**Ex. 1) Write a function called tripleAdd that is called like this, tripleAdd(10)(20)(30); And With a triple invocation, and all it does is it simply returns the total of all three numbers added together.**

Solution:  
function tripleAdd(num1) {

return function(num2) {

return function(num3) {

return num1 + num2 + num3;

};

};

}

tripleAdd(10)(20)(30); // returns 60

Note: tripleAdd function is known as a curried function. 

Ex. 2) another interesting curried function example:

function quadrupleAdd(num1) {

return function(num2) {

return function(num3, num4) {

return num1 + num2 + num3 + num4;

};

};

}

quadrupleAdd(10)(20)(30, 40); // returns 100

Q: What is IIFE and why are they used?  
  
Ans: IIFE stands for Immediately Invoked Function Expression. IIFE are self-executed or self-invoked functions, we don’t need to invoke them like regular JavaScript functions do. IIFE are executed right after its definition.  
  
The main reason to use IIFE is to preserve a private scope within a function. Inside your JavaScript code you want to make sure that you are not overriding any global variables. Sometimes in a file you will accidentally define a variable that overrides a global variable. For ex. the jQuery library always represented by the variable $ sign, if in your file you accidentally assigned the dollar sign variable to a different value because you forgot that jQuery uses it, then you will no longer have access to the jQuery library. We can solve this problem by using an IIFE. This way, you would wrap entire JavaScript file inside of IIFE and place all functionality inside of it. So if you overwrite a global variable in here like assigning value to $ variable, that will not harm jQuery’s $ variable. Because IIFE $ variable will not override the $ sign variable across our whole application.

Ex 1:

(function doubleNumber(num) {

return num \* 2;

})(10);

Output: 20

Ex 2:

(function() {

function getTotal(a, b) {

return a + b;

}

var $ = 'currency'; // this will not impact your jQuery $ variable

if (true) console.log('hello world');

})();

Output: 'hello world

**----------------------------------------------------------------------------------------------------------------------------------------**

**Que: What will be alerted to the screen when button number 5 is clicked, and why this certain piece of data is alerted?**

<script>

function createButtons() {

for (var i = 1; i <= 5; i++) {

var body = document.getElementsByTagName("BODY")[0];

var button = document.createElement("BUTTON");

button.innerHTML = 'Button ' + i;

button.onclick = function() {

alert('This is button ' + i);

}

body.appendChild(button);

}

}

createButtons();

</script>

Ans: **We will get alert with message ‘This is button 6’;** It is because when we click the button 5, that time our code has already done with for loop execution. And **value of ‘i’ will be 6 when we exit for loop**. Therefore on click of any button, we will get message ‘This is button 6’.

There are three ways through which we can solve this issue. And get the exact button number on clicking on it. As follows:

1. **Use IIFE within for loop**
2. **Call another function inside for loop and pass button and i’s value as an argument to that function**
3. **Use ‘let’ keyword instead of var keyword in for loop**
4. **Use IIFE within for loop**

**Ans:** We can use IIFE to overcome above issue. Through IIFE, we can store the current iteration value in one variable and through that variable we can display the correct button number.

<script>

function createButtons() {

for (var i = 1; i <= 5; i++) {

var body = document.getElementsByTagName("BODY")[0];

var button = document.createElement("BUTTON");

button.innerHTML = 'Button ' + i;

(function(num){

button.onclick = function() {

alert('This is button ' + num);

}

})(i)

body.appendChild(button);

}

}

createButtons();

</script>

**Note: In this example, our IIFE will create 5 instances for each iteration. And every iteration will store current value of ‘i’ using num variable. Thus if we click any of the button, that particular instance will get executed. So we will get the exact button number.**

**Remember, here also, value of ‘i’ will be 6 in the last, as we have seen in previous example.**

1. **Call another function inside for loop and pass button and i’s value as an argument to that function**

<script>

function createButtons() {

for (var i = 1; i <= 5; i++) {

var body = document.getElementsByTagName("BODY")[0];

var button = document.createElement("BUTTON");

button.innerHTML = 'Button ' + i;

getExactNumber(button, i)

body.appendChild(button);

}

}

createButtons();

function getExactNumber(button, num){

button.onclick = function() {

alert('This is button ' + num);

}

}

</script>

**Note: We are passing two arguments to getExactNumber function, to fire click event on particular button, we will use button parameter and to store the current button number, we will use num parameter.**

1. **Use ‘let’ keyword instead of var keyword in for loop**

**\*\*\*If we use let keyword instead of var, this will also solve the issue. It’s because let has block scope and var has function level scope. So let will display the exact iteration value**

**Que: What is closure? Code out an example of closure that work properly?**

**Ans:** Closure is an inner function that has access to the scope of an enclosing function.

**A closure has access to variables in 3 separate scopes;**  
1) Variables in its own scope

2) Variables in the scope of the outer function

3) Variables in the global scope

**The closure also has access to**

* Its own parameters
* Parameters of outer function(s)

Check below example, innerFunc is example of closure here. And it can access all the variables defined in its own scope, all variables defined in its outer function scope & global variables and it also has access of its own parameters and outer function parameters:

Example of closure:  
**const globalVariable = 'global var';**

**function outerFunc(param1) {**

**const variable1 = 'var one';**

**function innerFunc(param2) {**

**const variable2 = 'var two';**

**console.log('globalVariable: ', globalVariable);**

**console.log('variable1: ', variable1);**

**console.log('variable2: ', variable2);**

**console.log('param1: ', param1);**

**console.log('param2: ', param2);**

**}**

**innerFunc('Inner Param');**

**}**

**outerFunc('Outer Param');**

**Output:**

**globalVariable: global var**

**variable1: var one**

**variable2: var two**

**param1: Outer Param**

**param2: Inner Param**

**----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------**

**Que: What is ‘this’ keyword and how it is used?**

**Ans:**

**Answer:  There are 7 different cases where the value of this varies.**

1. **In the global context or inside a function ‘this’ refers to the** window object. Also in closure function, **‘this’** refers to window object.
2. **While executing a function in the context of an object (in short if we are within object method), the object becomes the value of ‘this’.**
3. **Inside IIFE (immediate invoking function Expression) “**this” refers to window object.
4. **Inside** setTimeout **function, the value of ‘this’ is the** window object**.**
5. **If you use a constructor function (using ‘new’ keyword to instantiate the object), the value of ‘this’ will refer to the newly created object.**
6. **You can set the value of this to any arbitrary object by passing the object as the first parameter of bind, call or apply.**
7. **For DOM event handler, value of ‘this’ will be the element that fired the event. Suppose button click event is there, in that case this will refer to the button.**

**---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------**

**Que: Describe what variable and function hoisting is? And how it works?**

**Ans: Refer “JS techniques file; search string will be “JavaScript Hoisting”**

**----------------------------------------------------------------------------------------------------------------------------------------------------------**

**Que: What will be the output below code will generate?**

**var myCar = {**

**color: "Blue",**

**logColor: function() {**

**var self = this;**

**console.log("In logColor - this.color: " + this.color);**

**console.log("In logColor - self.color: " + self.color);**

**(function() {**

**console.log("In IIFE - this.color: " + this.color);**

**console.log("In IIFE - self.color: " + self.color);**

**})();**

**}**

**};**

**myCar.logColor();**  
  
Output:  
**In logColor - this.color: Blue // this referencing to myCar**

**In logColor - self.color: Blue // self referencing to myCar**

**In IIFE - this.color: undefined // this referencing to global/ window object**

**In IIFE - self.color: Blue // self referencing to myCar  
  
\*\*\*\*Note:**

1. **if you have IIFE method inside object method, or if you have nested method inside object method, in both cases ‘this’ will be referencing to global object.**
2. **Since in above code, there is no color variable in global object, we get undefined in IIFE for ‘this.color’ statement. If you define color variable in global object, that value will be shown in IIFE.  
   --------------------------------------------------  
     
   II ) Another example of ‘this’:**

**Ex:   
this.globalProp = 'foo'**

**var myCar = {**

**color: "Blue",**

**logColor: function() {**

**var self = this;**

**console.log("In logColor - this.color: " + this.color);**

**console.log("In logColor - self.color: " + self.color);**

**var x = function(){**

**console.log("In X - this.color: " + this.color);**

**console.log("In X - self.color: " + self.color);**

**console.log("In X - this.globalProp: " + this.globalProp);**

**}**

**x()**

**}**

**};**

**myCar.logColor();**

**Output:**

***In logColor - this.color: Blue***

***In logColor - self.color: Blue***

***In X - this.color: undefined***

***In X - self.color: Blue***

***In X - this.globalProp: foo***

**----------------------------------------------------------------------------------------------------------------------------------------------------------**

**Que: What is the difference between == and ===**

**Ans:**

**== stands for equals and === stands for strict equals.  
== Checks just values, it tests values are similar or not;   
if values are similar; in such cases == will return true.**

**Whereas === checks values and types are similar or not. If values are similar but their types are different; in such cases === will return false.**  
For Ex:  
**console.log(7 == '7'); // true**

**console.log(7 === '7'); // false**

**------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------**

**Que: What will be the output below code will produce?**

Ex 1) var num = 50;

function logNumber() {

console.log(num);

var num = 100;

}

logNumber();  
  
Output: undefined  
  
  
Ex 2)   
let num = 50;

function logNumber() {

console.log(num);

let num = 100;

}

logNumber();

Output: ReferenceError: num is not defined

Variables declared with let keyword; does not allow their values to be hoisted at the top like var keyword. If we try to use let variables before its declaration we will get error

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Que: What does using ‘strict mode’ do to your code?**

**OR**

**What are the benefits of using ‘strict mode’  
Ans:   
The main purpose of using use strict is to enforce stricter parsing and error handling in your code.**

**Some of the main important things that use strict does are:  
a) Prevents the use of global variables:  
For ex; check below code**

**city = "London";**

**console.log(city);**

**In above code, city variable is defined without using var,let or const; thus this variable will be treated as global variable and code will run fine**

**But**

**If we use ‘use strict’ at the beginning of the file and we run the same code; we will get the error “city is not defined”  
  
It’s because stricter mode does not allow you to define any variable without var, let or const. This thing can save us from many unnecessary/unexpected future bugs.**

**b) All parameter names must be unique:**

**Strict mode enforces us to assign unique names to all parameters.  
For Ex:**

**function myFunc(a, a, b) {**

**console.log(a, a, b);**

**}**

**myFunc(1, 2, 3);  
  
Output: 2 2 3**

**Above code will run fine if we are not using strict mode. And second parameter “a" will override first parameter ‘a’. Hence code is displaying 2 twice in output. Since javascript does not give us error for such overriding, it is difficult to find such errors in long run.**

**But if we use ‘use strict’; then it will not allow us assigning similar name to parameters. In strict mode, it will give us error: argument name clash.  
  
C) Throws an error if we try to delete those properties on object that are not deletable:**

**For Ex:   
delete Object.prototype;  
  
we cannot delete prototype property of Object that is built in JavaScript. But we will not get any error in non-strict mode. Since we are not getting any error, one might think that this statement is successfully executed. But this is incorrect.**

**If we are using strict mode, in that case it will give us an error: cannot delete property. Such errors can be useful in long run.**

**------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------**

**Que: How can we turn below function into curry function?**function getProduct(num1, num2) {

return num1 \* num2;

}

getProduct(10,20)

**Ans:**function getProduct(num1) {

return function (num2) {

return num1 \* num2;

}

}

getProduct(10)(20) **--------------------------------------------------------------------------------------------------------------**

**Que: Write a function that keeps track of how many times it was called and returns that number? All functionality should be inside of the function – none outside.**

**Ans:**

**function myFunc() {**

**let count = 0;**

**return function() { /\* THIS IS CLOSURE \*/**

**count++;**

**return count;**

**};**

**}**

**const instanceOne = myFunc();**

**console.log(instanceOne());**

**console.log(instanceOne());**

**console.log(instanceOne());**

**console.log(instanceOne());  
  
output: 4**

**\*\*\*\*Note: We cannot call myFunc() function (outer function) to get the required output. It is because if we invoke outer function myFunc(); it will simply return the closure function. It happens because in outer function, we are returning closure. And our required value is within closure. Thus we will have to instantiate our outer function and then we need to call that instance.**

**------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------**

**Que: What will be the value we will get in console for variable x and y in below code?**

**(function() {**

**var x = y = 200;**

**console.log(x)**

**console.log(y)**

**})();**

**console.log('y: ', y);**

**console.log('x: ', x);  
output:**

***200***

***200  
y: 200***

***error: x is not defined..*  
  
\*\*Description: First statement of IIFE is important;**

**var x = y = 200; JavaScript will evaluate this code as follows:  
  
JavaScript will evaluate the code from right to left as follows:**

**y = 200;**

**x = y; /\* Now x equals to 200 \*/  
var x; /\* Now x has function scope \*/  
  
Since y does not have var or let or const attached, JavaScript will treat y as a global variable. And since x has var attached, it is local to IIFE function. Since var has function scope, we can access it inside IIFE, but cannot access outside of IIFE. So we are getting x is not defined error here.  
  
If we remove var keyword from x, then x becomes global. In that case, x can be accessible outside the IIFE, like y.   
  
we will get the same output if we put all the code in regular function instead of IIFE. It is happening all because of var keyword and function scoping.   
  
If we use strict mode inside IIFE, we will get the error that “y is not defined”. It is because strict mode does not allow global variables.**

**--------------------------------------------------------------------------------------------------------------------------------------------------------------------  
  
Que: Describe the javascript call() and apply() methods? Explain how are they different?**

**Ans: “*We often use call and apply methods to reuse method declared within one object into another object.*” And if we are using call or apply methods; those methods will change the context of ‘this’.  
Check below example, we can see car2 and car3 does not have getCarDescription() method, still they can access that method using call and apply methods. We can say that we no need to declare getCarDescription method for every object; this reduce our coding and also it saves memory.   
  
Ex:  
const car1 = {**

**brand: 'Porsche',**

**getCarDescription: function(cost, year, color) {**

**console.log(`This car is a ${this.brand}. The price is $${cost}. The year is ${year}. The color is ${color}.\n`);**

**}**

**};**

**const car2 = {**

**brand: 'Lamborghini'**

**};**

**const car3 = {**

**brand: 'Ford'**

**};**

**car1.getCarDescription(80000, 2010, 'blue');**

**car1.getCarDescription.call(car2, 200000, 2013, 'yellow');**

**car1.getCarDescription.apply(car3, [35000, 2012, 'black']);  
  
Output:  
This car is a Porsche. The price is $80000. The year is 2010. The color is blue.**

**This car is a Lamborghini. The price is $200000. The year is 2013. The color is yellow.**

**This car is a Ford. The price is $35000. The year is 2012. The color is black.**

**Note: The difference between call and apply is; through apply we pass arguments to the function as an array, but through call method we need to pass all those arguments individually. If there are multiple arguments then apply is the convenient way to use.**

**------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------**

**Que: Take a look at the code shown. What will list2 contain when it is logged out to console?**

const list1 = [1, 2, 3, 4, 5];

const list2 = list1;

list1.push(6, 7, 8); /\* pushing 3 elements at one go \*/

console.log(list2);  
  
Output: [ 1, 2, 3, 4, 5, 6, 7, 8 ]

typeof array returns object. When we assign one object with another, it creates shallow copy of first object. In shallow copy, newer object copies address of original object. If we change original object then that change will be reflected in newer object and vice-versa. This happens because both objects are pointing to the same memory location. And since both objects are at similar location, change in one object automatically updates another object.   
  
That is why; list1 and list2 will remain same at every change. If we change anything in list1, that change will be reflected in list2 as well and vice-versa.

----------------------------------------------------------------------------------------------------------------------------------------------------------

**Que: Singly or Doubly Invoked function:**

**Build a function called getTotal() that can be invoked in two different ways as follows:  
function getTotal() {**

**// code here...**

**} *getTotal(10,20); // returns addition i.e. 10 + 20 = 30  
getTotal(10)(20); // returns addition i.e. 10 + 20 = 30***

***Either way the function is called, you should still get the same result. The function should simply return to us the total of the two nos. added together.***

**Ans:  
function getTotal(num1) {**

**if (arguments.length === 2) {**

**return num1 + arguments[1];**

**}**

**else if (arguments.length === 1) {**

**return function(num2) {**

**return num1 + num2;**

**};**

**}**

**}**

**console.log(getTotal(10, 20));**

**console.log(getTotal(10)(200));**

**Output:**

**30**

**210**

**------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------**

**Que: What datatypes are allowed and not allowed in JSON?   
Ans:** Except undefined and function type, all types are allowed to use in JSON. **For Ex: Valid json sytax is as follows:  
const myJsonObj = {**

**"myString": "hello world",**

**"myNumber": 12345.6789,**

**"myNull": null,**

**"myBoolean": true,**

**"myArray": [20, 30, "orange"],**

**"myObject": {**

**"name": "Sam",**

**"age": 30**

**}**

**};  
  
All propertyNames and string values needs to be in double quotes. Remember single quotes are not allowed in JSON.**

**------------------------------------------------------------------------------------------------------------------------------------------------**

**Q: What will be the output?**

function logNumbers() {

console.log(1);

setTimeout(function(){console.log(2)}, 1000);

setTimeout(function(){console.log(3)}, 0);

console.log(4);

}

logNumbers();

**Output:**

**1**

**4**

**3**

**2  
  
Note:** 3rd statement that has setTimeout with zero second delay will be passed to event loop. All callbacks are going through the event loop and since setTimeout uses callback, that setTimeout will be passed to event loop. Hence fourth console statement will be executed before that setTimeout statement.

**--------------------------------------------------------------------------------------------------------------------------------------**

**Q: Describe 3 different ways to create an Object?**Ans:

**// 1. object literal syntax**

const student = {

grade: 12,

gradePointAverage: 3.7,

classes : ["English", "Algebra", "Chemistry"],

getClasses : function() {

return this.classes;

}

};

**// 2. new keyword & Object constructor**

const student = new Object();

student.grade = 12;

student.gradePointAverage = 3.7;

student.classes = ["English", "Algebra", "Chemistry"];

student.getClasses = function() {

return this.classes;

};

**// 3. constructor function**

function Car(color, brand, year) {

this.color = color;

this.brand = brand;

this.year = year;

}

Car.prototype.getColor = function() {

return this.color;

};

const carlysCar = new Car('blue', 'ferarri', 2015);

const jimsCar = new Car('red', 'tesla', 2014);

console.log(carlysCar.getColor());

console.log(jimsCar.getColor());