JAVASCRIPT

1. What are the different data types present in javascript?

There are 2 types of data types

1. Primitive : Used to store single values

* Boolean
* BigInt
* String
* Null (value is invalid)
* Undfined (Value is not assigned or defined)

1. Non-Primitive : Used to store multiple values

* Objects
* Functions
* Arrays

1. Explain Hoisting in javascript.

* Hoisting is the default behaviour of javascript where all the variable and function declarations are moved on top of there scope

Note: Only declarations are moved to the top not the initialization (eg : i = 9)

* //variable hoisting
* var x; // Declaration is hoisted to the top
* console.log(x); // undefined
* x = 5; // Initialization stays in place
* //functional hoisting
* greet();
* function greet(){
* console.log("this is for hoisting");
* }

1. Difference between “ == “ and “ === “ operators?

* Both are comparison operators. The difference between both the operators is that “==” is used to compare values whereas, “ === “ is used to compare both values and types.

1. Explain Type Coersion and Type Conversion in Javascript.

 **Type Coercion**: Automatic, implicit conversion by JavaScript, converts one datatype to another data type

let result = '5' + 1; // '51'

 **Type Conversion**: Explicit, manual conversion using functions or methods to change the type of a value.

let str = "123";

let num = Number(str); // 123

let int = parseInt(str); // 123

let float = parseFloat("123.45"); // 123.45

let num = 123;

let str = String(num); // "123"

let str2 = num.toString(); // "123"

1. Is javascript a statically typed or a dynamically typed language?

* It is a dynamically typed language, here the variables has no types

1. What is NaN property in JavaScript?

* Not a Number basically indicates that the value is not a legal number
* NaN converts the value to a number and then compares it.

1. What is an Immediately Invoked Function in JavaScript?

* IIFE is a type of a function which will be invoked as soon as declared

(function(){

    //something

      })()

* the first parenthesis () is when we declare a function it should have a name but this function is not having a name so we need to put
* () in the first, so we add the first set of parenthesis that tells the compiler that the function is not a function declaration, instead, it’s a function expression. and now the second parenthesis is for invoking a function.
* IIFE is particularly common in scenarios where you want to execute a block of code immediately without leaving behind any global variables or polluting the global scope. It's also frequently used in module patterns and in situations where you need to create closures for event handlers or callbacks to maintain state privacy.

1. Explain this keyword.

* this keyword depends upon the object that the function is property of.

var obj = {

    name:"saqib",

    userName:function(){

    console.log(this.name)

   }};

   obj.userName()

Note: Just remember what comes before the dot in keyword it's the property of that object.

1. Explain call(), apply() and, bind() methods.

* call() allows the objects to use the method or the function of another objects

function sayHello(){

    return "Hello " + this.name;

     }

     var obj = {name: "Sandy"};

    sayHello.call(obj);

    // Returns "Hello Sandy"

* Apply() :- similar to call but it takes arguments as an array.

function sayHello(greet,fname,lname){

  return greet + this.fname+ " "+ this.lname;

}

var obj = {

  fname : " saqib",

  lname: "ahmed"

};

const result  = sayHello.apply(obj,["hello"])

console.log(result)

**Bind()** : basically it creates a function with **this.** context, it will be usefull when we have to ensure the particular value (by using this) regardless how its called.

function introduce(greeting, punctuation) {

    return `${greeting}, I'm ${this.name}${punctuation}`;

}

const person = {

    name: "Bob"

};

// person is the new function

const introduceBob = introduce.bind(person, "Hi");

// added second parameter

console.log(introduceBob("!"));

const person = {

    name: "Bob"

};

function introduce(greeting, punctuation, age, job, country) {

    return `${greeting}, I'm ${this.name}. I'm ${age} years old, I work as a ${job}, and I'm from ${country}${punctuation}`;

}

// Bind the 'person' object and preset the first three arguments

const introduceBob = introduce.bind(person, "Hi", "!", 28);

// Call the bound function with the last two arguments

console.log(introduceBob("Software Developer", "USA"));

// Output: "Hi, I'm Bob. I'm 28 years old, I work as a Software Developer, and I'm from USA!"

1. What is the difference between exec () and test () methods in javascript?

* exce() is used to find the characters of a string it will return that character
* text = "saqib"
* result = /e/.exec(text)  // e ku search karo text me se
* test() : it will display the true or false whether the character is there or not
* let text = "Hello world!";
* let pattern = /Hello/g;
* let result = pattern.test(text);
* console.log(result);// it will give true

1. What is currying in JavaScript?

* Currying is a technique where a function is tranformed into sequence of functions and it will take single arguments
* function add(x){
* return function(y){
* return function (z){
* console.log(x+y+z);
* }
* }
* }
* const result = add(2)(2)(2)

1. Explain Scope and Scope Chain in javascript.

Scope is the accessability of variables or the functions across the various codes.

* **3 types of scopes**

* 1. **Global Scopes** : Variables can be access throughout the code.
  2. **Local / Functional Scope** : Any variables or functions declared inside a function have local/function scope, which means that all the variables and functions declared inside a function, can be accessed from within the function and not outside of it.
  3. **Block Scope** : Block scope is related to the variables declared using let and const. Variables declared with var do not have block scope. Block scope tells us that any variable declared inside a block { }, can be accessed only inside that block and cannot be accessed outside of it.
  4. **Scope Chain** : javascript tries to find the variable in local scope if it doesnot find it will try in outer scope again its not there then it will try to find in global scope . local to outer than global

var y = 29

function favFunction() {

  var x = 667;

  var anotherFavFunction = function () {

    console.log(x); // Does not find x inside anotherFavFunction, so looks for variable inside favFunction, outputs 667

  };

  var yetAnotherFavFunction = function () {

    console.log(y); // Does not find y inside yetAnotherFavFunction, so looks for variable inside favFunction and does not find              it, so looks for variable in global scope, finds it and outputs 24

  };

  anotherFavFunction();

  yetAnotherFavFunction();

}

favFunction();

1. Explain Lexical Scope and Closure ?

* Lexical Scope :In JavaScript, functions are lexically scoped, meaning they have access to variables from their outer (enclosing) scope.

1. function outer() {
2. var outerVariable = "I am outer";
3. function inner() {
4. console.log(outerVariable); // Accessing outerVariable from the outer scope
5. }
6. inner();
7. }
8. outer(); // Outputs: I am outer

* **Closure** : A closure is created when a function is defined inside another function, and the inner function has access to the outer function's variables (even after the outer function has finished executing).
* function outerFunction() {
* var outResult = "Hi am a function";
* function innerFunction() {
* console.log(outResult); //  accessing the variables from outside
* }
* return innerFunction(); // returns an inner function creating a closure
* }
* var result = outerFunction();

14. What are object prototypes?

* A prototype is a blueprint of an object. The prototype allows us to use properties and methods on an object even if the properties and methods do not exist on the current object.
* var arr = [];
* arr.push(2);
* console.log(arr); // Outputs [2]

**Note** : In the code above, as one can see, we have not defined any property or method called push on the array “arr” but the javascript engine does not throw an error.The reason is the use of prototypes. As we discussed before, Array objects inherit properties from the Array prototype.

15 . What is memoization ?

* Memoization is a form of storing a cache value if it's called again with same parameters
* function addTo256(num){
* return num + 256;
* }
* addTo256(20); // Returns 276
* addTo256(40); // Returns 296
* addTo256(20); // Returns 276

Note: Here in above example we have added called using argument 20 so it will use catch to display again , memoization is used for complex data to avaoid redendency.This is where memoization comes in, by using memoization we can store(cache) the computed results based on the parameters. If the same parameter is used again while invoking the function, instead of computing the result, we directly return the stored (cached) value.

16. What is the distinction between client-side and server-side JavaScript?

* Client server will make request using browser and javascript code and then server-side involves the execution of JavaScript code on a server in response to client requests.

17. What do you mean by BOM?

* Browser Object Model is used to interact with the browser, a browser's initial object is a window.

18. When do we use Constructor ?

* If we want to create multiple objects having similar properties and methods, constructor functions are used.
* function Person(name,age,gender){
* this.name = name;
* this.age = age;
* this.gender = gender;
* }
* var person1 = new Person("Vivek", 76, "male");
* console.log(person1);
* var person2 = new Person("Courtney", 34, "female");
* console.log(person2);

19. Explain Pass by value and reference

* + Pass by value : When we pass a primitive type (like numbers, strings, booleans, null, undefined, and symbols) Js create a copy and passes to a function without changing actual value
* function modifyValue(x) {
* x = x + 1;
* return x;
* }
* let num = 10;
* console.log(modifyValue(num)); // Outputs: 11
* console.log(num);               // Outputs: 10 (original value unchanged)
  + Pass by reference : When we pass an object (including arrays and functions) to a function, JavaScript passes a reference to that object, not a copy of the object itself. This means changes made to the object inside the function affect the original object.
* function modifyArray(arr) {
* arr.push(4);
* }
* let myArray = [1, 2, 3];
* modifyArray(myArray);
* console.log(myArray); // Outputs: [1, 2, 3, 4] (original array modified)