all = Promise.reject(new Error("Error"))
Promise_all.then((value)=>{
    console.log(value);
})
```

- **Async-Await.**
- The async keyword transforms a regular JavaScript function into an asynchronous function, causing it to return a Promise. The await keyword is used inside an async function to pause its execution and wait for a Promise to resolve before continuing.
- **JS code.**

```
async function Shaan(){
    var delhiWeather = new Promise((resolve, reject)=>{
        setTimeout(()=>{
            resolve("30 deg");
        },1000)
    })
    var bangalorWeather = new Promise((resolve, reject)=>{
        setTimeout(()=>{
            resolve("40 deg");
        },2000)
    })
    // delhiWeather.then(alert)
    // bangalorWeather.then(alert)
    console.log("Fetching Delhi Weather Please Wait...")
    var delhiW = await delhiWeather
    console.log("Fetching Delhi Weather:" + delhiW)
    console.log("Fetching Bangalor Weather Please Wait...")
    var bangalorW = await bangalorWeather
    console.log("Fetching Bangalor Weather:" + bangalorW)
    return[delhiW, bangalorW]
}
const cherry = async ()=>{
    console.log("Hey I am cherry and I am  waitting")
}
const main =  async() =>{
console.log("Welcome to the weather control room");
let a = await Shaan()
let b = await cherry()
}
main()
```

- **Fetch API.**
- The Fetch API provides an interface for fetching resources (including across the network). It is a more powerful and flexible replacement for XMLHttpRequest .

Use of free API.

- **Response header.**
- The response header is available in response header.
- **Request header.**
- To set a request header in fetch we can use the header option.
- **JS code.**

```
let p = fetch("https://goweather.herokuapp.com/weather/Dehradun")
p.then((value1)=>{
    console.log(value1.status)
    console.log(value1.ok)
    return value1.json()
}).then((value2)=>{
    console.log(value2)
})
```

- **Post request.**
- POST is used to send data to a server to create/update a resource. The data sent to the server with POST is stored in the request body of the HTTP request.

- **Json placeholder**
- **JS code.**

```
const createTodo = async (todo)=>{
let options  = {
    method: "POST",
    headers:{
        "Content-type":"application/json"
    },
    body: JSON.stringify({
        title: 'Shanelhai',
        body: 'bhai',
        userId: 1,
    }),
}
let p = await fetch('https://jsonplaceholder.typicode.com/posts',options)
let response = await p.json()
return response
}
const getTodo = async(id)=>{
    let response = await
fetch('https://jsonplaceholder.typicode.com/posts/1')
    let r = await response.json()
    return r
}
const mainFunc = async ()=>{
    let todo = {
        title: 'Shanelhai',
        body: 'bhai',
        userId: 1,
    }
    let todr = await createTodo(todo)
    console.log(todr)
    console.log(await getTodo(5))
}
mainFunc()
```

- **Error Handing in JS.**
  Error handling in JavaScript is the process of handling errors that occur during the execution of a JavaScript program. Errors can occur for a variety of reasons, such as incorrect syntax, missing variables, or unexpected input from the user.
- **JS code.**

```
setTimeout(()=>{
   console.log("A Hacking wifi....Please wait..")
},1000)
try{
   setTimeout(()=>{

      console.log(rahul)
   },1000)
}
catch(err){
   console.log("Shaan")
}
setTimeout(()=>{
   console.log("B Hacking wifi....Please wait..")
},2000)
setTimeout(()=>{
   console.log("C Hacking wifi....Please wait..")
},3000)
setTimeout(()=>{
   console.log("D Hacking wifi....Please wait..")
},4000)
```

- **The Error Object & Custom Errors.**
- The Error object in JavaScript is a built-in object that provides error information when an error occurs. It has two properties:
- name: The name of the error.
- message: A description of the error.
- **Custom Errors.**
- You can use this error class to construct your own error object prototype which is called as custom error. Custom errors can be constructed in two ways, which are: Class constructor extending error class. Function constructor inheriting error class.

- **JS code.**

```
try{
   let age = prompt("Enter the age:")// custom error.
   age = Number.parseInt(age)
   if(age>50){
   throw new ReferenceError("This is probably not true")
   }
}catch(error){
   console.log(error.name)
   console.log(error.message)
   console.log(error.stack)
}
console.log("The script is still running")
```

- **The Finally Clause.**
- The try… catch construct may have one more code clause finally.
- If it exists it run in all case.
- After try if there were no errors. After catch if there were errors.
- If there is a return in try, finally is executed just before the control returns to the outer code.
- **JS code.**

```
const f = ()=>{
try {
   let a = 0;
   //console.log(program)
   console.log("Program ran successfully")
   return                    }
catch (err){
   console.log("This is an error")
   console.log(p)                    }
finally{
   console.log("I am a good boy")
   // Close the file.
   // Exit the loop.
   // Write to the log file.          }              }
f()
console.log("End")
```