Question 1. Difference between `undefined` and `not defined` in JavaScript

> In JavaScript if you try to use a variable that doesn't exist and has not been declared, then JavaScript will throw an error `var name is not defined` and the script will stop execute thereafter. But If you use `typeof undeclared_variable` then it will return `undefined`.

Before starting further discussion let's understand the difference between declaration and definition.

`var x` is a declaration because you are not defining what value it holds yet, but you are declaring its existence and the need of memory allocation.

```
var x; // declaring x
console.log(x); //output: undefined
```

`var x = 1` is both declaration and definition (also we can say we are doing initialisation), Here declaration and assignment of value happen inline for variable x, In JavaScript every variable declaration and function declaration brings to the top of its current scope in which it's declared then assignment happen in order this term is called `hoisting`.

> A variable that is declared but not define and when we try to access it, It will result `undefined`.

```
var x; // Declaration
if(typeof x === 'undefined') // Will return true
```

> A variable that neither declared nor defined when we try to reference such variable then It result `not defined`.

```
console.log(y);  // Output: ReferenceError: y is not defined
```

Ref Link:

http://stackoverflow.com/questions/20822022/javascript-variable-definition-declaration

Question 2. What will be the output of the following code?

```
var y = 1;
if (function f() {}) {
  y += typeof f;
}
console.log(y);
```

> Above code would give output `1undefined`. If condition statement evaluate using `eval` so `eval(function f() {})` which return `function f() {}` which is true so inside if statement code execute. `typeof f` return undefined because if statement code execute at run time, so statement inside `if` condition evaluated at run time.

```
var k = 1;
if (1) {
  eval(function foo() {});
  k += typeof foo;
}
console.log(k);
```

Above code will also output `1undefined`.

```
var k = 1;
if (1) {
  function foo() {};
  k += typeof foo;
}
console.log(k); // output 1function
```

Question 3. What is the drawback of creating true private in JavaScript?

> One of the drawback of creating a true private method in JavaScript is that they are very memory inefficient because a new copy of the method would be created for each instance.

```
var Employee = function (name, company, salary) {
  this.name = name || "";       //Public attribute default value is null
  this.company = company || ""; //Public attribute default value is null
  this.salary = salary || 5000; //Public attribute default value is null

  // Private method
  var increaseSalary = function () {
    this.salary = this.salary + 1000;
  };

  // Public method
  this.dispalyIncreasedSalary = function() {
    increaseSalary();
    console.log(this.salary);
  };
};

// Create Employee class object
var emp1 = new Employee("John","Pluto",3000);
// Create Employee class object
var emp2 = new Employee("Merry","Pluto",2000);
```

```
// Create Employee class object
var emp3 = new Employee("Ren","Pluto",2500);
```

Here each instance variable `emp1`, `emp2`, `emp3` has own copy of increaseSalary private method.

So as recommendation don't go for a private method unless it's necessary.

Question 4. What is “closure” in javascript? Provide an example?

> A closure is a function defined inside another function (called parent function) and has access to the variable which is declared and defined in parent function scope.

The closure has access to variable in three scopes:

- Variable declared in his own scope
- Variable declared in parent function scope
- Variable declared in global namespace

```
var globalVar = "abc";

// Parent self invoking function
(function outerFunction (outerArg) { // begin of scope outerFunction
  // Variable declared in outerFunction function scope
  var outerFuncVar = 'x';
  // Closure self-invoking function
  (function innerFunction (innerArg) { // begin of scope innerFunction
    // variable declared in innerFunction function scope
    var innerFuncVar = "y";
    console.log(
      "outerArg = " + outerArg + "\n" +
      "outerFuncVar = " + outerFuncVar + "\n" +
      "innerArg = " + innerArg + "\n" +
      "innerFuncVar = " + innerFuncVar + "\n" +
      "globalVar = " + globalVar);
  // end of scope innerFunction
  })(5); // Pass 5 as parameter
// end of scope outerFunction
})(7); // Pass 7 as parameter
```

`innerFunction` is closure which is defined inside `outerFunction` and has access to all variable which is declared and defined in outerFunction scope. In addition to this function defined inside function as closure has access to variable which is declared in `global namespace`.

Output of above code would be:

```
outerArg = 7
outerFuncVar = x
innerArg = 5
```

```
innerFuncVar = y
globalVar = abc
```

Question 5. Write a mul function which will properly when invoked as below syntax.

```
console.log(mul(2)(3)(4)); // output : 24
console.log(mul(4)(3)(4)); // output : 48
```

> Below is code followed by an explanation how it works:

```
function mul (x) {
  return function (y) { // anonymous function
    return function (z) { // anonymous function
      return x * y * z;
    };
  };
}
```

Here `mul` function accept the first argument and return anonymous function which take the second parameter and return anonymous function which take the third parameter and return multiplication of arguments which is being passed in successive

In Javascript function defined inside has access to outer function variable and function is the first class object so it can be returned by function as well and passed as argument in another function.

- A function is an instance of the Object type
- A function can have properties and has a link back to its constructor method
- Function can be stored as variable
- Function can be pass as a parameter to another function
- Function can be returned from function

Question 6. How to empty an array in JavaScript?

For instance:

```
var arrayList =  ['a', 'b', 'c', 'd', 'e', 'f'];
```

How can we empty above array?

> There are a couple of ways by which we can empty an array, So let's discuss all the possible way by which we can empty an array.

Method 1

```
arrayList = [];
```

Above code will set the variable `arrayList` to a new empty array. This is recommended if you don't have **references to the original array** `arrayList` anywhere else because It will actually create a new empty array. You should be careful with this way of empty the array, because if you have referenced this array from another variable, then the original reference array will remain unchanged, Only use this way if you have only referenced the array by its original variable `arrayList`.

For Instance:

```
var arrayList = ['a', 'b', 'c', 'd', 'e', 'f']; // Created array
var anotherArrayList = arrayList;  // Referenced arrayList by another variable
arrayList = []; // Empty the array
console.log(anotherArrayList); // Output ['a', 'b', 'c', 'd', 'e', 'f']
```

Method 2

```
arrayList.length = 0;
```

Above code will clear the existing array by setting its length to 0. This way of empty the array also update all the reference variable which pointing to the original array. This way of empty the array is useful when you want to update all the another reference variable which pointing to `arrayList`.

For Instance:

```
var arrayList = ['a', 'b', 'c', 'd', 'e', 'f']; // Created array
var anotherArrayList = arrayList;  // Referenced arrayList by another variable
arrayList.length = 0; // Empty the array by setting length to 0
console.log(anotherArrayList); // Output []
```

Method 3

```
arrayList.splice(0, arrayList.length);
```

Above implementation will also work perfectly. This way of empty the array will also update all the references of the original array.

```
var arrayList = ['a', 'b', 'c', 'd', 'e', 'f']; // Created array
var anotherArrayList = arrayList;  // Referenced arrayList by another variable
```

```
arrayList.splice(0, arrayList.length); // Empty the array by setting length to 0
console.log(anotherArrayList); // Output []
```

Method 4

```
while(arrayList.length) {
  arrayList.pop();
}
```

Above implementation can also empty the array. But not recommended to use often.

Question 7. How to check if an object is an array or not?

> The best way to find whether an object is instance of a particular class or not using `toString` method from `Object.prototype`

```
var arrayList = [1 , 2, 3];
```

One of the best use cases of type checking of an object is when we do method overloading in JavaScript. For understanding this let say we have a method called `greet` which take one single string and also a list of string, so making our `greet` method workable in both situation we need to know what kind of parameter is being passed, is it single value or list of value?

```
function greet(param) {
  if() {
    // here have to check whether param is array or not
  }
  else {
  }
}
```

However, in above implementation it might not necessary to check type for array, we can check for single value string and put array logic code in else block, let see below code for the same.

```
 function greet(param) {
   if(typeof param === 'string') {
   }
   else {
     // If param is of type array then this block of code would execute
   }
 }
```

Now it's fine we can go with above two implementations, but when we have a situation like a parameter can be `single value`, `array`, and `object` type then we will be in trouble.

Coming back to checking type of object, As we mentioned that we can use `Object.prototype.toString`

```
if(Object.prototype.toString.call(arrayList) === '[object Array]') {
  console.log('Array!');
}
```

If you are using `jQuery` then you can also used jQuery `isArray` method:

```
if($.isArray(arrayList)) {
  console.log('Array');
} else {
  console.log('Not an array');
}
```

FYI jQuery uses `Object.prototype.toString.call` internally to check whether an object is an array or not.

In modern browser, you can also use:

```
Array.isArray(arrayList);
```

`Array.isArray` is supported by Chrome 5, Firefox 4.0, IE 9, Opera 10.5 and Safari 5

Question 8. What will be the output of the following code?

```
var output = (function(x) {
  delete x;
  return x;
})(0);

console.log(output);
```

> Above code will output `0` as output. `delete` operator is used to delete a property from an object. Here `x` is not an object it's **local variable**. `delete` operator doesn't affect local variable.

Question 9. What will be the output of the following code?

```
var x = 1;
var output = (function() {
  delete x;
  return x;
```

```
})();

console.log(output);
```

> Above code will output `1` as output. `delete` operator is used to delete property from object. Here `x` is not an object it's **global variable** of type `number`.

Question 10. What will be the output of the following code?

```
var x = { foo : 1};
var output = (function() {
  delete x.foo;
  return x.foo;
})();

console.log(output);
```

> Above code will output `undefined` as output. `delete` operator is used to delete a property from an object. Here `x` is an object which has foo as a property and from self-invoking function we are deleting foo property of object `x` and after deletion we are trying to reference deleted property `foo` which result `undefined`.

Question 11. What will be the output of the following code?

```
var Employee = {
  company: 'xyz'
}
var emp1 = Object.create(Employee);
delete emp1.company
console.log(emp1.company);
```

> Above code will output `xyz` as output. Here `emp1` object got company as **prototype** property. delete operator doesn't delete prototype property.

`emp1` object doesn't have **company** as its own property. you can test it `console.log(emp1.hasOwnProperty('company')); //output : false` However, we can delete company property directly from `Employee` object using `delete Employee.company` or we can also delete from `emp1` object using `__proto__` property `delete emp1.__proto__.company`.

Question 12. What is `undefined x 1` in JavaScript

```
var trees = ["redwood", "bay", "cedar", "oak", "maple"];
delete trees[3];
```

when you run above code and do `console.log(trees);` in chrome developer console then you will get `["redwood", "bay", "cedar", undefined × 1, "maple"]` and when you run above code in Firefox browser console then you will get `["redwood", "bay", "cedar", undefined, "maple"]` so from these it's cleared that chrome has its own way of displaying uninitialized index in array. But when you check `trees[3] === undefined` in both of the browser you will get similar output as `true`.

Note: Please remember you need not check for uninitialized index of array in `trees[3] === 'undefined × 1'` it will give an error, Because `'undefined × 1'` this is just way of displaying uninitialized index of array in chrome.

Question 13. What will be the output of the following code?

```
var trees = ["xyz", "xxxx", "test", "ryan", "apple"];
delete trees[3];
console.log(trees.length);
```

> Above code will output `5` as output. When we used `delete` operator for deleting an array element then, the array length is not affected from this. This holds even if you deleted all the element of array using `delete` operator.

So when delete operator removes an array element that deleted element is not longer present in array. In place of value at deleted index `undefined x 1` in **chrome** and `undefined` is placed at the index. If you do `console.log(trees)` output `["xyz", "xxxx", "test", undefined × 1, "apple"]` in Chrome and in Firefox `["xyz", "xxxx", "test", undefined, "apple"]`.

Question 14. What will be the output of the following code?

```
var bar = true;
console.log(bar + 0);
console.log(bar + "xyz");
console.log(bar + true);
console.log(bar + false);
```

> Above code will output `1, "truexyz", 2, 1` as output. General guideline for addition of operator:

- Number + Number -> Addition
- Boolean + Number -> Addition
- Number + String -> Concatenation
- String + Boolean -> Concatenation
- String + String -> Concatenation

Question 15. What will be the output of the following code?

```
var z = 1, y = z = typeof y;
console.log(y);
```

> Above code will output `undefined` as output. According to `associativity` rule operator with the same precedence are processed based on their associativity property of operator. Here associativity of the assignment operator is `Right to Left` so first `typeof y` will evaluate first which is `undefined` and assigned to `z` and then `y` would be assigned the value of z.

Question 16. What will be the output of the following code?

```
// NFE (Named Function Expression
var foo = function bar() { return 12; };
typeof bar();
```

> Above code will output `Reference Error` as output. For making above code work you can re-write above code as follow:

Sample 1

```
var bar = function() { return 12; };
typeof bar();
```

or

Sample 2

```
function bar() { return 12; };
typeof bar();
```

The function definition can have only one reference variable as a function name, In above code **sample 1** bar is reference variable which is pointing to `anonymous function` and in **sample 2** function definition is name function.

```
var foo = function bar() {
  // foo is visible here
  // bar is visible here
  console.log(typeof bar()); // Work here :)
};
// foo is visible here
// bar is undefined here
```

Question 17. What is the difference between declaring a function in below format?

```
var foo = function() {
  // Some code
};
```

```
function bar() {
  // Some code
};
```

The main difference is function `foo` is defined at `run-time` whereas function `bar` is defined at parse time. For understanding It in better way let see below code :

```
// Run-Time function declaration
<script>
  foo(); // Call foo function here, It will give an error
  var foo = function() {
    console.log("Hi I am inside Foo");
  };
</script>
```

```
// Parse-Time function declaration
<script>
bar(); // Call bar function here, It will not give an Error
function bar() {
  console.log("Hi I am inside Foo");
};
</script>
```

The another advantage of first-one way of declaration that you can declare function based on certain condition for example:

```
<script>
if(testCondition) { // If testCondition is true then
  function foo() {
    console.log("inside Foo with testCondition True value");
  };
 } else {
  function foo() {
    console.log("inside Foo with testCondition false value");
  };
 }
 </script>
```

But If you try to run similar code in the following format, it would give an error

```
<script>
if(testCondition) {// If testCondition is true then
   var foo = function() {
      console.log("inside Foo with testCondition True value");
   };
 } else {
    var foo = function() {
      console.log("inside Foo with testCondition false value");
   };
 }
 </script>
```

Question 18. what is function hoisting in JavaScript?

Function Expression

```
 var foo = function foo() {
     return 12;
 };
```

> In JavaScript variable and functions are `hoisted`. Let's take function `hoisting` first. Basically, the JavaScript interpreter looks ahead to find all the variable declaration and hoists them to the top of the function where it's declared. For Example:

```
foo(); // Here foo is still undefined
var foo = function foo() {
  return 12;
};
```

Above code behind the scene look something like below code:

```
var foo = undefined;
foo(); // Here foo is undefined
foo = function foo() {
  // Some code stuff
}
```

```
var foo = undefined;
foo = function foo() {
  // Some code stuff
}
foo(); // Now foo is defined here
```

Question 19. What will be the output of the following code?

```
var salary = "1000$";

(function () {
  console.log("Original salary was " + salary);

  var salary = "5000$";

  console.log("My New Salary " + salary);
})();
```

> Above code will output: `undefined, 5000$`. JavaScript has hoisting concept where newbie gets tricked. In above code, you might be expecting `salary` to retain it values from outer scope until the point that `salary` was re-declared in the inner scope. But due to `hoisting` salary value was `undefined` instead. To understand it better have a look of the following code, here `salary` variable is hoisted and declared at the top in function scope and while doing console.log it's result `undefined` and after that it's been redeclared and assigned `5000$`.

```
var salary = "1000$";

(function () {
  var salary = undefined;
  console.log("Original salary was " + salary);

  salary = "5000$";

  console.log("My New Salary " + salary);
})();
```

Question 20. What is the `instanceof` operator in JavaScript? what would be the output of the following code?

```
function foo() {
  return foo;
}
new foo() instanceof foo;
```

> `instanceof` operator checks the current object and return true if the object is of the specified type.

For Example:

```
var dog = new Animal();
dog instanceof Animal; // Output : true
```

Here `dog instanceof Animal` is true since `dog` inherits from `Animal.prototype`

```
var name = new String("xyz");
name instanceof String; // Output : true
```

Here `name instanceof String` is true since `name` inherits from `String.prototype`. Now let's understand the working of the following code

```
function foo() {
  return foo;
}
new foo() instanceof foo;
```

Here function `foo` is returning `foo` which is again pointer to function `foo`

```
function foo() {
  return foo;
}
var bar = new foo();
// here bar is pointer to function foo() {return foo}.
```

So the `new foo() instanceof foo` return `false`;

Ref Link: http://stackoverflow.com/questions/2449254/what-is-the-instanceof-operator-in-javascript

Question 21. If we have JavaScript associative array as below code:

```
var counterArray = {
  A : 3,
  B : 4
};
counterArray["C"] = 1;
```

How we will calculate length of the above associative array `counterArray`

> There are no built-in function and property available to calculate length of associative array object. However, there are ways by which we can calculate the length of associative array object, In addition to this we can also extend `Object` by adding method or property on prototype for calculating length but extending object might break enumeration in various libraries or might create cross-browser issue, so it's not recommended unless it's necessary. There is various ways by which we can calculate length.

`Object` has `keys` method which can we used to calculate the length of object.

```
Object.keys(counterArray).length; // Output 3
```

We can also calculate length of object by iterating through the object and by doing a count of own property of object.

```
function getSize(object) {
  var count = 0;
  for(key in object) {
    // hasOwnProperty method check own property of object
    if(object.hasOwnProperty(key)) count++;
  }
  return count;
}
```

> We can also add `length` method directly on `Object` see below code.

```
Object.length = function() {
  var count = 0;
  for(key in object) {
    // hasOwnProperty method check own property of object
    if(object.hasOwnProperty(key)) count++;
  }
  return count;
}
//Get the size of any object using
console.log(Object.length(counterArray));
```

Bonus: We can also use `Underscore` (recommended, as it's lightweight) to calculate object length.

Question 22. Difference between `Function`, `Method` and `Constructor` calls in JavaScript.

If your are familiar with Object-oriented programming, More likely familiar to thinking of functions, methods, and class constructors as three separate things. But In JavaScript, these are just three different usage patterns of one single construct.

functions : The simplest usages of function call:

```
function helloWorld(name) {
  return "hello world, " + name;
}

helloWorld("JS Geeks"); // "hello world JS Geeks"
```

Methods in JavaScript are nothing more than object properties that reference to a function.

```
var obj = {
  helloWorld : function() {
    return "hello world, " + this.name;
  },
  name: 'John Carter'
}
obj.helloWorld(); // // "hello world John Carter"
```

Notice how `helloWorld` refer to `this` properties of obj. Here it's clear or you might have already understood that `this` gets bound to `obj`. But the interesting point that we can copy a reference to the same function `helloWorld` in another object and get a difference answer. Let see:

```
var obj2 = {
  helloWorld : obj.helloWorld,
  name: 'John Doe'
}
obj2.helloWorld(); // "hello world John Doe"
```

You might be wonder what exactly happens in a method call here. Here we call the expression itself determine the binding of this `this`, The expression `obj2.helloWorld()` looks up the `helloWorld` property of obj and calls it with receiver object `obj2`.

The third use of functions is as constructors. Like function and method, `constructors` are defined with function.

```
function Employee(name, age) {
  this.name = name;
  this.age = age;
}

var emp1 = new Employee('John Doe', 28);
emp1.name; // "John Doe"
emp1.age; // 28
```

Unlike function calls and method calls, a constructor call `new Employee('John Doe', 28)` create a brand new object and passes it as the value of `this`, and implicitly returns the new object as its result.

The primary role of the constructor function is to initialize the object.

Question 23. What would be the output of the following code?

```
function User(name) {
  this.name = name || "JsGeeks";
```

```
}

var person = new User("xyz")["location"] = "USA";
console.log(person);
```

The output of above code would be `USA`. Here `new User("xyz")` creates a brand new object and created property `location` on that and `USA` has been assigned to object property location and that has been referenced by the person.

Let say `new User("xyz")` crated a object called `foo` and returned now `foo["location"]` would be assigned value as `USA` and `person` is referencing to `foo["location"]`.

Question 24. Suggest your question!

Question 25. What is the difference between a method and a function in javascript?

A function is a piece of code that is called by name and function itself not associated with any object and not defined inside any object. It can be passed data to operate on (i.e. parameter) and can optionally return data (the return value).

```
// Function definition
function myFunc() {
  // Do some stuff;
}

// Calling function
myFunc();
```

Here myFunc() function call is not associated with object hence not invoked through any object.

A function can be self-invoking anonymous function or named self-invoking function

```
// Named Self-invoking Function
(function myFunc() {
  // Do some stuff;
})();

// Anonymous Self-invoking Function
(function() {
  // Do some stuff;
})();
```

In a case of named self-invoking anonymous function or anonymous self-invoking function, there is no need of call function explicitly.

A method is a piece of code that is called by name and that is associated with the object. In most respects it is identical to function call except for some key difference:

- It is implicitly passed for the object for which it was called.
- It is able to operate on data that is contained within the class (remembering that an object is an instance of a class- the class is the definition, the object is an instance of that)
- Method call is always associated with object

```
var methodObject = {
  attribute: "xyz",
  display: function () {  // Method
    console.log(this.attribute);
  }
};

// Call method
methodObject.display();
```

Here methodObject is an object and display is a method which is associated with methodObject.

Question 26. What is JavaScript Self-Invoking anonymous function or Self-Executing anonymous function.

A `self-invoking` anonymous function also called `self-executing anonymous function` runs immediately or automatically when we define it and self-invoking anonymous function doesn't have any name at all. Self-Invoking anonymous function syntax:

```
(function() {
  console.log("Self-Invoking function code logic ");
})();
Output: Self-Invoking function code logic
```

We must have to know the fact that JavaScript functions run immediately when we put `()` after their names.

```
function display() {
  console.log("Display me");
}
display();  // This will run immediately

Output: "Display me"
/*
This will not run immediately as we haven't put () after function
name, function name is use full when we want to pass it as
callback to another function or method.
*/
display;
```

```
function testCallBack(callback) {
  callback ();  // This will be call display method immediately if callback
parameter is being set method display
}
testCallBack(display); // Here display function is being passed as callback
```

Question 27. Describe Singleton Pattern In JavaScript?

The singleton pattern is the most commonly used design pattern and one that you will probably is more than any others. It provides a great way to wrap the code into a logical unit that can be accessed through a single variable. The Singleton design pattern is used when only one instance of an object is needed throughout the lifetime of an application.

In JavaScript, there is a different way to achieve singleton object than any other object oriented supported language (Java, C++). In JavaScript Singleton pattern have many uses, they can be used for NameSpacing, which reduce the number of global variables in your page (prevent from polluting global space), organizing the code in a consistent manner, which increase the readability and maintainability of your pages. There are two important points in the traditional definition of Singleton pattern:

- There should be only one instance allowed for a class and
- We should allow global point of access to that single instance
- Let me define singleton pattern in JavaScript context:

> It is an object that is used to create namespace and group together a related set of methods and attributes (encapsulation) and if we allow to initiate then it can be initiated only once.

In JavaScript, we can create singleton though object literal. However, there is some another way but that I will cover in next post.

A singleton object consists of two parts: The object itself, containing the members (Both methods and attributes) within it, and global variable used to access it. The variable is global so that object can be accessed anywhere in the page, this is a key feature of the singleton pattern.

JavaScript: A Singleton as a Namespace

As I have already stated above that singleton can be used to declare Namespace in JavaScript. NameSpacing is a large part of responsible programming in JavaScript. Because everything can be overwritten, and it is very easy to wipe out variable by mistake or a function, or even a class without even knowing it. A common example which happens frequently when you are working with another team member parallel,

```
function findUserName(id) {

}

/* Later in the page another programmer
added code */
var findUserName = $('#user_list');
```

```
/* You are trying to call :( */
console.log(findUserName())
```

One of the best ways to prevent accidentally overwriting variable is to namespace your code within a singleton object.

```
/*  Using Namespace */

var MyNameSpace = {
  findUserName : function(id) {},
  // Other methods and attribute go here as well
}

/* Later in the page another programmer
added code */
var findUserName = $('#user_list');

/* You are trying to call and you make this time workable */
console.log(MyNameSpace.findUserName());
```

Singleton Design Pattern Implementation

```
/* Lazy Instantiation skeleton for a singleton pattern */

var MyNameSpace = {};
MyNameSpace.Singleton = (function() {

  // Private attribute that holds the single instance
  var singletonInstance;

  // All of the normal code goes here
  function constructor() {
    // Private members
    var privateVar1 = "Nishant";
    var privateVar2 = [1,2,3,4,5];

    function privateMethod1() {
      // code stuff
    }

    function privateMethod1() {
      // code stuff
    }

    return {
      attribute1 : "Nishant",
      publicMethod: function() {
        alert("Nishant");// some code logic
      }
```

```
    }
  }

  return {
    // public method (Global access point to Singleton object)
    getInstance: function() {
      //instance already exist then return
      if(!singletonInstance) {
        singletonInstance = constructor();
      }
      return singletonInstance;
    }
  }

})();

// getting access of publicMethod
console.log(MyNamespace.Singleton.getInstance().publicMethod());
```

The singleton implemented above is easy to understand. The singleton class maintains a static reference to the lone singleton instance and return that reference from the static getInstance() method.

Question 28. What are the way by which we can create object in JavaScript ?

Method 1: `Function Based`

If we want to create several similar `objects`. In below code sample, `Employee` which is called *constructor function*. This is similar to classes in object oriented languages.

```
  function Employee(fName, lName, age, salary){
    this.firstName = fName;
    this.lastName = lName;
    this.age = age;
    this.salary = salary;
  }

  // Creating multiple object which have similar property but diff value assigned
to object property.
  var employee1 = new Employee('John', 'Moto', 24, '5000$');
  var employee1 = new Employee('Ryan', 'Jor', 26, '3000$');
  var employee1 = new Employee('Andre', 'Salt', 26, '4000$');
```

Method 2: `Object Literal`

Object Literal is best way to create an object and this is used frequently. Below is code sample for create employee object which contains property as well as method.

```
var employee = {
    name : 'Nishant',
    salary : 245678,
    getName : function(){
        return this.name;
    }
}
```

Below code sample is Nested Object Literal, Here address is an object inside employee object.

```
var employee = {
    name : 'Nishant',
    salary : 245678,
    address : {
        addressLine1 : 'BITS Pilani',
        addressLine2 : 'Vidya Vihar'.
        phoneNumber: {
          workPhone: 7098889765,
          homePhone: 1234567898
        }
    }
}
```

Method 3: Using JavaScript new keyword

In below code sample object has been created using Object constructor function.

```
var employee = new Object(); // Created employee object using new keywords and
Object()
employee.name = 'Nishant';
employee.getName = function(){
    return this.name;
}
```

Note: As a best practices object literal way is used to create object over this method.

Question 29. Write a function called deepClone which takes an object and creates a object copy of it.

```
var newObject = deepClone(obj);
```

Solution:

```
function deepClone(object){
    var newObject = {};
    for(var key in object){
        if(typeof object[key] === 'object'){
         newObject[key] = deepClone(object[key]);
        }else{
         newObject[key] = object[key];
        }
    }
    return newObject;
}
```

Explanation: We have been asked to do deep copy of object so What's basically it's mean ??. Let's understand in this way you have been given an object `personalDetail` this object contains some property which again a type of object here as you can see `address` is an object and `phoneNumber` in side an `address` is also an object. In simple term `personalDetail` is nested object(object inside object). So Here deep copy means we have to copy all the property of `personalDetail` object including nested object.

```
var personalDetail = {
    name : 'Nishant',
    address : {
      location: 'xyz',
      zip : '123456',
      phoneNumber : {
        homePhone: 8797912345,
        workPhone : 1234509876
      }
    }
}
```

So when we do deep clone then we should copy every property (including the nested object).

Question 30. Best way to detect `undefined` object property in JavaScript.

> Suppose we have given an object `person`

```
var person = {
    name: 'Nishant',
    age : 24
}
```

Here the `person` object has a `name` and `age` property. Now we are trying to access the **salary** property which we haven't declared on the person object so while accessing it will return undefined. So how we will ensure whether property is undefined or not before performing some operation over it?

Explanation:

We can use `typeof` operator to check undefined

```
if(typeof someProperty === 'undefined'){
    console.log('something is undefined here');
}
```

Now we are trying to access salary property of person object.

```
if(typeof person.salary === 'undefined'){
    console.log("salary is undefined here because we haven't declared");
}
```

Question 31. Write a function called `Clone` which takes an object and creates a object copy of it but not copy deep property of object.

```
   var objectLit = {foo : 'Bar'};
    var cloneObj = Clone(obj); // Clone is the function which you have to write
    console.log(cloneObj === Clone(objectLit)); // this should return false
    console.log(cloneObj == Clone(objectLit)); // this should return true
```

solution:

```
function Clone(object){
  var newObject = {};
  for(var key in object){
    newObject[key] = object[key];
  }
  return newObject;
}
```

Question 32. Suggest your question!

Question 33. How to check whether a key exist in a JavaScript object or not.

> Let say we have `person` object with property **name** and **age**

```
var person = {
    name: 'Nishant',
    age: 24
}
```

Now we want to check whether `name` property exist in `person` object or not ?

In JavaScript object can have own property, in above example name and age is own property of person object. Object also have some of inherited property of base object like toString is inherited property of person object.

So how we will check whether property is own property or inherited property.

Method 1: We can use `in` operator on objet to check own property or inherited property.

```
console.log('name' in person); // checking own property print true
console.log('salary' in person); // checking undefined property print false
```

`in` operator also look into inherited property if it doesn't find property defined as own property. For instance If I check existence of toString property as we know that we haven't declared this property on person object so `in` operator look into there base property.

Here

```
console.log('toString' in person); // Will print true
```

If we want to test property of object instance not inherited properties then we will use `hasOwnProperty` method of object instance.

```
console.log(person.hasOwnProperty('toString')); // print false
console.log(person.hasOwnProperty('name')); // print true
console.log(person.hasOwnProperty('salary')); // print false
```

Question 34. Suggest your question!

Question 35. Suggest your question!

Question 36. What is best way to detect an arrays object on JavaScript ?

> We always encounter in such situation where we need to know whether value is type of array or not.

For Instance : Below code perform some operation based value type

```
function(value){
    if("value is an array"){
        // Then perform some operation
    }else{
        // otherwise
    }
}
```

Let's discuss some way to detect an array in JavaScript.

Method 1 :

Duck typing test for array type detection

```
// Duck typing arrays
function isArray(value){
   return typeof value.sort === 'function';
}
```

As we can see above isArray method will return true if value object have `sort` method of type `function`. Now assume you have created a object with sort method

```
var bar = {
    sort: function(){
        // Some code
    }
}
```

Now when you check `isArray(bar)` then it will return true because bar object has sort method, But the fact is bar is not an array.

So method 1 is not a best way to detect an array as you can see it's not handle the case when some object has sort method.

Method 2:

Juriy Zaytsev (Also known as kangax) proposed an elegant solution to this.

```
function isArray(value){
    return Object.prototype.toString.call(value) === '[object Array]';
}
```

This approach is most popular way to detecting a value of type array in JavaScript and recommended to use. This approach relies on the fact that, native toString() method on a given value produce a standard string in all browser.

> ECMAScript 5 has introduced **Array.isArray()** method to detect an array type value. The sole purpose of this method is accurately detecting whether a value is an array or not.

In many JavaScript libraries you may see below code for detecting an value of type array.

```
function(value){
   // ECMAScript 5 feature
    if(typeof Array.isArray === 'function'){
```

```
        return Array.isArray(value);
    }else{
       return Object.prototype.toString.call(value) === '[object Array]';
    }
}
```

Question 37. Best way to detect reference values of any type in JavaScript ?

> In Javascript Object are called as reference type, Any value other then primitive is definitely a reference type. There are several built-in reference type such as **Object**, **Array**, **Function**, **Date**, **null** and **Error**.

Detecting object using `typeof` operator

```
console.log(typeof {});           // object
console.log(typeof []);           // object
console.log(typeof new Array());  // object
console.log(typeof null);         // object
console.log(typeof new RegExp()); // object
console.log(typeof new Date());   // object
```

But the downside of using typeof operator to detect an object is that typeof returns `object` for `null` (However this is fact that null is an object in JavaScript).

The best way to detect an object of specific reference type using `instanceof` operator.

> Syntax : **value** instanceof **constructor**

```
//Detecting an array
if(value instanceof Array){
    console.log("value is type of array");
}
```

```
// Employee constructor function
function Employee(name){
    this.name = name; // Public property
}

var emp1 = new Employee('John');

console.log(emp1 instanceof Employee); // true
```

`instanceof` not only check the constructor which is used to create an object but also check it's prototype chain see below example.

```
console.log(emp1 instanceof Object); // true
```

Question 38. Describe Object-Based inheritance in JavaScript.

> Object-based inheritance also called prototypal inheritance in which we one object inherit from another object without invoking a constructor function.

The ECMAScript 5 **Object.create()** method is the easiest way for one object to inherit from another.

For Instance:

```
var employee = {
  name: 'Nishant',
  displayName: function () {
    console.log(this.name);
  }
};

var emp1 = Object.create(employee);
console.log(emp1.displayName());  // output "Nishant"
```

Above example create a new object `emp1` that inherits from `employee`. Here the inheritance occur as emp1's prototype is set to employee. After this emp1 is able to access the same properties and method on employee until new properties or method with the same name are defined.

For Instance: Defining displayName() method on emp1 automatically override the employee displayName.

```
emp1.displayName = function() {
    console.log('xyz-Anonymous');
};

employee.displayName(); //Nishant
emp1.displayName();//xyz-Anonymous
```

In addition to this **Object.create()** method also allows to specify a second argument which is an object containing additional properties and methods to add to the new object.

For Example

```
var emp1 = Object.create(employee, {
    name: {
        value: "John"
    }
});
```

```
emp1.displayName(); // "John"
employee.displayName(); // "Nishant"
```

In above example, emp1 is created with it's own value for name, so calling **displayName()** method display `"John"` instead of `"Nishant"`.

> Object created in this manner give you full control over newly created object. You are free to add, remove any properties and method you want.

Question 39. Describe Type-Based inheritance in JavaScript.

> Type-based inheritance works with constructor function instead of object, It means we need to call constructor function of the object from which you want to inherit.

Let say we have `Person` class which has name, age, salary properties and **incrementSalary()** method.

```
function Person(name, age, salary) {
  this.name = name;
  this.age = age;
  this.salary = salary;
  this.incrementSalary = function (byValue) {
    this.salary = this.salary + byValue;
  };
}
```

Now we wish to create Employee class which contains all the properties of Person class and wanted to add some additional properties into Employee class.

```
function Employee(company){
    this.company = company;
}

//Prototypal Inheritance
Employee.prototype = new Person("Nishant", 24,5000);
```

In above example, **Employee** type inherits from **Person**, which is called super types. It does so by assigning a new instance of Person to Employee prototype. After that, every instance of Employee inherits it's properties and methods from Person.

```
//Prototypal Inheritance
Employee.prototype = new Person("Nishant", 24,5000);

var emp1 = new Employee("Google");

console.log(emp1 instanceof Person); // true
console.log(emp1 instanceof Employee); // true
```

Let's understand Constructor inheritance

```
//Defined Person class
function Person(name){
    this.name = name || "Nishant";
}

var obj = {};

// obj inherit Person class properties and method
Person.call(obj); // constructor inheritance

console.log(obj); // Object {name: "Nishant"}
```

Here we saw calling **Person.call(obj)** define the name properties from `Person` to `obj`.

```
console.log(name in obj); // true
```

Type-based inheritance is best used with developer defined constructor function rather than natively in JavaScript. In addition to this also allows flexibility in how we create similar type of object.

Question 40. How we can prevent modification of object in JavaScript ?.

> ECMAScript 5 introduce several methods to prevent modification of object which lock down object to ensure that no one, accidentally or otherwise, change functionality of Object.

There are three levels of preventing modification:

1: Prevent extensions :

No new properties or methods can be added to the object, but one can change the existing properties and method.

For Example:

```
var employee = {
    name: "Nishant"
};

// lock the object
Object.preventExtensions(employee);

// Now try to change the employee object property name
employee.name = "John"; // work fine

//Now try to add some new property to the object
employee.age = 24; // fails silently unless it's inside the strict mode
```

2: Seal :

> It is same as prevent extension, in addition to this also prevent existing properties and methods from being deleted.

To seal an object, we use **Object.seal()** method. you can check whether an object is sealed or not using **Object.isSealed();**

```
var employee = {
    name: "Nishant"
};

// Seal the object
Object.seal(employee);

console.log(Object.isExtensible(employee)); // false
console.log(Object.isSealed(employee)); // true

delete employee.name // fails silently unless it's in strict mode

// Trying to add new property will give an error
employee.age = 30; // fails silently unless in strict mode
```

when an object is sealed, its existing properties and methods can't be removed. Sealed object are also non-extensible.

3: Freeze :

> Same as seal, In addition to this prevent existing properties methods from being modified (All properties and methods are read only).

To freeze an object, use Object.freeze() method. We can also determine whether an object is frozen using Object.isFrozen();

```
var employee = {
    name: "Nishant"
};

//Freeze the object
Object.freeze(employee);

// Seal the object
Object.seal(employee);

console.log(Object.isExtensible(employee)); // false
console.log(Object.isSealed(employee));     // true
console.log(Object.isFrozen(employee));     // true
```

```
employee.name = "xyz"; // fails silently unless in strict mode
employee.age = 30;     // fails silently unless in strict mode
delete employee.name   // fails silently unless it's in strict mode
```

Frozen objects are considered both non-extensible and sealed.

Recommended:

If you are decided to prevent modification, sealed, freeze the object then use in strict mode so that you can catch the error.

For Example:

```
"use strict";

var employee = {
    name: "Nishant"
};

//Freeze the object
Object.freeze(employee);

// Seal the object
Object.seal(employee);

console.log(Object.isExtensible(employee)); // false
console.log(Object.isSealed(employee));     // true
console.log(Object.isFrozen(employee));     // true


employee.name = "xyz"; // fails silently unless in strict mode
employee.age = 30;     // fails silently unless in strict mode
delete employee.name;  // fails silently unless it's in strict mode
```

Question 44. Write a log function which will add prefix `(your message)` to every message you log using console.log ?

> For example, If you log `console.log("Some message")` then output should be **(your message) Some message**

Logging error message or some informative message is always required when you dealing with client side JavaScript using console.log method. Some time you want to add some prefix to identify message generated log from your application hence you would like to prefix your app name in every console.log.

A general way to do this keep adding your app name in every console.log message like

```
console.log('your app name' + 'some error message');
```

But doing in this way you have to write your app name everytime when you log message using console.

There are some best way we can achieve this

```
function appLog() {
  var args = Array.prototype.slice.call(arguments);
  args.unshift('your app name');
  console.log.apply(console, args);
}

console.log(appLog("Some error message"));
//output of above console: 'your app name Some error message'
```

Question 45 . Write a function which will test string as a literal and as an object ?

For example: We can create string using string literal and using String constructor function.

```
// using string literal
var ltrlStr = "Hi I am string literal";
// using String constructor function
var objStr = new String("Hi I am string object");
```

> We can use typeof operator to test string literal and instanceof operator to test String object.

```
function isString(str) {
   return typeof(str) == 'string' || str instanceof String;
}

var ltrlStr = "Hi I am string literal";
var objStr = new String("Hi I am string object");
console.log(isString(ltrlStr)); // true
console.log(isString(objStr)); // true
```

Question 46 . What is typical use case for anonymous function in JavaScript ?

> Anonymous functions basically used in following scenario.

1. No name is needed if function is only used in one place, then there is no need to add a name to function.

   Let's take the example of setTimeout function

```
setTimeout(function(){
    alert("Hello");
},1000);
```

Here there is no need of using named function when we are sure that function which will alert `hello` would use only once in application.

2. Anonymous functions are declared inline and inline functions have advantages in the case that they can access variable in the parent scopes.

   Let's take a example of event handler. Notify event of particular type (such as click) for a given object.

   Let say we have HTML element (button) on which we want to add click event and when user do click on button we would like to execute some logic.

   ```
   <button id="myBtn"></button>
   ```

   Add Event Listener

   ```
   var btn = document.getElementById('myBtn');
   btn.addEventListener('click', function () {
     alert('button clicked');
   });
   ```

   Above example shows used of anonymous function as a callback function in event handler.

3. Passing anonymous function as a parameter to calling function.

   Example:

   ```
   // Function which will execute callback function
   function processCallback(callback){
       if(typeof callback === 'function'){
           callback();
       }
   }

   // Call function and pass anonymous function as callback
   processCallback(function(){
       alert("Hi I am anonymous callback function");
   });
   ```

The best way to take decision for using anonymous function is to asked.

> Will the function which I am going to define will use anywhere else.

If your answer is yes then go and create named function rather anonymous function.

Advantage of using anonymous function:

1. It can reduce a bit of code, particularly in recursive function and in callback function.
2. Avoid needless global namespace pollutions.

Question 47 . How to set a default parameter value ?

> If you are coming from python/c# you might be using default value for function parameter incase value(formal parameter) has not been passed. For Instance :

```
// Define sentEmail function
// configuration : Configuration object
// provider : Email Service provider, Default would be gmail
def sentEmail(configuration, provider = 'Gmail'):
    # Your code logic
```

In Pre ES6/ES2015

There are a lot of ways by which you can achieve this in pre ES2015.

Let's understand below code by which we achieved setting default parameter value.

Method 1: Setting default parameter value

```
function sentEmail(configuration, provider) {
  // Set default value if user has not passed value for provider
  provider = typeof provider !== 'undefined' ? provider : 'Gmail'
  // Your code logic
;
}
// In this call we are not passing provider parameter value
sentEmail({
  from: 'xyz@gmail.com',
  subject: 'Test Email'
});
// Here we are passing Yahoo Mail as a provider value
sentEmail({
  from: 'xyz@gmail.com',
  subject: 'Test Email'
}, 'Yahoo Mail');
```

Method 2: Setting default parameter value

```
function sentEmail(configuration, provider) {
  // Set default value if user has not passed value for provider
  provider = provider || 'Gmail'
```

```
  // Your code logic
;
}
// In this call we are not passing provider parameter value
sentEmail({
  from: 'xyz@gmail.com',
  subject: 'Test Email'
});
// Here we are passing Yahoo Mail as a provider value
sentEmail({
  from: 'xyz@gmail.com',
  subject: 'Test Email'
}, 'Yahoo Mail');
```

Question 48. Write code for merge two JavaScript Object dynamically.

> Let say you have two object

```
var person = {
    name : 'John',
    age  : 24
}

var location = {
    addressLine1 : 'Some Location x',
    addressLine2 : 'Some Location y',
    city : 'NewYork'
}
```

Write merge function which will take two object and add all the own property of second object into first object.

```
merge(person , location);

/* Now person should have 5 properties
name , age , addressLine1 , addressLine2 , city */
```

Method 1: Using ES6, Object assign method

```
function merge(toObj,fromObj){
    return Object.assign(person,location);
}
```

Method 2: Without using in-built function

```
function merge(toObj, fromObj) {
  // Make sure both of the parameter is an object
  if (typeof toObj === 'object' && typeof fromObj === 'object') {
    for (var pro in fromObj) {
      // Assign only own properties not inherited properties
      if (fromObj.hasOwnProperty(pro)) {
        // Assign property and value
        toObj[pro] = fromObj[pro];
      }
    }
  }else{
    throw "Merge function can apply only on object";
  }
}
```

Question 49. What is non-enumerable property in JavaScript and how can create ?

> Object can have properties that don't show up when you iterate through object using for...in loop or using Object.keys() to get an array of property names. This properties is know as non-enumerable properties.

Let say we have following object

```
var person = {
    name: 'John'
};
person.salary = '10000$';
person['country'] = 'USA';

console.log(Object.keys(person)); // ['name', 'salary', 'country']
```

As we know that person object properties `name`, `salary` ,`country` are enumerable hence it's shown up when we called Object.keys(person).

To create a non-enumerable property we have to use **Object.defineProperty()**. This is a special method for creating non-enumerable property in JavaScript.

```
var person = {
    name: 'John'
};
person.salary = '10000$';
person['country'] = 'USA';

// Create non-enumerable property
Object.defineProperty(person, 'phoneNo',{
    value : '8888888888',
    enumerable: false
```

```
})

Object.keys(person); // ['name', 'salary', 'country']
```

In above example `phoneNo` property didn't show up because we made it non-enumerable by setting **enumerable:false**

Now let's try to change value of `phoneNo`

```
person.phoneNo = '7777777777';
```

Changing non-enumerable property value will return error in `strict mode`. In non-strict mode it won't through any error but it won't change the value of phoneNo.

Bonus

Object.defineProperty() is also let you create read-only properties as we saw above, we are not able to modify phoneNo value of a person object.

Question 50. What is Function binding ?

> Function binding falls in advance JavaScript category and this is very popular technique to use in conjunction with event handler and callback function to preserve code execution context while passing function as a parameter.

Let's consider the following example:

```
var clickHandler = {
    message: 'click event handler',
    handleClick: function(event) {
        console.log(this.message);
    }
};

var btn = document.getElementById('myBtn');
// Add click event to btn
btn.addEventListener('click', clickHandler.handleClick);
```

Here in this example clickHandler object is created which contain message properties and handleClick method.

We have assigned handleClick method to a DOM button, which will be executed in response of click. When the button is clicked, then handleClick method is being called and console message. Here console.log should log the `click event handler` message but it actually log `undefined`.

The problem of displaying `undefined` is because of the execution context of clickHandler.handleClick method is not being saved hence `this` pointing to button `btn` object. We can fix this issue using bind method.

```
var clickHandler = {
    message: 'click event handler',
    handleClick: function(event) {
        console.log(this.message);
    }
};

var btn = document.getElementById('myBtn');
// Add click event to btn and bind the clickHandler object
btn.addEventListener('click', clickHandler.handleClick.bind(clickHandler));
```

`bind` method is available to all the function similar to call and apply method which take argument value of `this`.

Coding Questions

Hoisting

1. console.log(employeeId);

1. Some Value
2. Undefined
3. Type Error
4. ReferenceError: employeeId is not defined

Answer: 4) ReferenceError: employeeId is not defined

2. What would be the output of following code?

```
console.log(employeeId);
var employeeId = '19000';
```

1. Some Value
2. undefined
3. Type Error
4. ReferenceError: employeeId is not defined

Answer: 2) undefined

3. What would be the output of following code?

```
var employeeId = '1234abe';
(function(){
    console.log(employeeId);
```

```
    var employeeId = '122345';
})();
```

1. '122345'
2. undefined
3. Type Error
4. ReferenceError: employeeId is not defined

Answer: 2) undefined

4. What would be the output of following code?

```
var employeeId = '1234abe';
(function() {
    console.log(employeeId);
    var employeeId = '122345';
    (function() {
        var employeeId = 'abc1234';
    }());
}());
```

1. '122345'
2. undefined
3. '1234abe'
4. ReferenceError: employeeId is not defined

Answer: 2) undefined

5. What would be the output of following code?

```
(function() {
    console.log(typeof displayFunc);
    var displayFunc = function(){
        console.log("Hi I am inside displayFunc");
    }
}());
```

1. undefined
2. function
3. 'Hi I am inside displayFunc'
4. ReferenceError: displayFunc is not defined

Answer: 1) undefined

6. What would be the output of following code?

```
var employeeId = 'abc123';
function foo(){
    employeeId = '123bcd';
    return;
}
foo();
console.log(employeeId);
```

1. undefined
2. '123bcd'
3. 'abc123'
4. ReferenceError: employeeId is not defined

Answer: 2) '123bcd'

7. What would be the output of following code?

```
var employeeId = 'abc123';

function foo() {
    employeeId = '123bcd';
    return;

    function employeeId() {}
}
foo();
console.log(employeeId);
```

1. undefined
2. '123bcd'
3. 'abc123'
4. ReferenceError: employeeId is not defined

Answer: 3) 'abc123'

8. What would be the output of following code?

```
var employeeId = 'abc123';

function foo() {
    employeeId();
    return;

    function employeeId() {
        console.log(typeof employeeId);
    }
```

```
}
foo();
```

1. undefined
2. function
3. string
4. ReferenceError: employeeId is not defined

Answer: 2) 'function'

9. What would be the output of following code?

```
function foo() {
    employeeId();
    var product = 'Car';
    return;

    function employeeId() {
        console.log(product);
    }
}
foo();
```

1. undefined
2. Type Error
3. 'Car'
4. ReferenceError: product is not defined

Answer: 1) undefined

10. What would be the output of following code?

```
(function foo() {
    bar();

    function bar() {
        abc();
        console.log(typeof abc);
    }

    function abc() {
        console.log(typeof bar);
    }
}());
```

1. undefined undefined
2. Type Error

3. function function
4. ReferenceError: bar is not defined

Answer: 3) function function

Object

1. What would be the output of following code ?

```
(function() {
    'use strict';

    var person = {
        name: 'John'
    };
    person.salary = '10000$';
    person['country'] = 'USA';

    Object.defineProperty(person, 'phoneNo', {
        value: '8888888888',
        enumerable: true
    })

    console.log(Object.keys(person));
})();
```

1. Type Error
2. undefined
3. ["name", "salary", "country", "phoneNo"]
4. ["name", "salary", "country"]

Answer: 3) ["name", "salary", "country", "phoneNo"]

2. What would be the output of following code ?

```
(function() {
    'use strict';

    var person = {
        name: 'John'
    };
    person.salary = '10000$';
    person['country'] = 'USA';

    Object.defineProperty(person, 'phoneNo', {
        value: '8888888888',
        enumerable: false
    })
```

```
    console.log(Object.keys(person));
})();
```

1. Type Error
2. undefined
3. ["name", "salary", "country", "phoneNo"]
4. ["name", "salary", "country"]

Answer: 4) ["name", "salary", "country"]

3. What would be the output of following code ?

```
(function() {
    var objA = {
        foo: 'foo',
        bar: 'bar'
    };
    var objB = {
        foo: 'foo',
        bar: 'bar'
    };
    console.log(objA == objB);
    console.log(objA === objB);
}());
```

1. false true
2. false false
3. true false
4. true true

Answer: 2) false false

4. What would be the output of following code ?

```
(function() {
    var objA = new Object({foo: "foo"});
    var objB = new Object({foo: "foo"});
    console.log(objA == objB);
    console.log(objA === objB);
}());
```

1. false true
2. false false
3. true false
4. true true

Answer: 2) false false

5. What would be the output of following code ?

```
(function() {
    var objA = Object.create({
        foo: 'foo'
    });
    var objB = Object.create({
        foo: 'foo'
    });
    console.log(objA == objB);
    console.log(objA === objB);
}());
```

1. false true
2. false false
3. true false
4. true true

Answer: 2) false false

6. What would be the output of following code ?

```
(function() {
    var objA = Object.create({
        foo: 'foo'
    });
    var objB = Object.create(objA);
    console.log(objA == objB);
    console.log(objA === objB);
}());
```

1. false true
2. false false
3. true false
4. true true

Answer: 2) false false

7. What would be the output of following code ?

```
(function() {
    var objA = Object.create({
        foo: 'foo'
    });
    var objB = Object.create(objA);
    console.log(objA.toString() == objB.toString());
```

```
    console.log(objA.toString() === objB.toString());
}());
```

1. false true
2. false false
3. true false
4. true true

Answer: 4) true true

8. What would be the output of following code ?

```
(function() {
    var objA = Object.create({
        foo: 'foo'
    });
    var objB = objA;
    console.log(objA == objB);
    console.log(objA === objB);
    console.log(objA.toString() == objB.toString());
    console.log(objA.toString() === objB.toString());
}());
```

1. true true true false
2. true false true true
3. true true true true
4. true true false false

Answer: 3) true true true true

9. What would be the output of following code ?

```
(function() {
    var objA = Object.create({
        foo: 'foo'
    });
    var objB = objA;
    objB.foo = 'bar';
    console.log(objA.foo);
    console.log(objB.foo);
}());
```

1. foo bar
2. bar bar
3. foo foo
4. bar foo

Answer: 2) bar bar

10. What would be the output of following code ?

```
(function() {
    var objA = Object.create({
        foo: 'foo'
    });
    var objB = objA;
    objB.foo = 'bar';

    delete objA.foo;
    console.log(objA.foo);
    console.log(objB.foo);
}());
```

1. foo bar
2. bar bar
3. foo foo
4. bar foo

Answer: 3) foo foo

11. What would be the output of following code ?

```
(function() {
    var objA = {
        foo: 'foo'
    };
    var objB = objA;
    objB.foo = 'bar';

    delete objA.foo;
    console.log(objA.foo);
    console.log(objB.foo);
}());
```

1. foo bar
2. undefined undefined
3. foo foo
4. undefined bar

Answer: 2) undefined undefined

Array

1. What would be the output of following code?

```
(function() {
    var array = new Array('100');
    console.log(array);
    console.log(array.length);
}());
```

1. undefined undefined
2. [undefined × 100] 100
3. ["100"] 1
4. ReferenceError: array is not defined

Answer: 3) ["100"] 1

2. What would be the output of following code?

```
(function() {
    var array1 = [];
    var array2 = new Array(100);
    var array3 = new Array(['1',2,'3',4,5.6]);
    console.log(array1);
    console.log(array2);
    console.log(array3);
    console.log(array3.length);
}());
```

1. [] [] [Array[5]] 1
2. [] [undefined × 100] Array[5] 5
3. [] [] ['1',2,'3',4,5.6] 5
4. [] [] [Array[5]] 5

Answer: 1) [] [] [Array[5]] 1

3. What would be the output of following code?

```
(function () {
  var array = new Array('a', 'b', 'c', 'd', 'e');
  array[10] = 'f';
  delete array[10];
  console.log(array.length);
}());
```

1. 11
2. 5
3. 6
4. undefined

Answer: 1) 11

4. What would be the output of following code?

```
(function(){
    var animal = ['cow','horse'];
        animal.push('cat');
        animal.push('dog','rat','goat');
        console.log(animal.length);
})();
```

1. 4
2. 5
3. 6
4. undefined

Answer: 3) 6

5. What would be the output of following code?

```
(function(){
    var animal = ['cow','horse'];
        animal.push('cat');
        animal.unshift('dog','rat','goat');
        console.log(animal);
})();
```

1. ['dog', 'rat', 'goat', 'cow', 'horse', 'cat']
2. ['cow', 'horse', 'cat', 'dog', 'rat', 'goat']
3. Type Error
4. undefined

Answer: 1) ['dog', 'rat', 'goat', 'cow', 'horse', 'cat']

6. What would be the output of following code?

```
(function(){
    var array = [1,2,3,4,5];
    console.log(array.indexOf(2));
    console.log([{name: 'John'},{name : 'John'}].indexOf({name:'John'}));
    console.log([[1],[2],[3],[4]].indexOf([3]));
    console.log("abcdefgh".indexOf('e'));
})();
```

1. 1 -1 -1 4
2. 1 0 -1 4

3. 1 -1 -1 -1
4. 1 undefined -1 4

Answer: 1) 1 -1 -1 4

7. What would be the output of following code?

```
(function(){
    var array = [1,2,3,4,5,1,2,3,4,5,6];
    console.log(array.indexOf(2));
    console.log(array.indexOf(2,3));
    console.log(array.indexOf(2,10));
})();
```

1. 1 -1 -1
2. 1 6 -1
3. 1 1 -1
4. 1 undefined undefined

Answer: 2) 1 6 -1

8. What would be the output of following code?

```
(function(){
    var numbers = [2,3,4,8,9,11,13,12,16];
    var even = numbers.filter(function(element, index){
        return element % 2 === 0;
    });
    console.log(even);

    var containsDivisibleby3 = numbers.some(function(element, index){
        return element % 3 === 0;
    });

    console.log(containsDivisibleby3);
})();
```

1. [2, 4, 8, 12, 16] [0, 3, 0, 0, 9, 0, 12]
2. [2, 4, 8, 12, 16] [3, 9, 12]
3. [2, 4, 8, 12, 16] true
4. [2, 4, 8, 12, 16] false

Answer: 3) [2, 4, 8, 12, 16] true

9. What would be the output of following code?

```
(function(){
    var containers = [2,0,false,"", '12', true];
    var containers = containers.filter(Boolean);
    console.log(containers);
    var containers = containers.filter(Number);
    console.log(containers);
    var containers = containers.filter(String);
    console.log(containers);
    var containers = containers.filter(Object);
    console.log(containers);
})();
```

1. [2, '12', true] [2, '12', true] [2, '12', true] [2, '12', true]
2. [false, true] [2] ['12'] [ ]
3. [2,0,false,"", '12', true] [2,0,false,"", '12', true] [2,0,false,"", '12', true] [2,0,false,"", '12', true]
4. [2, '12', true] [2, '12', true, false] [2, '12', true,false] [2, '12', true,false]

Answer: 1) [2, '12', true] [2, '12', true] [2, '12', true] [2, '12', true]

10. What would be the output of following code?

```
(function(){
    var list = ['foo','bar','john','ritz'];
        console.log(list.slice(1));
        console.log(list.slice(1,3));
        console.log(list.slice());
        console.log(list.slice(2,2));
        console.log(list);
})();
```

1. ['bar', 'john', 'ritz'] ['bar', 'john'] ['foo', 'bar', 'john', 'ritz'] [] ['foo', 'bar', 'john', 'ritz']
2. ['bar', 'john', 'ritz'] ['bar', 'john','ritz] ['foo', 'bar', 'john', 'ritz'] [] ['foo', 'bar', 'john', 'ritz']
3. ['john', 'ritz'] ['bar', 'john'] ['foo', 'bar', 'john', 'ritz'] [] ['foo', 'bar', 'john', 'ritz']
4. ['foo'] ['bar', 'john'] ['foo', 'bar', 'john', 'ritz'] [] ['foo', 'bar', 'john', 'ritz']

Answer: 1) ['bar', 'john', 'ritz'] ['bar', 'john'] ['foo', 'bar', 'john', 'ritz'] [] ['foo', 'bar', 'john', 'ritz']

11. What would be the output of following code?

```
(function(){
    var list = ['foo','bar','john'];
        console.log(list.splice(1));
        console.log(list.splice(1,2));
        console.log(list);
})();
```

1. ['bar', 'john'] [] ['foo']
2. ['bar', 'john'] [] ['bar', 'john']
3. ['bar', 'john'] ['bar', 'john'] ['bar', 'john']
4. ['bar', 'john'] [] []

Answer: 1. ['bar', 'john'] [] ['foo']

12. What would be the output of following code?

```
(function(){
    var arrayNumb = [2, 8, 15, 16, 23, 42];
    arrayNumb.sort();
    console.log(arrayNumb);
})();
```

1. [2, 8, 15, 16, 23, 42]
2. [42, 23, 26, 15, 8, 2]
3. [15, 16, 2, 23, 42, 8]
4. [2, 8, 15, 16, 23, 42]

Answer: 3. [15, 16, 2, 23, 42, 8]

Function:

1. What would be the output of following code ?

```
function funcA(){
    console.log("funcA ", this);
    (function innerFuncA1(){
        console.log("innerFunc1", this);
        (function innerFunA11(){
            console.log("innerFunA11", this);
        })();
    })();
}

console.log(funcA());
```

1. funcA Window {...} innerFunc1 Window {...} innerFunA11 Window {...}
2. undefined
3. Type Error
4. ReferenceError: this is not defined

Answer: 1)

2. What would be the output of following code ?

```
var obj = {
    message: "Hello",
    innerMessage: !(function() {
        console.log(this.message);
    })()
};

console.log(obj.innerMessage);
```

1. ReferenceError: this.message is not defined
2. undefined
3. Type Error
4. undefined true

Answer: 4) undefined true

3. What would the output of following code ?

```
var obj = {
    message: "Hello",
    innerMessage: function() {
        console.log(this.message);
    }
};

console.log(obj.innerMessage());
```

1. Hello
2. undefined
3. Type Error
4. ReferenceError: this.message is not defined

Answer: 1) Hello

4. What would the output of following code ?

```
var obj = {
  message: 'Hello',
  innerMessage: function () {
    (function () {
      console.log(this.message);
    }());
  }
};
console.log(obj.innerMessage());
```

1. Type Error
2. Hello
3. undefined
4. ReferenceError: this.message is not defined

Answer: 3) undefined

5. What would the output of following code ?

```
var obj = {
  message: 'Hello',
  innerMessage: function () {
    var self = this;
    (function () {
      console.log(self.message);
    }());
  }
};
console.log(obj.innerMessage());
```

1. Type Error
2. 'Hello'
3. undefined
4. ReferenceError: self.message is not defined

Answer: 2) 'Hello'

6. What would the output of following code ?

```
function myFunc(){
    console.log(this.message);
}
myFunc.message = "Hi John";

console.log(myFunc());
```

1. Type Error
2. 'Hi John'
3. undefined
4. ReferenceError: this.message is not defined

Answer: 3) undefined

7. What would the output of following code ?

```
function myFunc(){
    console.log(myFunc.message);
```

```
}
myFunc.message = "Hi John";

console.log(myFunc());
```

1. Type Error
2. 'Hi John'
3. undefined
4. ReferenceError: this.message is not defined

Answer: 2) 'Hi John'

8. What would the output of following code ?

```
function myFunc() {
  myFunc.message = 'Hi John';
  console.log(myFunc.message);
}
console.log(myFunc());
```

1. Type Error
2. 'Hi John'
3. undefined
4. ReferenceError: this.message is not defined

Answer: 2) 'Hi John'

9. What would the output of following code ?

```
function myFunc(param1,param2) {
  console.log(myFunc.length);
}
console.log(myFunc());
console.log(myFunc("a","b"));
console.log(myFunc("a","b","c","d"));
```

1. 2 2 2
2. 0 2 4
3. undefined
4. ReferenceError

Answer: a) 2 2 2

10. What would the output of following code ?

```

```

```
function myFunc() {
  console.log(arguments.length);
}
console.log(myFunc());
console.log(myFunc("a","b"));
console.log(myFunc("a","b","c","d"));
```

1. 2 2 2
2. 0 2 4
3. undefined
4. ReferenceError

Answer: 2) 0 2 4

Object Oriented

1. What would the output of following code ?

```
function Person(name, age){
    this.name = name || "John";
    this.age = age || 24;
    this.displayName = function(){
        console.log(this.name);
    }
}

Person.name = "John";
Person.displayName = function(){
    console.log(this.name);
}

var person1 = new Person('John');
    person1.displayName();
    Person.displayName();
```

1. John Person
2. John John
3. John undefined
4. John John

Answer: 1) John Person

Scopes

1. What would the output of following code ?

```
function passWordMngr() {
    var password = '12345678';
    this.userName = 'John';
    return {
        pwd: password
    };
}
// Block End
var userInfo = passWordMngr();
console.log(userInfo.pwd);
console.log(userInfo.userName);
```

1. 12345678 Window
2. 12345678 John
3. 12345678 undefined
4. undefined undefined

Answer: 3) 12345678 undefined

2. What would the output of following code ?

```
var employeeId = 'aq123';
function Employee() {
  this.employeeId = 'bq1uy';
}
console.log(Employee.employeeId);
```

1. Reference Error
2. aq123
3. bq1uy
4. undefined

Answer: 4) undefined

3. What would the output of following code ?

```
var employeeId = 'aq123';

function Employee() {
    this.employeeId = 'bq1uy';
}
console.log(new Employee().employeeId);
Employee.prototype.employeeId = 'kj182';
Employee.prototype.JobId = '1BJKSJ';
console.log(new Employee().JobId);
console.log(new Employee().employeeId);
```

1. bq1uy 1BJKSJ bq1uy undefined
2. bq1uy 1BJKSJ bq1uy
3. bq1uy 1BJKSJ kj182
4. undefined 1BJKSJ kj182

Answer: 2) bq1uy 1BJKSJ bq1uy

4. What would the output of following code ?

```
var employeeId = 'aq123';
(function Employee() {
    try {
        throw 'foo123';
    } catch (employeeId) {
        console.log(employeeId);
    }
    console.log(employeeId);
}());
```

1. foo123 aq123
2. foo123 foo123
3. aq123 aq123
4. foo123 undefined

Answer: 1) foo123 aq123

Call, Apply, Bind

1. What would the output of following code ?

```
(function() {
    var greet = 'Hello World';
    var toGreet = [].filter.call(greet, function(element, index) {
        return index > 5;
    });
    console.log(toGreet);
}());
```

1. Hello World
2. undefined
3. World
4. ['W', 'o', 'r', 'l', 'd']

Answer: 4) ['W', 'o', 'r', 'l', 'd']

2. What would the output of following code ?

```
(function() {
    var fooAccount = {
        name: 'John',
        amount: 4000,
        deductAmount: function(amount) {
            this.amount -= amount;
            return 'Total amount left in account: ' + this.amount;
        }
    };
    var barAccount = {
        name: 'John',
        amount: 6000
    };
    var withdrawAmountBy = function(totalAmount) {
        return fooAccount.deductAmount.bind(barAccount, totalAmount);
    };
    console.log(withdrawAmountBy(400)());
    console.log(withdrawAmountBy(300)());
}());
```

1. Total amount left in account: 5600 Total amount left in account: 5300
2. undefined undefined
3. Total amount left in account: 3600 Total amount left in account: 3300
4. Total amount left in account: 5600 Total amount left in account: 5600

Answer: 1) Total amount left in account: 5600 Total amount left in account: 5300

3. What would the output of following code ?

```
(function() {
    var fooAccount = {
        name: 'John',
        amount: 4000,
        deductAmount: function(amount) {
            this.amount -= amount;
            return this.amount;
        }
    };
    var barAccount = {
        name: 'John',
        amount: 6000
    };
    var withdrawAmountBy = function(totalAmount) {
        return fooAccount.deductAmount.apply(barAccount, [totalAmount]);
    };
    console.log(withdrawAmountBy(400));
    console.log(withdrawAmountBy(300));
    console.log(withdrawAmountBy(200));
}());
```

1. 5600 5300 5100
2. 3600 3300 3100
3. 5600 3300 5100
4. undefined undefined undefined

Answer: 1) 5600 5300 5100

4. What would the output of following code ?

```
(function() {
    var fooAccount = {
        name: 'John',
        amount: 6000,
        deductAmount: function(amount) {
            this.amount -= amount;
            return this.amount;
        }
    };
    var barAccount = {
        name: 'John',
        amount: 4000
    };
    var withdrawAmountBy = function(totalAmount) {
        return fooAccount.deductAmount.call(barAccount, totalAmount);
    };
    console.log(withdrawAmountBy(400));
    console.log(withdrawAmountBy(300));
    console.log(withdrawAmountBy(200));
}());
```

1. 5600 5300 5100
2. 3600 3300 3100
3. 5600 3300 5100
4. undefined undefined undefined

Answer: 2) 3600 3300 3100

5. What would the output of following code ?

```
(function greetNewCustomer() {
    console.log('Hello ' + this.name);
}.bind({
    name: 'John'
})());
```

1. Hello John
2. Reference Error
3. Window

4. undefined

Answer: 1) Hello John

6. What would the output of following code ?

```
(function greetNewCustomer() {
    console.log('Hello ' + this.name);
}.bind({
    name: 'John'
})());
```

1. Hello John
2. Reference Error
3. Window
4. undefined

Answer: 1) Hello John

Callback Function

1. What would the output of following code ?

```
function getDataFromServer(apiUrl){
    var name = "John";
    return {
        then : function(fn){
            fn(name);
        }
    }
}

getDataFromServer('www.google.com').then(function(name){
    console.log(name);
});
```

1. John
2. undefined
3. Reference Error
4. fn is not defined

Answer: 1) John

2. What would the output of following code ?

```
(function(){
    var arrayNumb = [2, 8, 15, 16, 23, 42];
    Array.prototype.sort = function(a,b){
        return a - b;
    };
    arrayNumb.sort();
    console.log(arrayNumb);
})();

(function(){
    var numberArray = [2, 8, 15, 16, 23, 42];
    numberArray.sort(function(a,b){
        if(a == b){
            return 0;
        }else{
            return a < b ? -1 : 1;
        }
    });
    console.log(numberArray);
})();

(function(){
    var numberArray = [2, 8, 15, 16, 23, 42];
    numberArray.sort(function(a,b){
        return a-b;
    });
    console.log(numberArray);
})();
```

1. [2, 8, 15, 16, 23, 42] [2, 8, 15, 16, 23, 42] [2, 8, 15, 16, 23, 42]
2. undefined undefined undefined
3. [42, 23, 16, 15, 8, 2] [42, 23, 16, 15, 8, 2] [42, 23, 16, 15, 8, 2]
4. Reference Error

Answer: 1) [2, 8, 15, 16, 23, 42] [2, 8, 15, 16, 23, 42] [2, 8, 15, 16, 23, 42]

Return Statement

1. What would the output of following code ?

```
(function(){
    function sayHello(){
        var name = "Hi John";
        return
        {
            fullName: name
        }
    }
    console.log(sayHello().fullName);
})();
```

1. Hi John
2. undefined
3. Reference Error
4. Uncaught TypeError: Cannot read property 'fullName' of undefined

Answer: 4) Uncaught TypeError: Cannot read property 'fullName' of undefined

2. What would the output of following code ?

```
function getNumber(){
    return (2,4,5);
}

var numb = getNumber();
console.log(numb);
```

1. 5
2. undefined
3. 2
4. (2,4,5)

Answer: 1) 5

3. What would the output of following code ?

```
function getNumber(){
    return;
}

var numb = getNumber();
console.log(numb);
```

1. null
2. undefined
3. ""
4. 0

Answer: 2) undefined

4**. What would the output of following code ?

```
function mul(x){
    return function(y){
        return [x*y, function(z){
            return x*y + z;
```

```
        }];
    }
}

console.log(mul(2)(3)[0]);
console.log(mul(2)(3)[1](4));
```

1. 6, 10
2. undefined undefined
3. Reference Error
4. 10, 6

Answer: 1) 6, 10

5**. What would the output of following code ?

```
function mul(x) {
    return function(y) {
        return {
            result: x * y,
            sum: function(z) {
                return x * y + z;
            }
        };
    };
}
console.log(mul(2)(3).result);
console.log(mul(2)(3).sum(4));
```

1. 6, 10
2. undefined undefined
3. Reference Error
4. 10, 6

Answer: 1) 6, 10

6. What would the output of following code ?

```
function mul(x) {
    return function(y) {
        return function(z) {
            return function(w) {
                return function(p) {
                    return x * y * z * w * p;
                };
            };
        };
    };
```

```
}
console.log(mul(2)(3)(4)(5)(6));
```

1. 720
2. undefined
3. Reference Error
4. Type Error

Answer: 1) 720