**1) Write an example of Anonymous function.**

**Ans:-** Anonymous function is usually not accessible after its initial creation.

Normal function definition:

**Example:-**

function hello() {

alert('Hello world');

}

hello();

**2) What will be the output of the following code?**

**var output = (function(x){**

**delete x;**

**return x;**

**})(0);**

**console.log(output);**

**:-**  0 (zero)

**3) Explain Prototype Design pattern in Javascript with an example.**

**:-** The Prototype pattern is useful is the initialization of business objects with values that match the default values in the database.

The prototype object holds the default values that are copied over into a newly created business object.

**Benefits**

* Prototyping is native to JavaScript
* Allows you to extend an existing object
* Facilitates code modularization
* Takes variables and functions out of the global namespace
* Functions load into memory one time only

**Drawbacks**

* Heavy use of the ‘this’ keyword can cause confusion
* Requires both a Constructor and a Function Prototype

**Example**

// The Constructor (Part 1)

// Define an object

// Accept parameters

var Bicycle = function(brand) {

    this.brand = brand;

};

// Function Prototype (Part 2)

// Add functionality and extend objects

Bicycle.prototype = {

    goForward: function (percent) {

        percent = ' Bicycle Moving forward at ' + percent + ' percent speed!';

        return percent;

    }

};

// Calling Prototype based code

// Resembles traditional OOP styles

// 'New Up' instances, then call methods on that instance

var bike = new Bicycle('ProdecoTech');

var speed = bike.goForward(100);

When you run this example code, you can see it outputs the result based on the data we pass in. We used the new keyword to create a new instance of a Bicycle, passing in the brand of ProdecoTech, which is a really cool electric bicycle brand. Then, we can call methods off of that new object we have. We demonstrate this by calling bike.goForward() passing in a 100 percent value. If all works correctly, we see ProdecoTech Bicycle Moving forward at 100 percent speed! output to the screen.

**4) Write a function to calculate Fibonacci Numbers ( Fibonacci numbers are a sequence**

**of numbers where each value is the sum of the previous two, starting with 0 and 1.**

**The first few values are 0, 1, 1, 2, 3, 5, 8, 13 ,…, ).**

**:-**

var fibonacci\_series = function (n)

{

if (n===1)

{

return [0, 1];

}

else

{

var s = fibonacci\_series(n - 1);

s.push(s[s.length - 1] + s[s.length - 2]);

return s;

}

};

console.log(fibonacci\_series(8));

**5) Give an example for destructuring an object or an array in ES6?**

**:-** Destructuring is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables. That is, we can extract data from arrays and objects and assign them to variables.

**Example :-**

Var introduction = [“Hello”, ”I”, “am”, “Lalit”]

Var greeting = introduction[0];

Var name = introduction[3];

Console.log(greeting); // Hello

Console.log(name); // Lalit

**5) Explain Closures in Javascript with an example?**

**:-** A  closure is an inner function that has access to the outer (enclosing) function's variables—scope chain.

Closures are important because they control what is and isn't in scope in a particular function, along with which variables are shared between sibling functions in the same containing scope.

**Example:-**

A function can access variables defined outside the function:

<button type="button" onclick="myFunction()">Click Me!</button>

<p id="demo"></p>

<script>

var a = 4;

function myFunction() {

document.getElementById("demo").innerHTML = a \* a;

}

</script>

**6) Explain different types of scoping in Javascript also explain the use-case of:**

**let, var and const**

**:-**

**Let:-**

* Variables declared with the let keyword can have Block Scope.
* Variables declared inside a block **{}** can not be accessed from outside the block:

{

Let x= 4;

}

* let allows you to declare variables that are limited to a scope of a [block](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/statements/block) statement, or expression on which it is used, unlike the [var](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/statements/var" \o "The var statement declares a variable, optionally initializing it to a value.) keyword, which defines a variable globally, or locally to an entire function regardless of block scope.

**Var:-**

* var declarations, wherever they occur, are processed before any code is executed. This is called hoisting, and is discussed further below.
* The scope of a variable declared with var is its current execution context*,* which is either the enclosing function or, for variables declared outside any function, global. If you re-declare a JavaScript variable, it will not lose its value.

var carName = "Volvo";

**const:-**

* The const declaration creates a read-only reference to a value. It does not mean the value it holds is immutable—just that the variable identifier cannot be reassigned.
* A constant cannot share its name with a function or a variable in the same scope.

const PI = 3.141592653589793;  
PI = 3.14;      // This will give an error  
PI = PI + 10;