**Q: What is the difference between object.create and new keyword, when we use for creating an object?**

**Ans:** The difference is when we use Object.create method for creating an object; we cannot access those properties that defined inside prototype constructor. It is because prototype’s constructor function does not get call at all.  
  
See below example, buddy object cannot access pupper property that is defined into Dog Prototype. But at the same time, maddie object can access that property.

**Ex 1)**function Dog(){

this.pupper = 'Pupper';

};

Dog.prototype.pupperino = 'Pups.';

var maddie = new Dog();

var buddy = Object.create(Dog.prototype);

//Using Object.create()

console.log(buddy.pupper); // \*\*Output is undefined

console.log(buddy.pupperino); // Output is Pups

//Using New Keyword

console.log(maddie.pupper); // Output is Pupper

console.log(maddie.pupperino); // Output is Pups

Notice the output of buddy.pupper is undefined. Even though Object.create() sets its prototype to Dog, buddy does not have access to this.pupper in the constructor. This is due to the important difference that new Dog actually runs constructor code, whereas Object.create will not execute the constructor code.  
  
  
  
**Ex 2)**  
const Foo = function() {

this.greet = 'Hello world!';

}

Foo.prototype.greeting = function() {

console.log('I want to say: ' + this.greet);

};

const bar = new Foo();

const baz = Object.create(Foo.prototype);

bar.greeting(); // prints 'I want to say: Hello world!'

baz.greeting(); // prints `I want to say: undefined`

**output:**

**I want to say: Hello world!**

**I want to say: undefined**

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

**Q: What is difference between Single Page Apps vs multiple/ conventional Web Applications ?  
Ans:  
Single-Page Application:**

A single-page application is an app that works inside a browser and does not require page reloading during use. You are using this type of applications every day. These are, for instance: Gmail, Google Maps, Facebook or GitHub.  
SPA has two primary features; no page reloads & no extra wait time. It is just one web page that you visit which then loads all other content using JavaScript — which they heavily depend on.

**\*\*Cons of the Single-Page Application:**

It is very tricky and not an easy task to make SEO optimization of a Single-Page Application. Its content is loaded by AJAX (Asynchronous JavaScript and XML) — a method of exchanging data and updating in the application without refreshing the page.

It requires JavaScript to be present and enabled. If any user disables JavaScript in his or her browser, it won’t be possible to present application and its actions in a correct way.

Multi-Page Application

Multiple-page applications work in a “traditional” way. Every change e.g. display the data or submit data back to server requests rendering a new page from the server in the browser. These applications are large, bigger than SPAs because they need to be. Due to the amount of content, these applications have many levels of UI. Luckily, it’s not a problem anymore. Thanks to AJAX, we don’t have to worry that big and complex applications have to transfer a lot of data between server and browser. That solution improves and it allows to refresh only particular parts of the application. On the other hand, it adds more complexity and it is more difficult to develop than a single-page application.

Pros of the Multiple-Page Application:

Very good and easy for proper SEO management. It gives better chances to rank for different keywords since an application can be optimized for one keyword per page.

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

**Q: What is Prototypal inheritance?**

In JavaScript, every object have a special hidden property, that is either null or references to another object. That property is called “a prototype”:

**When we want to read a property from object, and it’s missing, JavaScript automatically takes it from the prototype. In programming, such taking from prototype property, is called as “prototypal inheritance”.**

The property [[Prototype]] is internal and hidden, but there are many ways to set it.

One of them is to use \_\_proto\_\_, like this:

let animal = {

eats: true

};

let rabbit = {

jumps: true

};

rabbit.\_\_proto\_\_ = animal; /\* since animal is just an object literal, do not use new keyword, if it is constructor, then we can use new \*/  
alert(rabbit.eats) // true

Please note that \_\_proto\_\_ is not the same as [[Prototype]]. That’s a getter/setter for it.

If we look for a property in rabbit, and it’s missing, JavaScript automatically takes it from animal.

In above example, when alert tries to read property rabbit.eats (\*\*), it’s not in rabbit, so JavaScript follows the [[Prototype]]reference and finds it in animal (look from the bottom up):

Here we can say that "animal is the prototype of rabbit" or "rabbit prototypally inherits from animal".

So if animal has a lot of useful properties and methods, then they become automatically available in rabbit. Such properties are called “inherited”.

* **Imp Note:**  
  If we call obj.method(), and the method is taken from the prototype, ‘**this**' still references obj. So methods always work with the **current object** even if they are inherited.

# --------------------------------------------------------------------------------------

**Q: What is Class inheritance?**Ans: One class can inherit another class. To inherit from another class, we use "extends" keyword.

Here Rabbit inherits from Animal:

class Animal {

constructor(name) {

this.speed = 0;

this.name = name;

}

run(speed) {

this.speed += speed;

alert(`${this.name} runs with speed ${this.speed}.`);

}

stop() {

this.speed = 0;

alert(`${this.name} stopped.`);

}

}

// Inherit from Animal

class Rabbit extends Animal {

hide() {

alert(`${this.name} hides!`);

}

}

let rabbit = new Rabbit("White Rabbit");

rabbit.run(5); // White Rabbit runs with speed 5.

rabbit.hide(); // White Rabbit hides!

## The extends keyword actually adds a [[Prototype]] reference from  Rabbit.prototype  to  Animal.prototype .

## ------------------------------------------------------------------------------------------------------------------------------------------------

## Q: What is method overriding?

If a class inherits a method from its superclass, then there is a chance to override the method declared in super class by declaring the same method in subclass.

In object-oriented terms, overriding means to override the functionality of an existing method.  
  
Suppose we have super class animal and subclass Rabbit, then Rabbit will inherit all the methods declared in animal. Let’s say we have stop method in our super class. That method will be inherited by Rabbit.  
  
But if we declare stop method in Rabbit too, then it will be used instead. Means stop method in Rabbit will override the stop method available in animal class. This is called as method overriding.  
  
But usually we don’t want to totally replace a parent method, but rather to build on top of it, or extend its functionality. We do something in our method, but call the parent method before/after it or in the process. Classes provide "super"  keyword for that. In subclass method, we need to call super class method as follows:  
  
A) calling super class method in subclass method:

# Syntax: super.*parentMethod*(…);

# Note: this statement needs to be written in subclass method

# B) calling super class constructor in subclass constructor:

# Syntax: super(…);

# Note: this statement needs to be written in subclass constructor

# ------------------------------------------------------------------------------------------------------------------------

**Q: Explain super keyword?**The **super** keyword is used to access and call Super class methods in subclass method. Often we need to extend functionality of superclass method or reuse super class method in subclass, in such cases we can use **super** keyword.

In below example, you will find in subclass annualSalary method, we are calling super class annualSalary method.

class emp {

constructor(name, monthlySal) {

this.name = name;

this.annualSal = monthlySal \* 12;

}

annualSalary() {

alert(`you are in superclass`);  
 alert(`Employee name is ${this.name}.`);

alert(`Annual salary is ${this.annualSal}.`);

}

}

class amol extends emp {

annualSalary() {

super.annualSalary(); // call parent annualSalary

alert("now you are in subclass");

}

}

let amolObj = new amol("Amol Mali", 50000);

amolObj.annualSalary();

# Output: you are in superclass Employee name is Amol Mali. Annual salary is 600000 now you are in subclass Note: we can call parent constructor too, in subclass using super keyword.

# --------------------------------------------------------------------------------------

# JavaScript Hoisting: Hoisting is a JavaScript mechanism where variable declarations(variables declared with var keyword) and function declarations are moved to the top of their scope before code execution. \*\**Note: Variables declared with ‘var’ keyword and functions declared with ‘function declaration’ syntax can be hoisted, Variables declared with let and const keyword and function expressions cannot be hoisted.*

# In many programming languages, we can use any variable after its declaration process completed. If we try to use any variable before its declaration, we get an error. For Ex: in C programming, suppose we need to use variable x in our program, in that case, 1) we have to declare a variable with its data-type like int x, int stands for integer type 2) Then we need to assign value to x, let’s say, x = 5 3) Now we can use value of variable x in our code. *Not acceptable: x = 5 // data type should be declared printf(x)*

# *acceptable: int x = 5; printf(x) But In JavaScript, a variable can be used before its declaration.* This process is called as JavaScript Hoisting. Hoisting is a JavaScript mechanism where variable declarations and function declarations are moved to the top of their scope before code execution. Variable Declarations are hoisted: Note: variable declaration hoised only if we are declaring variable with ‘var’ keyword. With let and const, it will throw an error,

# Example 1

# x = 5;  // Assign 5 to x; we are not declaring x, immediately setting the value console.log(x) // 5 var x; // Declare x, without declaration will also output the same Output: 5

### Example 2: What would be the output?

var x = 5; // Assign 5 to x

function myFunction(){

console.log(x);

var x = 15;

}  
myFunction();

Output: undefined

## We have assigned value 5 in global scope and 15 in function scope still Output is undefined. It is neither 5 nor 15, its because in function scope, we have assigned value to X after we have used it. So at top of the function it will be x = undefined. Hence we are getting value of x is undefined. And remember, if we remove var x= 15 from function or if we remove var keyword from function block , we will get value 5, declared in global scope.

## Declare Your Variables at the top!

JavaScript in ‘strict mode’ does not allow variables to be used if they are not declared.

[Hoisting functions](https://scotch.io/tutorials/understanding-hoisting-in-javascript#toc-hoisting-functions):

Function declarations can be hoisted completely to the top.   
  
Ex:

hoisted();

function hoisted(){

console.log('we are calling this function before its declaration')

}

Output: we are calling this function before its declaration

\*\*\*However function expressions cannot be hoisted,

hoisted();

var hoisted = function(){

console.log('we are calling this function before its declaration')

}  
  
output:

Error: hoisted is not a function

**Note:** Inner functions are hoisted at the top of **its outer function**; they will not hoisted at the top of the file. This happen because function scope feature available in JavaScript.

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

Q: Event Propagation means?

Event Propagation means calling all the listeners for the given event type, all the way up or down. Either it starts with innermost element and then propagated to outer elements [i.e. event bubbling] or it starts with outermost element and then propagated to inner elements [i.e. event capturing]

Event bubbling and capturing are two ways of event propagation in the HTML DOM.

Q: What is Event Bubbling?

With Event bubbling, the event is first captured and handled by the innermost element and then same event propagated to outer elements.

For Ex:  
<!DOCTYPE html>

**<html lang="en" onclick="alert('html')">**

<head>

<meta charset="UTF-8">

<title>Event Bubbling</title>

<style>

body \* {

margin: 10px;

border: 1px solid blue;

}

</style>

</head>

<body onclick="alert('body')">

<form onclick="alert('form')">FORM

<div onclick="alert('div')">DIV

<p onclick="alert('p')">P</p>

</div>

</form>

</body>

</html>

**Output:**   
A click on the inner <p> first runs onclick:

1. On that <p>.
2. Then on the outer <div>.
3. Then on the outer <form>.
4. Then on the body tag.

And finally on html tag

--------------------------------------------------------------------------------------------------------------------------------------------------  
  
Q: How can we stop event bubbling?  
A bubbling event starts from the target element and it goes upwards until it reach to <html> element.

But we can stop this bubbling by using **event.stopPropagation()** method, event.stopPropagation() is simply mean don’t go upward.

For instance, here body.onclick doesn’t work if you click on <button>:

<body onclick="alert(`body clicked`)">

<button onclick="event.stopPropagation()">Click me</button>

</body>

Ex 2) Persistent interview que:  
<body>

<style>

body \* {

margin: 10px;

border: 1px solid blue;

}

</style>

<div onclick="this.style.backgroundColor='red'; event.stopPropagation()">Div 1

<div onclick="this.style.backgroundColor='green'; event.stopPropagation()">Div 2

<div onclick="this.style.backgroundColor='blue'; event.stopPropagation()">Div 3

</div>

</div>

</div>

</body>

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

Q: What is Event Capturing?

With bubbling, the event is first captured and handled by the innermost element and then propagated to outer elements.

With capturing, the event is first captured by the outermost element and propagated to the inner elements.

Capturing is also called "trickling", which helps remember the propagation order:

<style>

body \* {

margin: 10px;

border: 1px solid blue;

}

</style>

<form>FORM

<div>DIV

<p>P</p>

</div>

</form>

<script>

for(let elem of document.querySelectorAll('\*')) {

elem.addEventListener("click", e => alert(`Capturing: ${elem.tagName}`), true);

}

</script>

Note: if we remove true that is third argument of addEventListener, event bubbling will get occur. It simply means there is only one difference calling event bubbling and capturing, that is third argument. If we pass third argument as true, event capture will run otherwise event bubbling will get executed by JavaScript. If we are not passing third argument, event bubbling will get executed; because by default third argument is false.

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

Q: How can we re-use variables defined in one function into another function?

Ans: We have to inherit one constructor function whose variables are inherited by another constructor as follows:

const A = function(x,y){

A.prototype.x = x;

A.prototype.y = y;

};

const B = function(){};

B.prototype = Object.create(A.prototype); // “B.prototype = new A”; is also acceptable

B.prototype.print = function(){

console.log(this.x); /\* since print is method of B, ‘this’ points to B, \*/

console.log(this.y);

};

const Aobj = new A(10,20);

const Bobj = new B();

Bobj.print()

Output:

10

20

Note: this.x and this.y in print method of constructor B, points to B. Since B does not have these variables, and B inheriting A constructor, B is taking those values from A. But if B contains those variables inside itself, in that case, B will not look for those variables inside A.  
  
Check below example wherein you will find local variable x and y is already available in B, so B is not going to take those values from A.   
--------------------------------------------------------------------------------------  
Q: What would be the output?

const A = function(x,y){

A.prototype.x = x;

A.prototype.y = y;

/\* remember, we are reusing x and y inside function b so in A’s constructor block we need to define these vars, outside the constructor block it won’t work \*/

};

const B = function(x,y){

this.x = x;

this.y=y;

};

B.prototype = Object.create(A.prototype); // “B.prototype = new A”; is also acceptable

B.prototype.print = function(){

console.log(this.x);

console.log(this.y);

};

const Aobj = new A(10,20);

const Bobj = new B(100,200);

Bobj.print()

Output:  
100

200

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

**Q: How can we debug any JS code?  
Ans:** Debugging is the process of testing, finding, and reducing bugs (errors) in computer programs.

**A) Use console.log or alert window to display the variable’s value at certain point**

**B) Setting Breakpoints:**

**1) In chrome, right click on page, select inspect element or press f12, then choose sources tab and open your file, where your code is written**

**2) Then set up a breakpoint by clicking on line numbers**

**3) Then on right you will find watch option, click on it, you will find plus icon, click on it then there you can enter variable name for which you want to check value at certain breakpoint.**

**4) If there are multiple variables you want to check at different breakpoints, at watch you need to add them**

**5) If multiple breakpoints are there then click on ‘resume script execution/f8' or step over next function call/f10**

**C) use debugger keyword:  
If you want to see value of any variable at certain point, write debugger keyword at that point in your code and use watch to see the value of variable to that point.**

**D) Try all the sizes using toggle device mode:   
Many a times we need to check our code at different resolutions. We cannot have every single mobile device on our desk then how can we overcome this issue? Chrome helps you with this. Open your inspector and click the ‘toggle device mode’ button. Watch your media queries come to life! There you can see your JS working or not like through JS if you are converting your navigation bar with hamburger icon and toggling the menus inside it.  
----------------------------------------------------------**

## ****Q: Best practices of writing good JS Code? OR Q: How to improve performance of your JS code** Ans:** 1) Avoid Global Variables

This includes all data types, objects, and functions.

Global variables and functions can be overwritten by other scripts.

Use local variables instead, and use closures as much as possible.

## 2) Always Declare Local Variables

All variables used in a function should be declared as **local** variables.

Local variables **must** be declared with the **var**keyword, otherwise they will become global variables. Remember variable declared within function without var keyword become global.

Strict mode does not allow undeclared variables.

## 3) Declarations on Top

It is a good coding practice to put all declarations at the top of each script or function.

This will:

* Give cleaner code
* Provide a single place to look for local variables
* Make it easier to avoid unwanted (implied) global variables
* Reduce the possibility of unwanted re-declarations

For Ex:   
/ Declare variables at the beginning  
var firstName, lastName, price, discount, fullPrice;  
  
// Use later  
firstName = "John";  
lastName = "Doe";

## 4) Initialize Variables

It is a good coding practice to initialize variables when you declare them.

This will:

* Give cleaner code
* Provide a single place to initialize variables
* Avoid undefined values

## 5) Don't Use new Object():

## It’s because if we assign value using new keyword, its type will be ‘object’ in all the cases:

Always treat numbers, strings, or booleans as primitive values. Not as objects.

### Example

var x = "John";              // typeof x return string  
var y = new String("John"); // typeof y return object\*\*  
(x === y) // is false because x is a string and y is an object.  
Below is also worse case, it’s because x and y both are treated as objects and we may expect them to be treated as string/primitive values.  
var x = new String("John");               
var y = new String("John");  
(x == y) /\* is false because you compare two objects & they are at different memory locations. \*/

* Use "" instead of new String()
* Use 0 instead of new Number()
* Use false instead of new Boolean()
* Use [] instead of new Array()
* Use /()/ instead of new RegExp()
* Use function (){} instead of new Function()
* Use {} Instead of New Object()

## 6) Beware of Automatic Type Conversions/ type juggling

Beware that numbers can accidentally be converted to strings or NaN (Not a Number).

JavaScript is loosely typed/ dynamically typed language. A variable can contain different data types, and a variable can change its data type:

### Example

var x = "Hello";     // typeof x is a string  
x = 5;               // changes typeof x to a number

x = x + ‘’ // again x is string

## 7) Use === Comparison

The == comparison operator always converts (to matching types) before comparison.

The === operator forces comparison of values and type.

## 8) Use Parameter Defaults

If a function is called with a missing argument, the value of the missing argument is set to **undefined**.

Undefined values can break your code. It is a good habit to assign default values to arguments.  
  
  
For Ex:  
function myFunction(x) {  
    if (x === undefined ) {  
       x = 10;

return x;  
  
    }  
}  
myFunction()  
Output: 10

## 9. Use curly braces to define block of code and always use semi-colons:

For Ex:  
  
var a = 1, x = 2;  
function callanother() {

console.log("you are in callanother")

}

if(a) /\* no curly braces here \*/

x = false

callanother()

Output:  
you are in callanother  
  
**\*\*\*One might think that the above code has interpreted correctly by the browser but browser interpretation will be as follows:**  
  
if(a) {

   x = false;

} //\*\*\* if block end here, this is the thing we must care about

callanother();

**\*\***We clearly see callanother function has called outside of **if block**. And we were assuming we are calling callanother from if block. Such situations can be painful in long term project and also they are hard to find quickly.

**10) If one time initialization is working fine then initialize variable or variables outside of the loop**  
if there are several iterations in the loop, it is always considered to be good practice that initializes your variables outside of the loop.

Bad practice of initializing the variable inside the for loop:  
<body>

<div class="container" id="container"></div>

<script>

var someArray = [1,2,3,4,5,6,7,8,9,10]; /\* array with 10 items \*/

for(var i = 0; i < someArray.length; i++) {

var container = document.getElementById('container'); /\* for loop will initialize this variable at 10 times \*/

container.innerHTML += "<br/>my number: " + i;

}

</script>

</body>  
  
**output:**Below texts will be appended to the element which has container id:  
my number: 0  
my number: 1  
my number: 2  
my number: 3  
my number: 4  
my number: 5  
my number: 6  
my number: 7  
my number: 8  
my number: 9  
  
Above for loop will initialize container variable 10 times which reduce our code performance. We can write in better way as follows:  
  
var container = document.getElementById('container'); // initialized only once

for(var i = 0, len = someArray.length; i < len;  i++) { code here }

## 11. Comment Your Code

It might seem unnecessary at first, but you must comment your code as best as possible. What happens when you return to the project months later, only to find that you can't easily remember what your line of thinking was? Or, what if one of your colleagues needs to revise your code? Always, always comment important sections of your code.

## 12. Long List of Variables? Omit the "Var" Keyword for each variable and Use Commas Instead

Bad:  
var someItem = 'some string';

var anotherItem = 'another string';

var oneMoreItem = 'one more string';

Better  
var someItem = 'some string',

anotherItem = 'another string',

oneMoreItem = 'one more string';

## 13) Raw JavaScript/Vanilla JavaScript Can Always Be Quicker than Using a Library For Ex: jQuery's "each" method is great for looping, but using a native "for" statement will always be an ounce quicker.

**14) Indent your code so it’s readable**

## 15) Remove "Language" attribute

Years ago, it wasn't uncommon to find the "language" attribute within script tags.

<script type="text/javascript" language="javascript"></script>  
However, this attribute has long since been deprecated; so leave it out.

16) Choose short but readable variable names:  
  
17) Use one global object to encapsulate any global variables you really need

18) Beware of ASI – automatic semi-colon insertions

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

Q: Explain any 3 CHALLENGES IN WEB APPLICATION DEVELOPMENT we often face?

***1) Performance perspective:*** The speed at which your website loads decides if it will be able to keep the visitors with it, or they will run away faster than your website’s loading speed: there are several ways through which we can improve loading speed of website like minimize http requests, compress your files using GZip, **Minify and combine css & JS files,** put css at top and JS at bottom, reduce your image sizes, enable cache, use CSS sprites, **specify image dimensions in html, use external stylesheets instead of @import etc.  
  
*2) Cross Browser support****:* For ex. Chrome supports input type number while firefox has not yet implemented it. Many newer browser version supports css pointer-events property but IE does not support.

**3) Client Window/viewport sizes**: your UI should be intact for all Screen Resolutions.

1. **IE, the big hurdle in UI development:** Making sure things look the same across all versions of Internet Explorer as they look in other browsers.

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

**What is RESTful API?**Ans: A RESTful API ([**Representational State Transfer)**](https://en.m.wikipedia.org/wiki/Representational_state_transfer)  is an application program interface ([API](http://searchexchange.techtarget.com/definition/application-program-interface)) that uses [HTTP](http://searchwindevelopment.techtarget.com/definition/HTTP) requests to GET, PUT, POST and DELETE data.

REST technology is an API that allows two software programs to communicate with each another.

## **--------------------------------------------------- filter() method in javascript (test-based):**

The filter() method creates an new array, filled with those array elements, that has passed in a certain test/condition. We need to specify our test in our callback function.

**Note:** filter() does not execute the function for array elements without values.

**Note:** filter() does not change the original array.  
  
Syntax: newArr = oldArr.filter(callback);

ex1: Return numbers from array that are greater than 17  
  
var ages = [32, 13, 18, 16, 40, 8, 88];

ages.filter(function(x) {

return x > 17;

});

Or define a separate callback function as follows: it’s similar as above code;

**var ages = [32, 13, 18, 16, 40, 8, 88];**

**function checkAdult(age) {**

**return age > 17; // test: age should be greater than 17**

**}**

**ages.filter(checkAdult); /\* checkAdult is a callback here \*/**

**Output: (4) [32, 18, 40, 88]/\* return new unnamed array with 4 elements, all elements are greater than or equal to 18 \*/**

# **\*\*You can store all new values in one array; just assign your filter expression to any variable you want.**

# ****Ex:**** How can we extract odd numbers from an array using filter function? **Ans: We are extracting odd values from ages array, and storing those odd values in newArr array. var ages = [33, 13, 18, 16, 40, 8, 88];**

# **function checkOdd (age) {**

# **return age % 2 != 0;**

# **}**

# **var newArr = ages.filter(checkOdd); /\* checkOdd is callback here \*/**

# **console.log(newArr)**

# ****Output: (2) [33, 13]** -----------------------------------------------------**

**map() method(operation-based):**The map() method calls the provided callback function once for each element in an array, from left to right order, and then creates a new array after performing some kind of operation on every element of original array. This means map creates new array that has same length as original array, means if 5 elements in original array then new array will also contain 5 elements.

**Note:** map() does not execute the function for array elements without values.

**Note:** map() does not change the original array.  
  
Syntax: newArr = oldArr.map(callback);

Ex 1) Return new array with double values available in original array

var array1 = [1, 4, 9, 16];

const map1 = array1.map(function(x){ return x \* 2 });

// const map1 = array1.map(x => x \* 2); // same as above

console.log(map1);  
Output: (4) [2, 8, 18, 32] // new Array with 4 new elements[2, 8, 18, 32]  
  
Imp Note:

1. map1 is a variable which will be new array created by map method..
2. parameter x is a variable which works as a iteration variable. It loops through the original array

**Ex 2 )** Return an array with the square root of all the values in the original array:

**var numbers = [4, 9, 16, 25,144];**

**console.log(numbers.map(Math.sqrt))**

**Output: (5) [2, 3, 4, 5, 12]**

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

**reduce( ) method(single-value-based):**  
The reduce() method reduces the array **to a single value**.

The reduce() method executes a provided function for each value of the array **(from left-to-right).**

**Ex 1)   
var numbers = [5, 4, 7, 10];**

**function getSum(total, num) {**

**return total + num;**

**}**

**function myFunction(item) {**

**return numbers.reduce(getSum);**

**}**

**var x = myFunction(getSum);**

**console.log(x)**

**Output: 26**

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

**Q: How can we create new image using javascript?  
Ans:**

The **Image()** constructor creates a new [HTMLImageElement](https://developer.mozilla.org/en-US/docs/Web/API/HTMLImageElement) instance.  
Ex:

var myImage = new Image(100, 200); //created instance for image element   
myImage.src = ‘dummyImage.jpg’ // image source here

document.body.appendChild(myImage) // add image to body element  
  
This would be the equivalent of defining the following HTML tag inside the [<body>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/body):  
<img width=”100” height=”200” src=’dummyImage.jpg’>

\*\*It is functionally equivalent to [**document.createElement('img')**](https://developer.mozilla.org/en-US/docs/Web/API/Document/createElement)

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

**Q: Add below 3 images dynamically to your webpage using callback …  
  
**

****

**// relaxed.jpg will be shown on web-page correctly, paste this image in your system**

**Ans:  
  
<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<title>mastering callbacks</title>**

**<style>**

**img { display: block; width: 100%; margin-bottom: 15px; }**

**</style>**

**</head>**

**<body>**

**<h1>Adding 3 images dynamically using callback</h1>**

**<script>**

**var bears = ['1bear','polar-bear','relaxed'];**

**var count = bears.length;**

**bears = bears.map(function(bear) { /\* map iterates through bears array, 3 elements in bears array so callback will executed thrice, bear holds current element value \*/**

**var img = new Image(); /\* on every iteration of map method, we are creating instance for img element \*/**

**img.onload = function(){ /\* callback function, executing inside another function, on every img instance onload event, this will be called \*/**

**next();**

**}**

**img.src = bear + '.jpg';**

**/\* after ‘count-- ' statement execute, this statement will execute, it will assign value for src attribute of image we have creted at first line of map method \*/**

**console.log(img); // this will display all the img elements with src attributes   
  
return img;**

**/\* return statement will execute thrice it’s because map method will iterate thrice due to 3 items available in original bears array, and every return value will be assigned to bears itself, after map iteration over, img variable will contain three img element with their respective src attribute, at first iteration of map method, first image will be created, then at second iteration second image will be created and so on \*/**

**})**

**function next(){**

**count--; /\* after this statement, control will pass to map method where we set attribute value for the image, also it will check for below condition too.. \*/**

**if(count < 1) {**

**bears.forEach(function(bear) {**

**console.log(bear.width) // check image width**

**document.body.appendChild(bear); // append images to body**

**})**

**}**

**}**

**console.log(bears); /\* now bears is an new array created by map method which holds three img elements.. \*/**

**</script>**

**</body>**

**</html>**

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

**Q: What would be the output if we use plus operator between two different types?  
Ans:**

* Number + Number -> Addition , For ex 2 + 2 = 4 [type number]
* Boolean + Number -> Addition, For ex. true + 1 = 2 [type number]
* Number + String -> Concatenation, 2 + “2” = “22” [type string]
* String + Boolean -> Concatenation, “2” + true = “2true” [type string]
* String + String -> Concatenation, “2” + “2” = “22” [type string]

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

**function calculateSalary(){**

**return this.salary \* 12; /\* salary variable is not declared in function \*/**

**}**

**var empObj = {**

**Ename: "Amol",**

**salary: 40000,**

**AnnualSalary : calculateSalary /\* assigning calculateSalary function to AnnualSalary, calculateSalary is a function in global scope \*/**

**};**

**alert( empObj.Ename + " earns " + empObj.AnnualSalary() + " per annuam " )  
  
Output:  
Amol earns 480000 per annuam**

**Note:**

**1) calculateSalary function is in global scope so we can access it everywhere in our code**

**2)”*this*” points to current context hence *this* in AnnualSalary method will point to its parent object i.e. empObj.**

**3)\*\* calculateSalary is property of window object, it’s not property of empObj, so we cannot call empObj.calculateSalary();**

**-------------------------------------------------------------------  
  
Q: functions as array elements: how can we call functions declared as array element?  
Ans:**

**var arr = [];**

**function first() { /\* named function \*/**

**console.log("first function")**

**}**

**arr[0] = first; // assign function to array element**

**arr[1] = function(){ // function expression, assign unnamed function**

**console.log('Second function')**

**}**

**arr[2] = function(){**

**console.log('Third function')**

**}**

**for(var i=0; i<arr.length; i++) {**

**arr[i]();**

**}**

**Output:  
First function**

**Second function**

**Third function**

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

**Q: Assign anonymous function to outer function inside same outer function:  
  
function doSomething(a,b){**

**function displaySum(){**

**alert("Sum is:" + (a+b))**

**}**

**displaySum(); // calling displaySum**

**/\* assign anonymous function to outer function, this simply means after executing below statement, outer function will hold below value i.e. below anonymous function and it will forget all above statements \*/**

**doSomething = function(a,b) { // assign anonymous function to outer function**

**alert("difference is " + (a-b));**

**}**

**}**

**doSomething(10,20) // at first call, displaySum will get executed and JavaScript will //assign anonymous function to doSomething here**

**doSomething(10,20) // anonymous function will get executed now onwards**

**doSomething(100,150) // anonymous function will get executed**

**Output:  
Sum is:30**

**difference is -10**

**difference is -50**

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

**Q: Assign inner function to outer function inside same outer function:**

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

**function displaySum(){**

**alert("Sum is:" + (a+b))**

**}**

**function displayMultiplication(){ // this will execute only once**

**alert("Multiplication is:" + (a\*b))**

**}**

**displaySum();**

**displayMultiplication(); // this will get call only once**

**doSomething = displaySum; // assign inner function to outer function**

**}**

**doSomething(10,20) //**

**doSomething(10,20) //**

**doSomething(100,150) //  
Output:  
Sum is:30  
Multiplication is:200**

**Sum is:30**

**Sum is:30**

**Q: function returning function  
function doSomething(a,b) { // outer function**

**function displaySum(){**

**console.log("Sum is " + (a+b).toString());**

**}**

**function displayDiff(){**

**console.log("Difference is " + (a-b).toString())**

**}**

**displaySum(); // it will execute displaySum function**

**return displayDiff; // it will execute displayDiff function, don’t use parenthesis**

**}**

**var a = doSomething(10, 20);**

**a();**

**Output:  
Sum is 30**

**Difference is -10**

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

**Q: Explain functions in JavaScript?**

1. **A function is block of executable code**
2. **Function gets executed when it gets called**
3. **Function can have parameters**
4. **Function can return values back**
5. **JavaScript functions can be loosely classified as the following::**

**a) Tradional way, called as function declarations:  
For Ex: function functionName() { function body }  
  
b) using function expressions:   
For Ex: var variableName = function(a,b) { function body }**

**6) Functions can be assigned to another variable,**

**For ex:**

**function doSomething(a,b) { function body }**

**var f = doSomething;**

**f(10,20);**

1. **The difference between function declaration and function expression is:  
   if we declare function using function declaration, that function we can call before its declaration. This process is called as function hoisting. But we cannot call function expression before its declaration.**
2. **Functions can be passed as an argument. We mainly use this technique in callbacks. Callback involves execute a function inside another function.**
3. **If we want function can invoke itself and execute immediately without waiting for calling anywhere, then we can use IIFE. IIFE stands for immediately invoked function expression.**

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

**Q: Explain Arrays in JavaScript?  
Ans:**

1. **Array is a variable which can hold multiple values in a single time.**

**2) We can access every item/ element in array using its index. Index means the position where the array element is actually stored. All indexes are numerical. Array index starts with zero means first element will always have index zero and second element will have index 1 and so on.**

**3) JavaScript array can hold multiple types of values, means within a array, we can store value that is associated with any type like number or string or Boolean or function etc. This facility is not available in another programming languages, we can store values associated with only one type.   
  
4) We use length property to get length of an array. Using length property and for loop we can display all elements of an array.**

**5) Associative arrays: Javascript arrays can have string-based/key-based indexes too; this means if we don’t need array with numerical indexes, we can assign string-based indexes to our arrays. Like emp[‘name’] = “amol”; here emp is array and name is its string-based-index. String-based-index is also known as its key. Such String-based-arrays are also called as Associative arrays.**

**6) through for-in loop, we can get all the keys available in associative array.  
  
For ex:  
var empdata = []**

**empdata['name'] = "amol"**

**empdata['salary'] = 45000;**

**for(key in empdata) {**

**console.log(key)**

**}**

**Output:**

**name**

**salary**

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

**Q: What would be the output below code will produce?  
  
Ans:**

**var empdata = []**

**empdata[3] = "amol" // value amol is stored at 3rd index**

**console.log(empdata.length);**

**console.log(empdata[0])  
console.log(empdata[1])**

**console.log(empdata[2])**

**Output:   
4 // array contains four elements with index 0, index 1, 2 and 3**

**undefined // we have not assigned any value for zero-indexed-element  
undefined // we have not assigned any value for first-indexed-element  
undefined // we have not assigned any value for second-indexed-element**

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

**Q: Explain public and private members?  
Public members: Members with ‘this’ are accessible everywhere in your code. Means public variables can be accessible outside the object where they have declared.**

**Private members: Members with ‘var’ are accessible only within the object. Means private variables can be accessible only inside the object where they have declared.  
  
Ex:   
<script>  
var emp = function(empName, monthlySalary){**

**this.Ename = empName;**

**var salary = monthlySalary;**

**this.annualSalary = 0.0;**

**this.calcAnnualSal = function(){**

**this.annualSalary = salary \* 12;**

**}  
  
 var pvtmethod = function(){**

**console.log(salary) /\* remember private method can access private variable \*/**

**}**

**}**

**var empObj = new emp('amol', 40000);**

**console.log(empObj.annualSalary); // 0, default value is 0, public member**

**empObj.calcAnnualSal(); // calling calcAnnualSal method, public member**

**console.log(empObj.annualSalary); // 480,000, public member**

**console.log(empObj.salary); // undefined, private variable**

**console.log(empObj.Ename); // amol, public variable , accessible everywhere**

**console.log(empObj.pvtmethod()); //error: empObj.pvtmethod is not a function, its //because it is not accessible outside the object, within object we can access it/ call it, if we call private function within its object, in that case it will work**

**</script>**

**--------------------------------------------------  
  
Conditional Callbacks:  
Nested callbacks are used to execute a particular code/particular function, if certain condition is true. If condition is false then that code/ that function will not be executed.**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<title>Magic of Nested Callbacks</title>**

**</head>**

**<body>**

**<script>**

**let students = [**

**{ name:"Mary", score: 90, school: "East" },**

**{ name:"James", score: 100, school: "East" },**

**{ name:"Steve", score: 40, school: "East" },**

**{ name:"Gabe", score: 88, school: "West" },**

**{ name:"Rachel", score: 85, school: "East" },**

**{ name:"Rodger", score: 95, school: "West" },**

**{ name:"Lary", score: 75, school: "East" }**

**];**

**/\* let processStudents = function(data, callback) { // this is also acceptable \*/**

**function processStudents (data, callback) {**

**for (let i=0; i<data.length; i++) {**

**if(data[i].school.toLowerCase() === 'east') {**

**if(typeof callback === 'function') { // you can specify any condition here like 1 < 2, if this condition is true only in that case, callback function will get executed/get called..  
 // console.log("a") // this will print 5 times 'a' because we have 5 iterations in array , 5 students // from East school, this means callback will get executed 5 times whenever we call it**

**callback(data[i]) // callback function called here and passing current**

**// array element**

**}**

**}**

**}**

**}**

**processStudents(students, function(obj) { // second paremeter is callback**

**// function**

**if(obj.score > 60) {**

**console.log(obj.name + " passed.")**

**}**

**});**

**let determineTotal = function() {**

**let total = 0,**

**count = 0;**

**processStudents(students, function(obj) { // second paremeter is callback**

**// function**

**total = total + obj.score;**

**count++;**

**});**

**console.log('Total score from EAST school is ' + total + " & Total Students are " + count);**

**}**

**determineTotal()**

**</script>**

**</body>**

**</html>**

**---------------------------------------------------------  
  
Q: What is callback hell? OR  
Q: What are nested callbacks? OR  
Q: Why promises are better than callbacks?  
Ans:**When using single-threaded asynchronous programming, there are two main helpful techniques, Promises and callbacks. The most common one is using callbacks.   
  
JavaScript Promises use callback functions actually to specify what to do after a Promise has been **resolved** or **rejected**, so the two are not fundamentally different. The main idea behind Promises is to take callbacks - especially nested callbacks where you want to perform a series of actions.

If a single callback versus a single promise is there, it's true there's no significant difference. It's when you have a zillion(zillion means an extremely large number of people or things) callbacks versus a zillion promises that the promise-based code tends to look much nicer.

**For ex: let’s say, in jQuery, we have 3 nested AJAX requests to fetch data from server, if first request is succeed then we are making second nested request using nested callback and if second request succeed then we are making third nested request using another nested callback. Since jQuery callback use $.ajax method to make ajax requests, we will have to write $.ajax and within $.ajax, we will have to put success and error methods. On success we will make another nested ajax request using nested callback. And on error, we will display appropriate error message. Means three success methods and 3 error methods nested in each other. This way we will find our code looks ugly and difficult to understand and hard to maintain. This process is called as ‘callback hell’.**

**But,  
By using promises, we can make our code nicer and maintainable than callback hell. Since Promises can be chained**, and chains can be extended at any time more easily. In Jquery promises, we use $.get method to make ajax request and then you can use then methods to achieve the same and every ‘then’ method will return some value(in this case, value will be the file path we are requesting from server) to next ‘then’ method. And if any error occurs you no need to throw error message for every request, only one error handler we can use to handle the error situation.   
  
 **So in short, promises are indeed better if multiple nested callbacks are there inside nested callbacks to avoid callback hell like situation, handling errors are very easy in promises and promises makes your code easy to understand, easy to testing and more maintainable.**  **----------------------------------------------------------**

**----------------------------------------------------------  
\*\*What is the difference between Synchronous & Asynchronous Programming?  
  
Synchronous Way:** **It waits for each operation to complete, after that, it executes the next operation**.   
  
See In below example, second function will not be executed until & unless first function has finished its execution and third function will not be executed until second function has finished its execution. It’s because in **Synchronous programming, operations are executed in sequential manner. We have called function first at very first and function third at very last thus function first will execute first and function third will execute at very last.**  
  
Ex:  
function first() {

return console.log("first")  
}  
function second() {

return console.log("second")

}

## function third() {

## return console.log("third")

## }

## first();

## second();

## third(); Output: first

## second

third  
  
**Asynchronous way:** It never waits for each operation to complete, rather it executes all operations in the first GO only. **The result of each operation will be handled once the result is available.**See in below example, you will see though we are calling function first at very first still function third and function second is executing before the function first. It’s because in asynchronous, one operation does not wait for another operation to complete. Those operations will execute first, that takes less time/duration than another. **Ex:**function first() {

setTimeout(function(){ return console.log("first")}, 2000)

}

function second() {

setTimeout(function(){

return console.log("second")},1000)

}

function third() {

setTimeout(function(){ return console.log("third")},0)

}  
  
first();

second();

third(); **Output:  
third**

## second

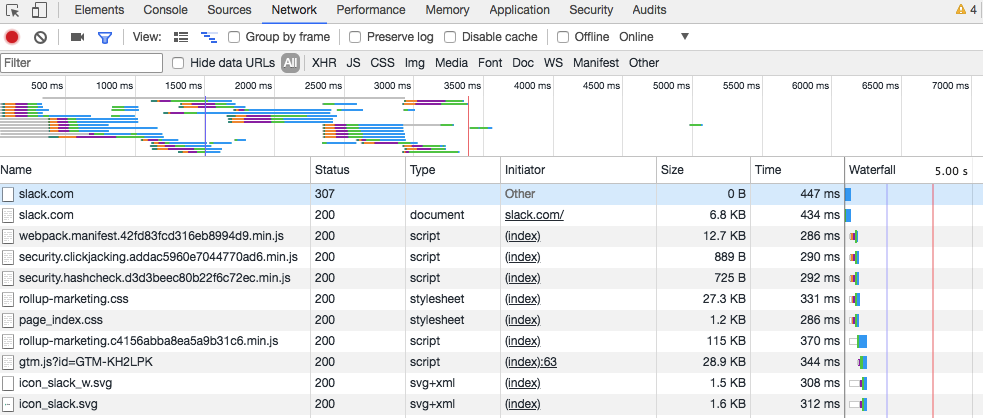
first **-----------------------------------------------**How to improve loading speed of website?  
According to Google, your site needs to be finished it’s loading in three seconds.   
 **1. Minimize HTTP requests**

According to Yahoo, [80% of a Web page’s load time](https://developer.yahoo.com/performance/rules.html#num_http) is spent downloading the different parts of the page, like images, stylesheets, and scripts.

The more http requests, the longer it takes for the page to render.

If you use Google Chrome, you can use the browser’s Developer Tools to see how many HTTP requests your site makes.

Open “Inspector,” then click the “Network” tab, you will see the requests made by page; with the duration they took to load the file.



**2. Minify and combine files:**

Minifying a file involves removing unnecessary formatting, whitespace, and code.

Since every unnecessary character increase the size of your page, it’s important that you eliminate extra spaces, line breaks, and indentation. This ensures that your pages are as lean as possible.

Combining files is exactly what it sounds like. If your site runs multiple CSS and JavaScript files, you can combine them into one.

## 3. ****Reduce image sizes:**** The larger your content, the slower the site.

Some basic ways to counteract this is by shrinking the file sizes of images on your site, reducing the number of images you use, or eliminating them altogether. But having no images on your site is boring! Rather than removing them, optimize images before uploading them to your site by:

1. **Changing the resolution:** reducing the “quality” of the image (and thereby the file size)
2. **Compressing the picture:** increasing the efficiency of image data storage
3. **Cropping the picture:** when cropping, you are cutting out unneeded areas and thus making the image smaller in size

## You can make these kinds of changes in a premium tool like Photoshop, or a free program like [Gimp](http://www.gimp.org/). There are even in-browser tools like [picresize.com](http://picresize.com/).

## 4) Browser caching for website speed:

Browser caching is when a visitor is able to keep the downloadable files from your website stored in their browser. This way the next time that person visits your website the page will load faster, as they will only have to download the updates that have been made to your website because they already have the rest of the files downloaded.

## The easiest way to enable browser caching on your website is through plugins.

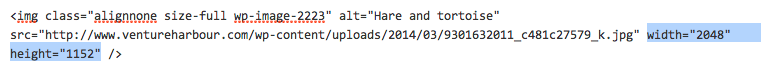
## ****5. Enable compression using Gzip like tool:****

Compression of files is one of the easiest ways to reduce load times, and today, enabling compression with [**Gzip**](https://www.gnu.org/software/gzip/) is considered standard practice.

## This works well with CSS and HTML, because these files typically have repeated code and whitespace.

## 6.Specify image dimensions:

Before your browser can display your webpage, it has to figure out how to lay out your content around your images. It takes longer if you don’t specify your image width and height in image tag.



Specifying your image dimensions in img tag, saves the browser time compare to specifying image dimensions in CSS, which helps to render your page quickly.

**7. Put CSS at the top and JS at the bottom**

## It’s considered best practice to put your CSS as close to the top of your page as possible, as browsers won’t render your page before rendering the CSS file. Javascript, on the other hand, should be as close to the bottom as possible, since it prevents browsers from parsing after the tag before it is loaded and complete.

**Alternative way to load at bottom:**

## C:\Users\lenovo\Desktop\defer-js.png 8. Use CSS Sprites:

A sprite is one big image that contains all of your images. Using CSS you can ‘hide’ everything in the image and you can show the required image using its co-ordinates.

CSS sprites work because it’s faster to load one big image than loading lots of smaller images. This is because the browser doesn’t need to make as many requests.

[SpriteMe](http://spriteme.org/) is a tool that enables you to easily turn all of your images into a CSS sprite.

## 9. Include all external style-sheets using link tag, don’t use @import

The @import rule allows you to import a style sheet into another style sheet. The @import rule must be at the top of the document.

But, CSS @import takes more time to load than including external CSS file. Another reason why not to use @import is that some of the older browsers do not support it, so it’s best to avoid it where possible.

10) **Reduce server response time:**

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

**\*\*What does it mean “!x”?  
It means:  
Either x is zero or undefined or null or empty string or false.**

**In these 5 cases this expression will evaluate to true.**

**Check below example for better understanding:  
Ex:**function myFunction(x){

if(!x){

console.log("falsey value " + x)

}

else {

console.log("truthy value " + x)

}

}

myFunction(0)

myFunction()

myFunction(null)

myFunction("")

myFunction(false)

myFunction("amol")

myFunction(true)

myFunction(1) **Output:  
falsey value 0**

**falsey value undefined**

**falsey value null**

**falsey value // empty string cannot be displayed**

**falsey value false**

**truthy value amol**

**truthy value true**

**truthy value 1**

**Note: Remember zero and negative zero both are same.**

**----------------------------------------------------------------------------------  
Explain output for below code:  
Ans:**function myFunction (x, y) {

if (x == undefined || x == null) {

this.x = 50;

}

else {

this.x = x;

}

if (y == undefined || y == null) {

this.y = 50;

}

else {

this.y = y;

}

//method

this.myMethod = function() {

return "[" + this.x + ", " + this.y + "]";

}

}

var f = new myFunction(); // not passing any argument

console.log("Our arguments are: " + f.myMethod()); **Output: Our arguments are: [50, 50]**

Note: since we are not passing arguments to myFunction, undefined will evaluate to true; and it will show default arguments i.e. 50,50

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

**Constructing an Object Using a Generic Object Constructor**

Mostly used way to create an object literal is to start with an empty object, and then add properties to it. You can create an empty, generic object in one of two ways:

var obj1 = { };

var obj2 = new Object();

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

**What will be the output for below code?**function makeObj() {

# return { x: 1 }; // returning object

# }

# var myObj = makeObj(); /\* myObj is a var that stores the value that makeObj returns \*/

# console.log(myObj)

**Output:** {x: 1} **// we are returning object through the function; we can return number or string or expression etc.  
  
We already aware, we can assign any function to a variable. And in that case, that variable will hold the value which is returned by the function.**\*\*Note, in above example, **makeObj()** is *not* a constructor function because we didn't call it with **new**.

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

# \*\* Objects and their Truthy-ness: Check carefully below outputs:

# Ex: 1) var p = {}; // empty object

# if (p) {

# console.log("This object is truthy!");

# }

# else {

# console.log("This object is falsey!");

}  
  
Output: This object is truthy!  
(Remember that empty strings are falsey, so you might expect that empty objects are also falsey, but above example shows you how this is not correct)

Strangely, Objects returns ‘false’ in most of the cases. Check below outputs returned by objects

# Ex: 2)

# var p = {};

# console.log(p === true) // returns false

# console.log(p === false) // this is also false

# Output: false

# false

# Ex: 3) var p = {};

# console.log(p == 0)

# console.log(p == null)

# console.log(p == undefined)

# console.log(p == "{}")

# console.log(p == {})

Output:  
false

false  
false

false  
false

false

# Ex: 4) var book1 = {

# title: "Harry Potter",

# author: "JK Rowling"

# };

# var book2 = {

# title: "Harry Potter",

# author: "JK Rowling"

# };

# if (book1 == book2) {

# console.log("The two books are the same");

# }

# else {

# console.log("The two books are different");

}  
  
Output: The two books are different  
  
The two books, **book1** and **book2**, are exactly the same: they have the same properties. All the property-names are the same, the property values are the same, and the number of properties is the same. So why aren't they equal?   
  
The two objects are not equal because of an important difference in how primitive values are stored and how objects are stored in the computer's memory. When you create a primitive value, let's say:

var x = 3;  
the computer allocates a bit of memory, gives it the name "x", and saves the value 3 in that bit of memory:  
Now, when you compare two objects, let's say **book1** from the example above:  
In this case, the “**computer allocates some memory for each of the properties in the object**, **and then allocates a separate bit of memory for the variable name**, **and in that memory stores a value that points to the place in memory where the object is actually stored.”** This is called an **object reference**. So the variable **book1** doesn't contain the object itself; it actually contains a *reference* to the object.  
Now let's see what happens when we create the second book object, **book2**:  
Even though the properties are exactly the same, a completely separate book object is created and stored in a completely different part of memory,  
So when we compare **book1** and **book2**:  
the values that are compared are the memory locations of the two objects. They are *not* equal, so we see the message "The two books are different."

Ex 4)

var book1 = {

title: "Harry Potter",

author: "JK Rowling"

};

var book2 = book1; // shallow copy

if (book1 == book2) {

console.log("The two books are the same");

}

else {

console.log("The two books are different");

}  
  
Output: The two books are the same

\*\*\*Remember, if we compare our object with its shallow copy, only in that case, comparison will return true. It is because they both points to similar location in memory.

---------------------------------------------------------------------------------------------------------------------------------------  
  
Q: Method nested in object, ‘this’ refers to which context? And how can we call method available in object?   
Ans:

var person = {

name: "James T. Kirk",

ship: {

name: "USS Enterprise",

},

getInfo: function() {

return this.name + " commands the " + this.ship.name;

}

};

person.getInfo();  
  
Output: "James T. Kirk commands the USS Enterprise"

-----------------------------------------------------------------------------------  
  
  
Q: Explain output for the below code:  
  
var valid\_num = parseInt("16");

var invalid\_num = parseInt("I'm not a number!");

console.log(valid\_num)

console.log(typeof valid\_num)

console.log(invalid\_num)

console.log(typeof invalid\_num)  
  
Output:  
16

number

NaN

number  
  
Note: Remember, parseInt() method convert your string into number

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

Q: How can we display addition of all digits/ numerals available in array?  
  
Ans:  
var arr = [10,20,"amol",4,true,15], result = 0;

for(i=0; i<arr.length; i++){

if(typeof arr[i] == "number") {

result += arr[i]

}

}

console.log(result)  
  
 Output: 49   
--------------------------------------------------------------------------------

**Shallow Copy:**  
It copies address of original object into duplicate object. Suppose original object is X and duplicate object is Y; then Y copies X’s address. So the addresses of X and Y will be same i.e. they will be pointing to the same memory location. Hence change in x object will get reflected into y, and vice-versa.  
  
var x = {a:10, b:20, c:30}

# var y = x;

# y.a = 100;

# x.b = 200

# console.log(x)

# console.log(y) Ouput: {a: 100, b: 200, c: 30}

# {a: 100, b: 200, c: 30} -------------------------------------------------------------------

# Ex 2)\*\*Shallow copy of constructor function’s object: function Book(title, author) {

# this.title = title;

# this.author = author;

# }

# var book1 = new Book("Harry Potter", "JK Rowling");

# var book2 = book1;

# if (book1 == book2) {

# console.log("book1 is equal to book2");

# } else {

# console.log("book1 is NOT equal to book2");

# }

# book1.newProp = "abc";

# book2.anotherNewProp = "xyz";

# console.log(book1);

# console.log(book2); Output:

# book1 is equal to book2 Book {title: "Harry Potter", author: "JK Rowling", newProp: "abc", anotherNewProp: "xyz"}

# Book {title: "Harry Potter", author: "JK Rowling", newProp: "abc", anotherNewProp: "xyz"}

# Note: book1 & book2 are instances of book constructor, so if in console we are displaying instances; it returns its prototype along-with its all member. Ex 3) shallow copy example 3rd: var x = {a:3,b:4,c:5,d:6}

# var y = {a:3,b:4,c:5}

# console.log(x==y) // false

# x = y;

# console.log(x==y) // true

# y.a = 30;

# console.log(x) // all properties and values inherited by y object

# console.log(y) // all properties of y object console.log(x.d) // undefined

# Output: false

# true {a: 30, b: 4, c: 5}

# {a: 30, b: 4, c: 5}

# undefined

# ---------------------------------------------------------- Deep copy

A deep copy copies all fields from original object into duplicate object, but it allocates different memory for duplicate object. So any change in original object will not affect in duplicate object and vice versa.

Ex:   
var x = {a:10, b:20, c:30}, y;

for(prop in x) {

y[prop] = x[prop]

}  
**//** var y = { a:x.a, b:x.b, c:x.c } this also creates deep copy

y.a = 100;

x.b = 200;

console.log(x)

console.log(y)  
  
Output:  
{a: 10, b: 200, c: 30}

{a: 100, b: 20, c: 30}

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

querySelector & querySelectorAll

Que: What is exactly difference between querySelector and querySelectorAll?

Ans: Through querySelector, we can select only one single element, for ex. if our DOM contains 10 input fields, in that case querySelector will select only one element i.e. very first input field.

Whereas querySelectorAll is used to select all elements of similar type. This means if our DOM contains 10 input fields in that case through querySelectorAll we can select all those 10 inputs. querySelectorAll crates array like object and within that array, it stores references of all those elements that matches with the case defined in parenthesis.

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

Que: How can we use querySelectorAll to select all elements of similar type?  
Ans: We use for loop along with querySelectorAll to select all elements of similar types.

1. **For ex. To select all DIV’s available in DOM, we will have to write below code,**

**var getDiv = document.querySelectorAll("div"); /\* it creates nodeList that contains all the Div nodes/elements, nodeList is an array like object, it’s not an array \*/**

**for(var i=0; i<getDiv.length; i++){**

**getDiv[i].addEventListener("click", function(){**

**console.log(this.textContent) ;**

**});**

**}  
Output: This will return texts available for clicked DIV element**

b) To select all input elements available in DOM, we will have to write below code,

var getInput = document.querySelectorAll("input");

for(var i=0; i<getInput.length; i++){

getInput[i].addEventListener("focus", function(){

console.log(document.activeElement.tagName) ;

});

}  
**Output: This will return tagname i.e. INPUT, for currently focused input…..**

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

Que: Describe some commonly used <input>types and what if we do not place them inside form tag?

Ans:A web form consists of any number of input fields grouped in a <form>tag. A lot of field types use the <input> tag. This tag’s type attribute is used to select the field’s style. These are some commonly used <input> types:

text: A single-line text field

password: Same as text but hides the text that is typed

checkbox: An on/off switch

radio: (Part of) a multiple-choice field

file: Allows the user to choose a file from their computer

Form fields do not necessarily have to appear in a <form> tag. You can put them anywhere in a page. **Such fields cannot be submitted**.

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

***Focus[Set focus and remove focus]:***

Que: How can we set focus for any input field?

Ans: We can control focus from JavaScript with the focus and blur methods.We use focus method to set focus as follows:

document.querySelector("input").focus();

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

Que: How can we set focus for any input field?

Ans: we use blur method to remove focus as follows: document.querySelector("input").blur();

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

Que: How can we get currently focused element through javascript?  
Ans: “document.activeElement” returns currently focused element.

For Ex., You will get tagname of currently focused element as follows:

document.activeElement.tagName;  
**----------------------------------------------------------------------------------------------------------**

Que: What is autofocus?

autofocus is the HTML attribute which is used to set focus for particular element when browser finished loading.

Suppose your DOM contains 4 input fields and you want to set focus for last input after loading the page, then through autofocus we can simply achieve this.

Que: What is tabindex?

By default, most types of HTML elements cannot be focused. But you

can add a tabindex attribute to any element, which will make it focusable.

We can change/influence the order in which elements receive focus with the tabindex attribute. The following example document will let focus jump from the text input to the OK button, rather than going through the help link first:

**<input type ="text" tabindex =1 >**

**<a href ="#" >help</a >**

**<button tabindex = 2 > OK </button>**

**If we remove tabindex from input and button, focus will jump to input first, then it moves to anchor and finally will move to button.**

----------------------------\*\*\*\*\*\*\*\*\*\*@@@@@\*\*\*\*\*---------------------------------------

**Disabled fields:**

All form fields can be ***disabled*** through their disabled attribute, which also exists as a property on the element’s DOM object.

< button>I ' m all right </ button >

< button disabled>I ' m out </ button >



Disabled fields cannot be focused or changed, and unlike active fields, they usually look gray and faded. When a program is in the process of handling an action caused by some button or other control, which might require communication with the server and thus take a while, it can be a good idea to disable the control until the action finishes. That way, when the user gets impatient and clicks it again, they don’t accidentally repeat their action.

Que: What is elements property of form element?

Ans: The<form>element, has a property called elements that **contains an array-like collection of the fields inside it.** This means if form contains 10 fields, we can access those fields using index numbers with the help of elements property, exactly like we access array elements using index numbers.

<form action ="example/submit.html">

Name : <input type =" text " name ="name"><br><br>

Password: <input type ="password" name ="password" ><br><br>

checkBox: <input type ="checkbox" name ="check" ><br><br>

<button type ="submit" >Log in</ button >

</form>

<script>

var form = document.querySelector("form") ;

console.log(form.elements[0].type ); //accessing field type using index number

console.log(form.elements[1].type ); //accessing field type using index number

console.log(form.elements[2].type ); //accessing field type using index number

console.log(form.elements.password.type); // accessing field type using field name

console.log(form.elements.check.type); // accessing field type using field name

</script >

Output: text

password

checkbox

password

checkbox

Que: “submit a form” What does it means?

Submitting a form normally means that the browser navigates to the page indicated by the form’s action attribute, using either a GET or a POST request. But before that, a "submit" event gets fired.

Que: How can we retrieve checkbox checked and unchecked case using JavaScript?

Ans: Below we are assigning yellow background if checkbox is checked and background white if it is unchecked:

**<input type ="checkbox" id ="yellow">**

**<label for="yellow">Click here to make this page Yellow</label>**

**<script>**

**var checkbox = document.querySelector("#yellow") ;**

**checkbox.addEventListener("change", function() {**

**document.body.style.background = checkbox.checked ? "yellow" : "";**

**});**

**</script>**  
-------------------------------------------------------------------------------------------------------------------------------------------

Que: What is getElementsByName?  
Ans:We use name attributes for **form input fields.getElementsByName**is used to select elements whose values for **name** attributes are similar. The document.getElementsByName method gives us all elements with a given name attribute.

For Ex: Below we are accessing inputs through getElementsByName and changing body background-color as per radio button value:

<input type ="radio" name ="color" value ="mediumpurple" >Purple

<input type ="radio" name ="color" value ="lightgreen" >Green

<input type ="radio" name ="color" value ="lightblue">Blue

<script>

var buttons = document.getElementsByName("color");

function setColor (event) {

document.body.style.background = event.target.value;

}

for(var i = 0; i < buttons.length; i++)

buttons[i].addEventListener("change" , setColor) ;

</script>

**Select fields**

Select box is commonly like a *drop-down* control that shows the options only when you open it.

<select>

<option > Pancakes </ option>

<option > Pudding </ option>

<option > Ice cream </ option>

</ select >  
  
When given the multiple attribute, a <select>tag will allow the user to select any number of options, rather than just a single option. Hold Ctrl key to select multiple options.  
<select multiple></ select >  
  
**size attribute:**The “size”attribute to the <select>tag is used to set the number of options that are visible at the same time. For example, setting the size attribute to"3" will make the field show three lines, whether it has the multiple option enabled or not.

value attribute:Each <option>tag has a value. This value can be defined with a “value attribute”, but when that is not given, the text inside the option will count as the option’s value.  
-----------------------------------------------------------------------------

Can we concatenate [or make addition of] values of selected options for select Box having multiple selection option?  
Ans: Yes. Below we are making addition of every value for each selected option.  
  
<select multiple>

<option value="1">First Item</option >

<option value="2">Second Item</option >

<option value="3">Third Item</option >

<option value="4">Fourth Item</option >

<option value="5">Fifth Item</option >

</select>

=<span id ="output" >0</span>

<script>

var select = document.querySelector("select") ;

var output = document.querySelector("#output") ;

select.addEventListener("change" , function() { // loop through every option

var number = 0;

for ( var i = 0; i < select.options.length; i++) {

var option = select.options[i];

if ( option.selected )// check whether option is selected or not, using selected attribute..

number += Number(option.value) ;

}

output.textContent = number;

}) ;

</script>  
  
----------------------------------------------------------------------------------------------------------------------------------------------

Que: What will be the output for below code?

<body>  
<script>

document.body.style.background="blue";

setTimeout(function(){

document.body.style.background="yellow";

},2000);

</script>  
</body>  
  
Output:This page turns from blue to yellow after two seconds. After finish page loading, we will find, page background is blue for first two seconds and then background will turn to yellow.

**Keydown & keyup events:**keydown event: When a key on the keyboard is pressed, your browser fires a "keydown"event.   
  
keyupevent: When key is released [after pressing], a "keyup" event fires.

event.keyCode:event.keycode returns Unicode character code associated with the key is being pressed or released.

For ex. Unicode character code for key ‘a’ is ‘65’. If you want to fire an event if user is pressing key ‘a’, you can write like if(event.keyCode==65) in conjunction with keyup or keydown event.  
  
Ex: Below code will assign background-color violet when you hold the V key. And when you release the key, background color will turn to white.   
Solution:  
<body>

<p>This page turns violet when you hold the V key.</p>

<script>

addEventListener("keydown",function(event){

if(event.keyCode==86)

document.body.style.background="violet";

});

addEventListener("keyup",function(event){

if(event.keyCode==86)

document.body.style.background="";

});

</script>

</body>  
Note:document.body.addEventListener and addEventListenerboth are similar. We can omit document.body if we want to.   
  
**-----------------------------------------------------------event.preventDefault():**

Many events have a default actions associated with them. If you click a link, you will be taken to the link’s target. If you press the down arrow, the browser will scroll the page down. If you right-click, you’ll get a context menu. And so on.

If the handler doesn’t want the normal behavior to happen, we can call the **preventDefault** method on the event object. For example, here is a link that cannot be followed:

<body>

<a href="https://developer.mozilla.org/">MDN</a>

<script>

var link=document.querySelector("a");

link.addEventListener("click", function(event){

console.log("Nope.");

event.preventDefault();

});

</script>

</body>  
  
Note: Try not to do such things unless you have a really good reason to. For

People using your page, it can be unpleasant when the behavior they expect is broken.

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

***event.target:***

**Q:** How can we identify exact element from list of similar elements? Suppose there are 3 buttons, how can we get which exact button we have clicked? ***Solution:***We can use event.target to get exact element from list of similar elements. Below we are displaying label for that button which we are clicking.  
  
<body>

<button>Button A</button>

<button>Button B</button>

<button>Button C</button>

<script>

document.body.addEventListener("click" , function(event) {

if (event.target.nodeName == "BUTTON")

// nodeName must be in uppercase letters, lowercase will not work

console.log("Clicked" , event.target.textContent);

}) ;

</script>

</body>

Note: We are finding target element through nodeName/Tag-Name. We can even use Class-Name instead of nodeName. If we are finding target via className, we will have to use **event.target.className.**

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

***event.which:*** If we want to know ***which***mouse button was pressed, we can use event object’s **which** property. This means through event.which, we can identify whether we have clicked left-button, right-button or middle button of the *mouse* on any DOM Element.

Que: **Write a code to detect which mouse button we have clicked for a button element?**

Solution:  
<body>

<button>Click me any way you want..!!</button>

<script>

var button=document.querySelector("button");

button.addEventListener("mousedown",function(event){

// mousedown and mouseup both events are acceptable

if (event.which==1)

console.log("Left button");

else if (event.which==2)

console.log("Middle button");

else if(event.which==3)

console.log("Right button");

});

</script>

</body>

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

Write a function which returns total ‘p’ [Paragraph] elements available in DOM. And also it needs to return all ‘p’ elements having class ‘animal’.

Solution:

<body>

<h1>This is h1 element</h1>

<p>This is first p element with no ID Or class</p>

<span class ="animal”>This is span with animal class</span>

<p>This is second p element with no ID Or class</p >

<p class="animal"> This is third p element and it has animal class</p>

<script>

function count(selector) {

return document.querySelectorAll(selector).length;

}

console.log( count ("p") ) ; // All <p> elements

console.log( count ("p.animal") ) ; // P having class animal

</script>

</body>

Output:

3

1

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

### 4 imp Output’s:

### null == undefined Output: true

* **0 == 0 Output: true**

### null == 0 Output: false

### undefined == 0 Output: false

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

What will be the Output for below code:  
  
"use strict";

function Person(name) {

this.name = name;

console.log(name)

}

var ferdinand = Person("Ferdinand"); // ‘new’ keyword is not defined  
  
Output: Uncaught TypeError: Cannot set property 'name' of undefined  
  
It’s because Strict mode does not allow you to create new object of constructor without ‘new’ keyword. If you remove use strict from above code, it will output as expected i.e. “Ferdinand”.

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

Write a program to print below pattern:

**# # # #**

**# # # #**

**# # # #**

**# # # #**

**# # # #**

**# # # #**

**# # # #**

**# # # #**

Solution:

function printPattern(){

var result = ""; /\* empty string \*/

for(var x=0; x<8;x++){

for(var y=0; y<8;y++){

if(x%2==0 && y%2==0 || x%2!=0 && y%2!=0 ){ result+= "#"; }

else { result+=" ";} /\* space added \*/

} // y for loop end

result+="\n";

} // x for loop end

console.log(result);

}

printPattern();

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

Note: We can use ‘in’ keyword between property and object, to check whether property lives or not in that object.  
  
Syntax: ‘propertyName’ in object /\* property-name in quotes \*/

var x = {a:10,b:20}

console.log('a' in x);

console.log('b' in x);

console.log('c' in x);  
  
Output:  
true

true

false

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

**The global object**

The global scope, the space in which global variables live, can also be approached as an object in JavaScript. Each global variable is present as a property of this object. In browsers, the global scope object is stored in the window variable.

**Ex: var myVar = 10;**

**console.log("myVar" in window) ; // Output: true**

**console.log(window.myVar); // Output: 10**

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

Q: How many arguments console.log function can take?  
Some functions can take any number of arguments, like console.log. You can pass any no. of arguments with console.log. Find below example wherein we are using 3 arguments through console.log function:

Ex:

function argCounter () {

console.log("You gave me " , arguments.length , "arguments.") ;

}

argCounter("Straw man " , "Tautology" , " Ad hominem") ;

Output:You gave me 3 arguments.

Note: Some functions accept only one argument like alert function. If you are passing more than one argument, in such case alert will accept only very first argument. And it will ignore rest arguments.

For Ex: alert("first","second","third");

Output: alert will display only first argument i.e. ‘first’ and it will simply ignore other arguments.

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

Print 3 digits always Program: We want to write a program that prints two numbers, the numbers of cows and chickens on a farm, with the words Cows and Chickens after them, and zeros padded before both numbers so that they are always three digits long.

Ans:  
function length3Always (cows, chickens) {

var cowString = String (cows) ;

while (cowString.length < 3)

// no curly braces in while, this means your while loop have single liner code

cowString = "0" + cowString; // this statement is part of while loop

console.log( cowString + "Cows") ; // this statement is **not** part of while loop

var chickenString = String (chickens ) ;

while ( chickenString.length < 3)

{

chickenString = "0" + chickenString;

}

console.log(chickenString+ " Chickens ") ;

}

length3Always (7, 11) ;

**Output:  
007 Cows**

**011 Chickens**

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

Write a program to display below interesting view by using closure:

\_ \_ \_ / ' ' ' '\ \_ \_ \_ \_ \_ \_ / '\ \_

Answer:

var landscape = function() {

var r = "";

var flat = function(size) {

for( var count = 0; count < size ; count++)

r += "\_ ";

};

var mountain = function(size) {

r += "/";

for( var count = 0; count < size ; count++)

{

r+= "'"; // inserting single quote within double quote

}

r+= "\\"; // inserting single backslash now, backslash is an escape character so need two backslashes to add single backslash

};

flat(3) ;

mountain(4) ;

flat(6) ;

mountain(1) ;

flat(1) ;

return r;

};

console.log( landscape());

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

**Q: What would be the output below code will generate?**

var x = true;

x = !x; /\* reverse of the existing value, so true becomes false \*/

console.log(x)

x = !x; /\* reverse of the existing value, so false becomes true \*/

console.log(x)  
  
Output:  
false

true

**\*\*Note: !x always return the opposite value of existing x’s value**  
------------------------------------------------------------

Crest Interview Ques:

**Que1: Take any integer value like 45679671 and insert dashes between two odd digits for the same number. For ex: output for 45679671 => 4567-967-1**

Solution:  
function addDashes(num){

/\* you need to deal with each character (each digit), so first convert your number within a string\*/

var numarray = num.toString(); // String(num) or num + ""; is also acceptable

/\* take empty array, within this array you are going to push your every character, every digit \*/

var result = [];

result.push(numarray[0]);// push first character of string at result[0] position, remember string[0] and string.charAt(0) both represents character at zero index..

for(i=1;i<numarray.length;i++){

/\* check every digit is fully divisible by 2 or not , if remainder is zero means your value is even, if value is even then directly push it to your array \*/

if(numarray[i]%2 == 0){

result.push(numarray[i])

}

else{

/\* if value of existing element and previous element is odd then push dash among them (this is what we are looking for), if condition is not satisfied means one of them is even so directly push your number to array \*/

if(numarray[i-1]%2!=0){

result.push("-");

result.push(numarray[i]);

}

else{

result.push(numarray[i]);

}

}

}

// console.log(result); // it shows your new array with dashes in it

/\* now convert your array to a single string \*/

result = result.join("");

// empty string as a separator, this will merge your string with no any space

console.log(result)

}

addDashes(45679671); // Output: 4567-967-1

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

Que2: Take any number. Write a function which return fizz if number is fully divisible by 3. If the number is fully divisible by 5 then return buzz. And return fizzbuzz if number is fully divisible by 3 and 5 as well.

**Solution:**

var divideby3 = "fizz", divideby5 = "buzz";

function fizzbuzz(num){

if(num % 3 == 0 && num % 5 == 0){ //if(num % 15 == 0 )

return divideby3 + " " + divideby5;

}

else if(num % 3 == 0 ){

return divideby3;

}

else if(num % 5 == 0 ){

return divideby5;

}

else return "Not divisible by 3 and 5, try another number " ;

}

fizzbuzz(60)

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

\*\*Web Workers:

JavaScript is a single threaded language because you cannot run multiple scripts at the same time.  
  
Suppose we have two different tasks, we are manipulating the DOM and at the same time we are trying to perform some complex calculations, in that case, complex calculations can hold to update the UI, this will make your page unresponsive. **Often we see the warning message on the page telling page is unresponsive, this happens often due to scripts that takes time to finish their execution. To run such complex scripts in the background, we can use Web Workers. A web worker is a JavaScript running in the background, without affecting the performance of the page.**

**Note: Manipulating or changing the DOM means changing your DOM using JavaScript like add or remove elements, add or remove classes, change CSS styles etc.**

Simple example for web workers:

Ex 1) Create one html file (i.e. webworker.html) and one JS file (“worker.js”) as follows:

**Webworker.html**  
<script>

if(window.Worker){ // check whether your browser supports Web worker or not

var myWorker = new Worker("worker.js"); **// worker file path here..**

var message = { addThis: { num1: 19, num2: 20} };

myWorker.postMessage(message);

myWorker.onmessage = function(e) {

console.log(e.data.repsult);

};

}

</script> // html file ends

**Worker.js file:**

this.onmessage = function(e){

if(e.data.addThis !== undefined){

this.postMessage ({ result: e.data.addThis.num1 + e.data.addThis.num2 });

}

}

// JS file ends

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

# Arrow functions [Fat Arrows]:

Arrow functions are also called as Fat Arrows. It is **alternative way** of writing function expression. An **arrow function** has a shorter syntax than a [**function expression**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/function)**.**

**Suppose you have function expression like this:**  
var f = function(a){

return a;

};

f(10);**// Output: 10**

Through arrow functions you can write the similar code as follows:  
f = a => a; f(10); **// Output: 10**f = (a) => { return a }; f(10); **//same output, well-structured code  
  
Note: function body comes on the right of fat arrow.**

**\*\*Arrow functions with multiple arguments:**

let add = (a,b) => { return a + b }

add(9,10); // **Output: 19**  
----------------------------------------------------------------------------------------------------------------------------------

Que: What is method chaining / function chaining in JavaScript?  
**Chaining Methods refers to repeatedly calling one method after another in one continuous line of code.**

## Suppose in jQuery if you are writing code like this:

## $(" myDiv").removeClass(‘off’).addClass("on"); In above jQuery code, we are removing class off and adding class ‘on’ in a one single line by using dot notation, this is called as function chaining. ------------------------------------------------

**Que:** What is the difference between a method and a function in JavaScript?

Functions and methods both are functions in JavaScript. A method is just a function which is a property of an object.

**function** fun(param1, param2){

// some code...

}

\*\*The above code is an example of function.

**var** obj ={

name :"John snow",

work :**function**(paramA, paramB){

// some code

}

}

\*\*In the above code, work is a method.

\*\*Keep below things in mind:

So inshort All methods are functions But all functions are not methods.

**Q: what would be the output below code will produce?**  
  
function f(){

console.log(this)  
  
 }

var obj = {

abc: ‘hi all’,

work: function(){

console.log(this);

console.log(this.abc);

}

}

f();

obj.work();

Output:  
Window  // ‘**this’** points to **window** object

**{abc: "hi all", work: ƒ}** // ‘**this’** points to object **obj**, it shows all the object fields with values

hi all // method can access all the fields of its object using **this** keyword  
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Q: Explain Spread operators and rest parameters in JavaScript?**  
Spread operators**: Spread operator is used to create an array from your passed arguments. For ex. If you are passing four arguments like 10, 20, 30, 40; and you want those arguments stored in a single array then you can do this through spread operator.**

**Spread operator syntax: …n**

**/\* Just 3 dots followed by array name. We can give any name to array \*/**

**Ex:  
var myarr = function(...n){ // function myarr(...n){ } also work fine**

**console.log(n);**

**};**

**myarr(11,22,55,44,88); // we are passing 5 arguments here..**

**Output: (5) [11, 22, 55, 44, 88] /\* Array length is 5 \*/**

Merging 2 arrays through spread operator: **Another use of spread operator is we can merge two different arrays. This means you can add all elements of one array into another array in a simple way. And you can add those elements anywhere in your existing array, like at the beginning, in the middle or at the end of an array. This is very useful feature provided by spread operator.**

**EX:  
var arr2 = [7, 8, 9, 10];**

**var arr1 = [1, 2, 3, 4, 5, 6, ...arr2];**

**console.log(arr1);**

**Output: (10) [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]**

**Description: We have added arr2 array inside arr1 array through spread operator at the last index. This means your new elements will add from the last index of your existing arr1 array.**

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

**Que:Write a program to merge two arrays into one single array and then sort it’s elements in ascending order.**

**var arr1 = [111,22,33,4,8];**

**var arr2 = [222,2,3,23,18];**

**var arr3 = [...arr1, ...arr2];**

**arr3.sort(function(a,b) { return a-b} );**

**Output: (10) [2, 3, 4, 8, 18, 22, 23, 33, 111, 222]**

**2) Rest parameters:**

**Rest parameters also creates an array from the arguments like spread operator, only the difference is, in spread operator every argument is included in your new array but in rest parameter, you are not including every argument in your new array.**

For Ex:

var myarr = function(a,b,c,...arr){// array name is arr here, first 3 argument will not be part of your new array, array will start from fourth element

console.log(arr);

};

myarr(11,22,55,44,88,110);

Output: (3) [44, 88, 110] // array length is 3 with their respective values.

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

Q: Explain constants in JavaScript:   
In [programming](http://www.webopedia.com/TERM/P/program.html), a constant is a value that never changes.

Often in programming language, you need a value which you don’t want to change once you define it, so constant help us to achieve this goal.

#### ECMAScript 6:

ECMAScript 6 added 2 keywords to overcome disadvantages of using var keyword. They are let and const,

const keyword allows us to set constants in JavaScript.

Ex:

const PI = 3.14; // PI is constant

console.log(PI);

Output: 3.14

Note: If you try to change value of PI, it will throw error as below:

“TypeError: Assignment to constant variable.”

This is because you cannot change value of ‘const’ after setting for the first time.

## Q: What is prototype chain in JavaScript?

Ex:  
const A = function(){}; // empty function expression

A.prototype.print = function(){ return "I am print method of A prototype ";}

const B = function(){};

B.prototype = Object.create(A.prototype); // “B.prototype = new A”; is also acceptable

B.prototype.print = function(){

return "I am print method of B prototype"

};

const C = function(){};

C.prototype = Object.create(B.prototype); //

//C.\_\_proto\_\_ = B.\_\_proto\_\_ [we can use “**objectName.\_\_proto\_\_”** to inherit //members from another object’s prototype]

C.prototype.print= function(){

return "I am print method of C prototype "

};

const Aobj = new A();

const Bobj = new B();

const Cobj = new C();

console.log(Aobj.print());

console.log(Bobj.print());

console.log(Cobj.print());

Output:

I am print method of A prototype

I am print method of B prototype

I am print method of C prototype

Description: In above example, every function prototype has its own print method. Function c inheriting B function and function B Inheriting A function through prototype inheritance. If we remove print method of C then C will look for print method one level up, means in prototype of B Function and if it finds there then it will display B’s print method. And if it is not exists in B function also then it will look again one level up, means in function A prototype. This process is called as Prototype chaining.   
  
This means each link in its prototype chain is traversed until the attribute is found; is called as Prototype chaining.

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

What will be the output for below code:

**Ex:**var foo = 'outside';

function logit(){

console.log(foo);

var foo = 'inside';

}

**logit();**

Output: undefined. It’s because hoisting. Within function scope, we are assigning value to ‘foo’ after console.log, if we are assigning value, somewhere inside the function; the value of same variable will be undefined at top **of the function**. It is undefined **until we reach the statement** where we are assigning the value.

## Note: If we remove foo declaration var foo = “inside” which is written after console.log then the global variable foo which is at very first line will display in console.log.

NUANCE QUESTIONS STARTS:

1. What are Boolean in JavaScript?

A JavaScript Boolean represents one of two values: **true** or **false**.

Very often, in programming, we need a data-type that can only have one of two values, like

* YES / NO
* ON / OFF
* TRUE / FALSE

For this, JavaScript has a **Boolean** data type. It can only take the values **true** or **false**.

You can use the Boolean() function to find out if an expression (or a variable) is true:

Boolean(10 > 9)        // returns true

## Never forget, Everything Without a "Real" Value is False

The Boolean value of **0** (zero) is **false: Ex: var x = 0;Boolean(x);**

The Boolean value of **-0** (minus zero) is **false. Ex:Boolean(-0);**

**Note: “0” & “-0” [zero and minus zero within quotes] are strings with length 1 and 2 respectively, so they both R truthy. Means, Boolean (‘0’) and Boolean (‘-0’) is true.**

The Boolean value of **""**(empty string) is **false. Ex: var x = "";Boolean(x);**

The Boolean value of **undefined** is **false. Ex: var x;Boolean(x);**

The Boolean value of **null** is **false. Ex: var x = null;Boolean(x);**

The Boolean value of **false** is **false**: **Ex: var x = false; Boolean(x);**

**Remember Boolean('false') returns true because ‘false’ is a string, it is within quotes.**

The Boolean value of **NaN** is **false. Ex: var x = 10 / "H";Boolean(x);**

Remember, Your variable's type depends on the assigned value in JavaScript.

var a = "true"; // data type is string here

a = true; //now your variable is a boolean

**Another way to Set Boolean values to your variables**

var myvar1=new Boolean(false);

var myvar2=new Boolean(true);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Q: What are the local JavaScript variables?**

Variables declared within a JavaScript function, become LOCAL to the function. Local variables have local scope as they can only be accessed within the function. Local variables are created when a function starts, and deleted when the function execution ends.

For Ex:

Ex: Check carName variable

// code here cannot use carName  
  
function myFunction() {  
    var carName = "Volvo";  
  
    // code here can use carName  
  
}

// code here cannot use carName

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

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

1. \*\*\*What is callback function in JavaScript?

A **callback function** is a **function** that is passed to another **function** as a parameter, and **callback function** is executed inside another Function. A **callback function** is also known as a **higher-order function.** Instead of immediately returning some result like most functions do, functions that use callbacks often takes some time to produce the result. setTimeout method or addEventListener method are perfect example of callback function. Both takes function as a parameter and execute that callback function to produce the result.

For ex:  
setTimeout(function(){ console.log(‘show this msg after 2 seconds’) }, 2000)

Take a look at below jQuery code:

$("#btn\_1").click(function() {

alert("Btn 1 Clicked");

});

In the above example, we are passing function as a parameter to click method. And the click method will execute that callback function to produce the result.

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

1. How to check whether your variable is an array or not?

There are 2 ways of checking if a variable is an array or not, As follows:

I) variableName.constructor===Array

OR

II) Array.isArray(variableName)

Ex: var myArr = [12,13,14]

myArr.constructor === Array // Array.isArray(myArr)

Output: true

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

1. What is prompt box in JavaScript?

The **prompt**() method displays a **dialog box that prompts the visitor for input**. A **prompt box** is often used if you want the user to input a value before entering a page. Note: When a **prompt box** pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

7) How can we change style of an element through JavaScript?

The syntax is: ElementName.Style.Property = “value”

Suppose we are changing style for h1 element. Your syntax would be:

**h1.style.color = "red";**

**h1.style.fontSize = "15px";**

**h1.style.backgroundColor = "#FFFFFF";**

1. How can we change Class of an element- through JavaScript?
2. To remove all existing classes and attach a new class to the element:

document.getElementById("MyElement").className = "MyClass";

OR

document.getElementById("MyElement").setAttribute("class","MyClass");

Above lines will remove all previous classes and will add new class to your element.

**Note: You cannot change class Name unless you trigger any event in JavaScript.**

Ex:

**<body>**

**<h1 id='MyElement' class="old1 old2 old3">Element here</h1>**

**<button onclick="clickBtn()">Click here</button>**

**<script>**

**function clickBtn(){**

**document.getElementById("MyElement").className = "MyClass";**

**}**

**</script>**

**</body>**

II) Add class to your element, without removing/affecting existing classes: -  
**you need to use plus operator along-with equal to sign and a space followed by new Class Name**:

document.getElementById("MyElement").className += " MyClass";

// note, before class-Name space needs to be there

**Que: How can you check whether your element has a particular class has attached or not?**

Ans: To check whether your element has a particular class attached or not, we have two ways:

* 1. **Use element.classList.contains(‘Your className here’)**

OR

* 1. **Use element.className === ‘Your className here’**

1) element.classList.contains(‘className’)  
var testElement = document.getElementById(top');

console.log(testElement.classList.contains('answer'));

Output will be true if you have answer class attached for the same element. If JavaScript finds class is not attached then it will return false.

2) Use if(element.className === “className”){}

For Ex: below we are toggling two classes i.e.active and inactive  
let elm = document.getElementById('MyElement');

if(elm.className === 'active'){

elm.className = 'inactive';

} else {

elm.className = 'active';

}

Note: above code will set only one class at a time to your element, this means if you have more classes, those classes will be removed from the DOM.

1. What is called Variable typing/ dynamic typing in JavaScript?

**Your variable's type depends on the assigned value in JavaScript.**

For Ex:

var x; typeof x; // Output: undefined"

x = "**2"; typeof x; // Output: "string"**

**x = 2; ty**peof x; // Output: "number"

x = true; typeof x //Output : "boolean"

Means whenever we change value of a variable, its data-type gets changed automatically. This process is called as Variable typing in JavaScript.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

10) How to detect the operating system on a client machine?

To **detect the operating system on the client machine**, we should use **navigator.appVersion**

and To find Browser Name use-> **navigator.userAgent**;

# [How do I empty an array in JavaScript?](https://stackoverflow.com/questions/1232040/how-do-i-empty-an-array-in-javascript)

There are 2 ways to make your array empty:

1. **First way,** just use pair of empty square brackets,

Assume you have an array having name A

A =[1,2,3,4];

Now next to array-Name, just add empty opening and closing square brackets

**A =[]; // array is empty now…**

This code will set the variable A to a new empty array.This is perfect if you don't have **references to the original array**A anywhere else because this actually creates a brand new (empty) array. You should be careful with this method because if you have referenced this array from another variable or property, the original array will remain unchanged.

Please find below example to understand where we have emptied arr1 but arr2 still contains the similar values..

var arr1 =['a','b','c','d','e','f']; // string array

var arr2 = arr1;// Reference arr1 by another variable

arr1 =[];

console.log(arr2);

**// Output ['a','b','c','d','e','f']**

# Setting array Length to 0:

A =[1,2,3,4];

**A.length =0**

Length zero will remove all your array elements...

12) What does void mean in JavaScript?

The void operator evaluates the given expression and then returns **undefined**.

## JavaScript URIs [Universal Resource Identifier]

URI stands for Universal Resource Identifier.In often cases we use void as a part of JavaScript: URI.Javascript:URI is useful in many cases. Check below examples:

## 1) If you don’t want to navigate anywhere through your href, then you can use javascript: URI as follows:

**1)<a href="javascript:void(0);"> Click here to do nothing</a> // undefined href value…**

It means it’ll do nothing. It’s an attempt to have the link not ‘navigate’ anywhere.

**2) We can use javascript:URI to change style of particular element.**

**<a href="javascript:void(document.body.style.backgroundColor='green');">**

**Click here for green background**

**</a>**

3) We can display alert messages also through javascript:URI as below:

**Ex:<a href="javascript:void(alert('Hey this is warning!!!'))">Click me!</a>**

**14) What are the escape characters in JavaScript?**

**5 commonly used escape characters are as follows:**

* **\'** Single quote // to insert a single quote
* **\"** Double quote. // to insert a double quote
* **\\** Backslash// to display a single backslash
* **\n** new line // to insert your characters on next line
* **\t** tab // to insert a tab

**Below are not so commonly used….**

* **\r** carriage return.
* **\b**backspace.
* **\f** form feed.

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

15) What is the difference between continue and break statement in JavaScript?

**continue** skips the current executing iteration within the loop and MOVES TO the next iteration whereas **break** MOVES OUT of the loop. Means break takes you out of the loop and gives control to the next statement which is immediately after the loop, whereas continue takes you to the next iteration in the same loop.

Ex:  
break example:  
**for(i=0;i<10;i++)**

**{**

**if (i==4)**

**{**

**break;**

**}**

**console.log(i);**

**}**

Output: 0 1 2 3

for(i=0; i<10; i++)

{

if (i==4)

{

continue;

}

Console.log(i);

}

Output: 0 1 2 3 5 6 7 8 9

Nuance Ques. Ends

# JavaScript split() Method

The split() method is used to split a string into an array of substrings, and returns the new array.

**Tip:** If an empty string ("") is used as the separator, the string is split between each character. And if anything other than empty string is provided then split look for that separator character and then it split string from that character.

**Note:** The split() method does not change the original string.

Ex:

**var str = "How are you doing today?";**

**var res = str.split(" ");**//space as a split character, this means whenever it finds space, it will create a new array element

**console.log(res)**

Output: (5) ["How", "are", "you", "doing", "today?"]  
  
  
Ex. 2)\*\*   
var str = "How are you, doing today?";

var res = str.split(",");// whenever comma find, split string from that comma

console.log(res);  
  
Output: (2) ["How are you", " doing today?"]  
Since one comma is in string, split will convert our string into two array elements. One element contains string which are before the comma, and another element is those characters which are after comma.  
  
  
Ex. 3)  
var str = "How are you, doing today, amol?";

var res = str.split(",");// split string into an array, when comma is there

res = res.join("+") // array elements will be concatenated with plus sign first, and then converted to string

console.log(res);

Output: How are you+ doing today+ amol?

\*\*Cybage Que:  
**Que: How can we reverse string in javascript?**Ans:  
var str = "amol"

str = str.split("").reverse().join("");

console.log(str);

Output: loma

\*\*\*\*split method is used to convert string into an array and join method is used to convert array into string..

**Que: Math.floor():**

Ans: The Math.floor() function returns the largest integer less than or equal to a given number.

1) Math.floor(16.99)

Output: 16;

2) Math.floor(16.1)

16

**3) Math.floor(15.00)**

**15**

**Que: Math.sqrt():**

Ans: Math.sqrt() return the square root of a number:

Math.sqrt(9); // Output: 3

Math.sqrt(49); // Output: 7

Math.sqrt(11) // Output:3.3166247903554

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

## Timing Events:

In JavaScript, we can decide to execute a function after a certain delay, not right now. That can be done by timing events. Here we specify waiting time in milliseconds in the form of second parameter and once that waiting time is over, we produce the result.

There are two methods for timing events:

* **setTimeout(function, milliseconds)**Executes a function, after waiting a specified number of milliseconds
* **setInterval(function, milliseconds)**: Same as setTimeout(), but repeats the execution of the function continuously.

The setTimeout() MethodThe **window.** setTimeout **()** method can be written without the window prefix.

Ex: Click a button. Wait 3 seconds, and the page will alert "Hello":

<button onclick="setTimeout(myFunction, 3000)">Try it</button>  
  
<script>  
function myFunction() {  
    alert('Hello');  
}  
</script>

## How to Stop the Execution?

The clearTimeout() method stops the execution of the function specified in setTimeout().

window.clearTimeout(timeoutVariable)

The **window.clearTimeout()** method can be written without the window prefix.

The clearTimeout() method uses the variable returned from setTimeout():

myVar = setTimeout(*function*,*milliseconds*);  
clearTimeout(myVar);

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

## The setInterval() Method

The **window.setInterval()** method can be written without the window prefix.

This example executes a function called "myTimer" once every second (like a digital watch).

Ex:  
var myVar = setInterval(myTimer, 1000);

function myTimer() {

var d = new Date();

console.log(d)

}

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

16) How can you create an Object in JavaScript?

There are 3 ways through which we can create objects in javascript –

var emp ={

name:"Zara",

age:10

};

OR

var emp ={}; // this is empty object..

OR

var emp =new Object(); // this is also empty object..

17) What is JavaScript?

JavaScript is a lightweight, interpreted programming language with object-oriented capabilities that allows you to build interactivity into your static HTML pages.

19) What are the advantages of using JavaScript?

Following are the advantages of using JavaScript −

* **Less server interaction −** **you can validate user input before sending the input to the server.** This saves server traffic, which means less load on your server.
* **Immediate feedback to the visitors –** visitors don't have to wait for a page reload to see if they have forgotten to enter something.
* **Increased interactivity −** you can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
* **Richer interfaces −** You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors

**20) What are disadvantages of using JavaScript?**

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features −

* Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
* JavaScript doesn't have any multithreading or multi-process capabilities.

21) Is JavaScript a case-sensitive language?

Yes! JavaScript is a case-sensitive language. This means that language keywords, variables, function names, and any other identifiers needs to be consistent.

22) How can you read and write properties of an Object in JavaScript?

You can read and write properties of an object using the dot notation as follows:

// Setting [writing] object properties

emp.name ="Daisy"// <== Daisy

emp.age =20// <== 20

// Getting object properties

emp.name // ==> Zara

emp.age // ==> 10

23) How can you create an Array in JavaScript?

You can define arrays using the **array literal** as follows:

var x =[]; // this is empty array

var y =[12,3,4,5]; // array with 5 elements

24) How to **read elements of an array** in JavaScript?

An array has a **length** property that is useful for iteration. We can read elements of an array as follows −

var x =[1,2,3,4,5];

for(var i =0; i < x.length; i++){

console.log(x[i]);  
}

25) What is a named function in JavaScript? How to define a named function?

A named function has a name when it is defined. A named function can be defined using **function** keyword as follows −

function named(){

// do some stuff here

}

26) How many types of functions JavaScript supports?

A function in JavaScript can be either **named or anonymous.**

27) How to define an anonymous function?

An anonymous function can be defined in similar way as a normal function but it would not have any name.

28) Can you assign an anonymous function to a variable?

Yes! An anonymous function can be assigned to a variable.

29) Can you pass an anonymous function as an argument to another function?

Yes! An anonymous function can be passed as an argument to another function.

30) How can you get the **type of arguments** passed to a function?

Using typeof operator, we can get the type of arguments passed to a function. For example:

functiongetArgType(x){

console.log(typeof x, arguments.length);

}

getArgType();//==> "undefined", 0

getArgType(1);//==> "number", 1

getArgType("1","2","3");//==> "string", 3

getArgType(1,"2","3");//==> "number", 3 [first argument is number]

getArgType(true); //==> "Boolean", 1

Check below example:

arr1 = [10,20,30];

arr2 = [11,22,33]

function getArgType(x){

console.log(typeof x, arguments.length);

}

getArgType(arr1,arr2); // passing two arrays

Output: object 2

\*\*Datatype of array is object, passing two arrays so arguments.length is 2..

Ex: passing function as an argument:

function f(){};

function getArgType(x){

console.log(typeof x, arguments.length);

}

getArgType(f);

**Output: function 1**

\*\*Remember typeof function(){} returns function. So if you are passing function as an argument then its typeof will return function.

33) What is the purpose of 'this' operator in JavaScript?

JavaScript’s **this** always refers to the **current context.**

34) What are the valid scopes of a variable in JavaScript?

In JavaScript there are two types of scope:

* **Local scope / Function scope −** A local variable has local scope/ function scope which means it is accessible only within a function where it is declared. Function parameters are always local to that function. We need to prefix “**var”** when declaring the variable in function.
* **Global scope −** A global variable has global scope which means it is accessible everywhere in your JavaScript code. In HTML, the global scope is the window object. All global variables belong to the window object.

\*\*\*Remember, if you declare variable without var keyword in function, it will automatically become a GLOBAL variable.

This code example will declare a global variable **carName**, even if the value is assigned inside a function.

myFunction();  
  
// code here can use carName   
console.log(carName) // \*\*\*output “volvo”  
  
function myFunction() {  
  
if(typeof carName =='undefined') { // \*\*\*carName is undefined here

console.log(‘**in function:** undefined here');

}  
  
carName = "Volvo";  
console.log(“**in function:** ” + carName) // output “volvo”  
  
}

Output:  
in function: undefined here

in function: Volvo

Volvo

**35) Which type of variable among global and local, takes precedence over other if names are same?**

A local variable takes precedence over a global variable with the same name.

36) Give an example of closure?

A closure is an inner function that has access to the variables in the outer (enclosing) function’s scope chain. Specifically, the closure has access to variables in three scopes: (1) variable defined in its own scope, (2) variables defined in the outer function, and (3) global variables.   
\*Also remember closer can access parameters of outer function as well.

Following example shows how the variable counter is visible within the create, increment, and print functions, but not outside of them −

**function create() {**

**var counter = 0; // this line will execute only once…**

**return {**

**/\* create function returns an object; since JavaScript ignore all the statements written after return statement, curly brace needs to be on same line to indicate we are returning object otherwise JavaScript will assume we are returning nothing here.. \*/**

**increment: function() { // this is closure**

**counter++;**

**},  
decrement: function() { // this is closure**

**counter--;**

**},**

**print: function() { // this is also closure**

**console.log(counter); // two functions we are using in return**

**}**

**}**

**}**

**var c = create(); /\* since create function returns an object and within that object we have necessary method, we will access them using dot notation. \*/**

**c.increment(); // counter =1 here; c["increment"](); this will also work**

**c.increment(); // counter =2;**

**c.increment(); // counter =3;**

**c.increment(); // counter =4;**

**c.increment(); // counter =5;**

**c.decrement(); // counter =4;**

**c.decrement(); // counter =3;**

**c.print();**

Output: 3

39) Which built-in method returns the **character at the specified index?**

charAt() method returns the character at the specified index.

40) Which built-in method combines the text of two strings and returns a new string?

concat() method combines the text of two strings. and returns a new string with all the new characters..

Syntax: string1.concat(string2);

**var str = "amol ";**

**var str2 = "Sheetal and Gauri";**

**var newString = str.concat(str2); //similar as** var newString = str + str2

**console.log(newString)**

Output: amol Sheetal and Gauri

41)\*\*Which built-in method calls a function for each element in the array?

Ans: Javascript array **forEach()** method calls a function for each element in the array.   
  
forEach syntax: arrayName.forEach(callback function);

**Ex:**

var numbers = [65, 44, 12, 4];  
  
function displayArray() {

numbers.forEach(function(i,idx) { console.log(i + “ ” + idx )})

}  
displayArray();  
  
Output: 65, 44, 12, 4   
  
Note: first parameter of callback holds the value of current element and second parameter hold the index number of an element. Second parameter is optional. We can omit it.

42) Which built-in method returns the index within the calling String object of the first occurrence of the specified value?

indexOf() method returns the index within the calling String object of the first occurrence of the specified value, or −1 if not found.

43) Which built-in property returns the length of the string?

length property returns the length of the string.

44) Which built-in method removes the last element from an array and returns the new array?

pop() method removes the last element from an array and returns the new array. Remember, pop method reduces your array length.

45) Which built-in method adds one or more elements to the end of an array and returns the new length of the array?

push() method adds one or more elements to the end of an array and returns the new length of the array. Remember, push method increase your array length.

46) Which built-in method reverses the order of the elements of an array?

**reverse()** method reverses the order of the elements of an array −− the first becomes the last, and the last becomes the first. And remember reverse method does not work with string values. If we try to do so, it gives error.

47) Which built-in method sorts the elements of an array?

sort() method sorts the elements of an array.

But the issue is sort() treats array elements as strings. Means ‘**a**’ comes before ‘**b**’ and b comes before C. so in similar way sort() shows 11 before 2, because 1 which is at upper left position in 11 takes precedence in sorting as compare to 2.

**For Ex: var a = [21,1,3,84,5,57,68];**

**var newA = a.sort();**

**console.log(newA);**

**Output: [1, 21, 3, 5, 57, 68, 84]**

## So to overcome this issue, we have 1 simple solution:

**1)use callback function in sort method and return a – b;**

**For Ex: var a = [21,1,3,84,5,57,68];**

**var newA = a.sort(function(a,b){return a- b});**

**console.log(newA);**

**Output: [1, 3, 5, 21, 57, 68, 84]**

**And if you want array to be displayed in descending order, two simple solutions are there: 1) return b – a in callback function of sort method or use reverse method after ascending the array elements  
  
i) return b - a  
var a = [21,1,3,84,5,57,68];**

**var newA = a.sort(function(a,b){return b - a});**

**console.log(newA);**

**ii) Use reverse method**

**var a = [21,1,3,84,5,57,68];**

**var newA = a.sort(function(a,b){return a- b}).reverse();**

**console.log(newA);**

**Output: (7) [84, 68, 57, 21, 5, 3, 1]**

48) Which built-in method returns the characters in a string beginning at the specified location?

**substr()** method returns substring from your original string. It takes 2 parameters, first parameter is starting index and second parameter is total no. of characters needs to be extracted.

Syntax: substr(start index, total no. of chars);

**var str = "amol";**

**var str2 = " Sheetal and Gauri";**

**var newString = str.concat(str2);**

**console.log(newString.substr(5,10)) // start at 5th index and total 10 characters needs to be extract from index 5..**

**Output:Sheetal an**

# JavaScript substring() Method

**substring()** method returns a substring from your original string. The difference between substr method and substring method is second parameter. In substr method, second parameter decide how many characters needs to be extracted from actual string whereas in substring method second parameter tells that substring needs to be end before the given index.

Syntax: substring(start index, end before this index);

Ex:

Extract characters from a string:

var str = "amol";

var str2 = " Sheetal and Gauri";

var newString = str.concat(str2);

console.log(newString.substring(5,12)) // start at 5th index and end before 12 index

Output**: Sheetal**

If "start" is greater than "end", this method will swap the two arguments, meaning str.substring(1, 4) == str.substring(4, 1).. Both outputs same.

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

49) Which built-in method returns the calling string value converted to lower case?

**toLowerCase**() method returns the calling string value converted to lower case.

**var str = "AMoL";**

**console.log(str.toLowerCase());**

**output: amol**

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

50) Which built-in method returns the calling string value converted to upper case?

toUpperCase() method returns the calling string value converted to upper case.

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

51) Which built-in method returns the **string representation of the number's value?**

**toString()** method returns the string representation of the number's value. This means toString() method converts your number into string.

**52) What are the variable naming conventions in JavaScript?**

While naming your variables in JavaScript, keep following rules in mind.

1. **Avoid reserved keywords:** You should not use any of the JavaScript reserved keyword as variable name. For example, **break, continue, var**, **Boolean** etc. names are not valid.
2. **Variable names should not start with a numeral:** JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or the underscore character. For example, 123test is an invalid variable name but **\_123test** is a valid one.
3. **Variable Names are case-sensitive:** JavaScript variable names are case sensitive. For example, Name and name are two different variables.

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

53) How typeof operator works?

The typeof is a unary operator that is placed before its single operand, which can be of any type. **Its value is a string indicating the data type of the operand.**

54) What typeof returns for a null value?

It returns "object".

59) How to redirect to a new url using JavaScript?

To redirect your site-visitors to a new page, you just need to add window.location as follows

60) How can you give print for your current webpage using JavaScript?

OR

Can we achieve Ctrl + p effect with JavaScript?

**Ans: window.print()** will print the current web page when executed…with the help of this, you can print your page or even you can save your page in PDF format.

<h1 onClick="window.print()">Click here to take print of this page</h1>

63) How to handle exceptions in JavaScript?

**JavaScript uses try, catch, finally blocks and throw operator to handle exceptions.**

You can catch programmer-generated and runtime errors, but you cannot catch JavaScript syntax errors.

The **throw** statement allows you to create a custom error. Whenever throw executes it goes first within catch block.

## Exception handling Syntax:

try {  
    Block of code to try}  
catch(err) {  
    Block of code to handle errors}   
finally {  
    No matter your code is throwing exception or not, finally block will execute all the time..}

**Exception handling example:**  
<!DOCTYPE html>

<html>

<body>

<p>Please input a number between 5 and 10:</p>

<input type="text" id="demo">

<button type="button" onclick="myFunction()">Test Input</button>

<p id="message"></p>

<script>

function myFunction() {

var message, x;

message = document.getElementById("message");

message.innerHTML = "";

x = document.getElementById("demo").value;

try {

if(x == "") throw "is empty";

if(isNaN(x)) throw "is not a number";

if(x > 10) throw "is too high";

if(x < 5) throw "is too low";

}

catch(err) {

message.innerHTML = "Input " + err;

}

finally {

document.getElementById("demo").value = "";

}

}

</script>

</body>

</html>

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

Thatjsdude.com

## 65. **Question:**What are the differences between null and undefined?

**Answer:** JavaScript has two distinct values for nothing, null and undefined.

#### Undefined:

undefined means, value of the variable is not defined.

If we are just declaring the variable but not defining any value to it, then it will return it’s typeof undefined.

Ex:

var x; typeof x;

output: undefined

And even if we are trying to get typeof for undeclared variable, it also return undefined.

Ex:

typeof xyz  
output: undefined  
  
typeof undefined is "undefined". Remember, undefined is not a constant or a keyword. undefined is a type with exactly one value: undefined

**8 Ways to get Undefined:**

* A declared variable without assigning any value to it.
* Implicit returns of functions due to missing return statements.
* return statements that do not explicitly return anything.
* Lookups of non-existent properties in an object.
* Function parameters that have not passed.
* Anything that has been set to the value of undefined.
* Any expression in the form of **void**(expression)
* The value of the global variable undefined

#### null

null means empty or non-existent value which is used by programmers to indicate “no value”. null is a primitive value and you can assign null to any variable. null is not an object, it is a primitive value. For example, you cannot add properties to it. Sometimes people wrongly assume that it is an object, because typeof null returns "object".

## 66) What are the differences between == and ===?

**Answer:**The simplest way of saying that, == will not check types and === will check whether both sides are of same type.

=== compares the types and values. Hence, if both sides are not same type, answer is always false.

Ex: ==

2 == "2"

Output: true

Ex: ===

2 === "2"

Output: false

## 68. Truthy isn't Equal to true

**Question:**As [] is true, []==true should also be true. right?

**Answer:** You are right about first part, [], empty array is an object and object is always truthy. Hence, if you use if([]){console.log('it’s true')} you will see the log.

In short,

Boolean([])

Output: true

However, special case about == (double equal) is that it will do some implicit coercion.

Since left and right side of the equality are two different types, JavaScript can't compare them directly. Right side is of Boolean type and left is of object type.

Hence, JavaScript will convert them to compare. first right side of the equality will be cooereced to a number and number of true would be 1.

After that, JavaScript implementation will try to convert [] by usingtoPrimitive (of JavaScript implementation). since [].valueOf is not primitive will use toString and will get ""

Now you are comparing "" == 1 and still left and right is not same type. Hence left side will be converted again to a number and empty string will be 0.

Finally, they are of same type, you are comparing 0 === 1 which will be false.

In short,

Boolean([] == true)

Output: false

69.How could you write a **method on instance of a Date which will give you next day**?

**Solution:**

**Date.prototype.nextDay = function(){ // Date.prototype is mandatory**

**var currentDate = this.getDate();**

**return new Date(this.setDate(currentDate + 1));**

**// set tomorrows date now through setDate method**

**}**

**var dateObj = new Date(); // dateObj is instance of Date method**

**dateObj.nextDay(); // we are calling nextDay function now**

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

70.If you want to use an arbitrary object as value of this, how will you do that?

**Answer:** There are at least three different ways to doing this by using bind, call and apply. For example, I have a method named deductMontlyFee in the object monica and by default value of this would be monica inside the method.

Solution is as follows:

var monica = {

name: 'Monica Geller',

total: 400,

deductMonthlyFee: function(fee){

this.total = this.total - fee;

return this.name + ' remaining balance is '+ this.total;

}

}

var rachel = {name: 'Rachel Green', total: 1500};

var rachelFeeDeductor = monica.deductMonthlyFee.bind(rachel, 200);

console.log(rachelFeeDeductor());

//"Rachel Green remaining balance is 1300"

console.log(rachelFeeDeductor());

//"Rachel Green remaining balance is 1100"

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

## 71.arguments and call

**Question:** Write a simple function to check whether 2 is passed as function parameter or not?

**Answer:**First convert arguments to an array by using **Array.prototype.slice.call** method and pass arguments as an object. After that simply use indexOf method.

Solution:

function isTwoPassed(){

var args = Array.prototype.slice.call(arguments); // array created for arguments

// We are creating new array here by using **Array.prototype.slice.call.** We are calling isTwoPassed thrice, this means we are going to create 3 array here…If you want to see those arrays, type **console.log(args);**

console.log(args.indexOf(2) != -1)

// Remember indexOf() method works with string as well as with array. When you are using indexOf method with string value, the searching value needs to be in quotes whereas if you are searching in array of integers, then never use quotes around the integer value.

}

isTwoPassed(1,4) //false

isTwoPassed(5,3,1,2) //true

isTwoPassed(5,3,1,22) //false

72.Take 2 arrays. One contains keys and another contains values for the same keys. Write a code through which you can insert those keys and their respective values in one single object.  
Solution:

var \_keyarray = ["key1", "key2" ,"key3"];

var \_valuearray = ["1", "2" ,"3"];

var obj = {};

for(i=0; i<\_keyarray.length;i++){

obj[\_keyarray[i]] = \_valuearray[i];

}

console.log(obj)

Output: {key1: "1", key2: "2", key3: "3"}

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

**73)How could you use Math.max to find the max value in an array?**

**Answer:** Use apply on Math.max and pass the array as apply takes an array of arguments. Since we are not reading anything from this, we can simply pass null as first parameter.

Solution:

**arr = [1,4,55,8,11]**

**function getMax(arr){ // accepting array as an parameter**

**return Math.max.apply(null, arr); // since we are not reading anything from any object, we are using null as first parameter. We can use Math itself instead of null**

**}**

**getMax(arr); // We can send array also, as an argument**  
Output: 55

## \*\*74) this

**Question:** What is this in JavaScript?

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

1. In the global object 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, the object becomes the value of this.**
3. **Inside IIFE (immediate invoking function Expression) “**this” refers to window object.
4. **Inside a** setTimeout **function, the value of this is the** window object**.**
5. **If you use a constructor (by using new keyword) to create an 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 this case this will refer to the button**

# 75) Rapid Fire Questions:

**Question:** What is typeof []

**Answer:** Object. Actually Array is derived from Object. If you want to check array use Array.isArray(arr)

**Question:** What is typeof arguments

**Answer:** Object. **arguments are like array but not an array**. It has length, can access by index but can't push pop, etc.

**Question:** What is 2+true

**Answer:** 3. The plus operator between a number and a boolean or two boolean will convert boolean to number. Hence, true converts to 1 and you get result of 2+1

**Question:** What is true + true

**Answer:** 2. The plus operator between a number and a boolean or two boolean will convert boolean to number. Hence, true converts to 1 and you get result of 1+1

**Question:** What is false + false

**Answer:** 0. The plus operator between two boolean will convert boolean to number so you get result of 0 + 0

**Question**: What is "false" + "false"

**Answer:** falsefalse. [Both operands are strings]

**Question:** What is '6'+9

**Answer:** 69. If one of the operands of the plus (+) operator is string it will convert other number or boolean to string and perform a concatenation. For the same reason, "2"+truewill return "2true"

**Question:** What is the value of 4+3+2+"1"

**Answer:**  91. The addition starts from the left, 4+3 results 7 and 7+2 is 9. So far, the plus operator is performing addition as both the operands are number. After that 9 + "1" where one of the operands is string and plus operator will perform concatenation.

**Question:** What is the value of "1"+2+4

**Answer:** "124". For this one "1" + 2 will produce "12" and "12"+4 will generate "124".

**Question:** What is the value of -'34'+10

**Answer:** -24. minus(-) in front of a string is an unary operator that will convert the string to a number and will make it negative. Hence, -'34' becomes, -34 and then plus (+) will perform simple addition as both the operands are number.

**Question:** What is the value of  -'34' + '10'

**Answer:** "-3410"

**Question:** What is the value of +'dude'

**Answer:** NaN. The plus (+) operator in front of a string is a unary operator that will try to convert the string to number. Here, JavaScript will fail to convert the "dude" to a number and will produce NaN.

**Question:** If you have var y = 1, x = y = typeof x; What is the value of x?

**Answer:** "undefined" [Because x is undefined ]

**Question:** for var a = (2, 3, 5, 65); what is the value of a?

**Answer:** 65. The comma operator evaluates each of its operands (from left to right) and returns the value of the last operand.

**Question:** for var a = (1, 5 - 1) \* 2 what is the value of a?

**Answer:** 8

**Question:** What is the value of !'bang'

**Answer:** false. ! is NOT. If you put ! in front of truthy values, it will return false.

[false + true = false], Using !! (double bang) is a tricky way to check anything truthy or falsy by avoiding implicit type conversion of == comparison.

Note:

**!""** Output: true (false + false = true)

**!!"bang"**Output: true (false + false + true = true)

**!!""**Output: false (false + false + false = false)

**Question:** What is the value of parseFloat('12.3.4')

**Answer:** 12.3

**Question:** What is the value of Math.max([2,3,4,5]);

**Answer:** NaN (Because syntax is incorrect, **Math.max(2,3,4,5);** this is correct syntax)

**Question:** 3 instanceof Number

**Answer:** false [if you are creating instance with new keyword, only in that case instanceof returns true]  
  
**Question:**  What will be the output?

var three = Number(3)

three instanceof Number

Output: false

**Question:**  What will be the output for below code?

var three = new Number(3)

three instanceof Number

**Answer:** true [if you are creating instance with new keyword, only in that case instanceof returns true]

**Question:**null == undefined

**Answer:** true

**Question:**null === undefined

**Answer:** false [typeof null is object and typeof undefined is undefined]

**Question:**What is the boolean value of !!function(){};

**Answer:** true

**Question:** What is the value of typeof bar

**Answer:** "undefined"

**Question:** What is the value of typeof null

**Answer:** "object"

**Question:** If var a = 2, b =3 what would be value of a && b

**Answer:** 3[if a is truthy then return b and if a is falsey then return a, this is how logical && works, here a is truthy]

**Question:** What is -5%2

**Answer:**-1. the result of remainder always get the symbol of first operand

**Question:**42..toString()

**Anwser:** "42" // 42 is string now….Earlier it was a number

If you want to convert the number to a string you cannot write like number.toString() in JavaScript. To achieve this follow below 5 syntaxes:

2..toString();// the second point is correctly recognized  
2.toString();// note the space left to the dot  
(2).toString();// 2 is evaluated first

/\* convert number to string without toString method \*/

2 + “” // number alongwith empty string

String(2) // this is also popular way to convert number into string type

**Question:**42.toString();

**Anwser:** Uncaught SyntaxError (You cannot convert a number to a string by using single dot notation)

**Question:** 4.2..toString();

**Anwser:** //SyntaxError: you cannot convert float like this.

**Question:**42 .toString() //note the space at left before the dot

**Anwser:** "42"

**Question:** typeof(NaN)

**Anwser:**"number"

**Question:** 2 in [1,2]

**Anwser:** false. Because "in" returns whether a particular property/index available in the Object. In this case object has index 0 and 1 but don't have 2. Hence you get false.

**Question:** 2,1 in [1,2]

true

|  |  |
| --- | --- |
| 76) | instanceof  The instanceof**operator** tests whether an object is available in its prototype chain or not. Syntax: object instanceof constructor  Checks the current object and returns true if the objectis of the specified object type. |

var color1 = new String("green");

color1 instanceof String; // returns true

var color2 = "coral"; //not using ‘new’ keyword and no type specified

color2 instanceof String; // returns false (color2 is not in String prototype)

One thing worth mentioning is instanceof evaluates to true if the object inherits from the classe's prototype:

var p = new Person("Jon");

p instanceof Person

That is p instanceof Person is true since p inherits from Person.prototype.

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

75) Extract numbers from the array and then print the sum of all the numbers.

Ans:

var arr = [11, "a" , "b", 3, 5, 8, "c", 2, 4];

function arraySum(arr){

var sum = 0;

for(var i=0; i<arr.length; i++){

if(typeof arr[i] == 'number'){

sum = sum + arr[i];

}

}

return sum;

}

arraySum(arr); // array passing

Output: 33

## 76. log prefix

Que: How could you set a prefix before everything you log? for example, if you log('my message') it will log: "[amy] my message"..

**Answer:** Just get the arguments, convert it to an array through **Array.prototype.slice.call**, and unshift whatever prefix you want to set. Finally, use apply to pass the arguments array to console.

**Answer:**

function LogMyName(){

var args = Array.prototype.slice.call(arguments); // we created array here

args.unshift('[Amy]'); // attached prefix **amy** here

console.log.apply(null,args); // since we are not reading anything from any object, we are taking null as a first parameter. Instead of null, console is also acceptable as 1st parameter

}

LogMyName("Hi");

LogMyName("Amol this side", "Location Pune");

LogMyName("May I have your attention please", "I am waiting:)");

**Ouput: [Amy] Hi**

**[Amy] Amol this side Location Pune**

**[Amy] May I have your attention please I am waiting:)**

.

# Check below 2 examples carefully:

Ex:1)

function add(a,b){

console.log(a + b)

}

add(10,20);

Output: 30

Ex:2)

function add(a,b){

return;

console.log(a + b)

}

add(10,20);

Output:**undefined**[console will show nothing here because your return statement stops your function execution, so the next console statement will not execute.We **can use return keyword anywhere in the function: at the last, at the middle or at very first line**. Only the thing is, next statements which comes after return statement in the function body, will not execute at all]

77) What will you see in the console for the following example?

**var a = 11; // global variable**

**function b() { a = 10; return; } // value of a is 10 now…**

**function c() { a = 100; }**

**b();**

**console.log(a);**

**c();**

**console.log(a);**

**Output: 10**

**100**

\*\*Remember when you change your global variable’s value in any function, then on next moment, the same variable moves with new value [Local value which he has taken from previous function]

## When a return statement is find in a function body, the execution of the function is stopped. If specified, a given value is returned to the function caller. If the expression is omitted, undefined is returned instead. The following all return statements break the function execution:

**return;**

**return true;**

**return false;**

**return x;**

**return x + y /3;**

### Automatic Semicolon Insertion: **ASI** Line terminator is not allowed between the return keyword and the expression.

For Ex:  
return

a + b;

### is transformed by ASI into:

return;

a + b; // this statement will not execute..

### Interrupt a function

A function immediately stops at the point where return is called.**We can use return keyword *anywhere* in the function: at the last, at the middle or at very first line**

**function counter() {**

**for (var count = 1; count < 10; count++) {**

**console.log(count + 'A'); // until 5**

**if (count === 5) {**

**return; // we can use return keyword anywhere in the function**

**}**

**console.log(count + 'B'); // until 4**

**}**

**console.log(count + 'C');**

## // never appears because this statement will execute after for loop finish but due to return statement, execution moves out of the main function ….

**}**

**counter();**

// Output:

// 1A

// 1B

// 2A

// 2B

// 3A

// 3B

// 4A

// 4B

// 5A

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

78) What will you see in the console for the following example?

var a = 1;

function b() {

a = 10;

return;

function a() {} // a is empty function

}

b();

console.log(a);

**Output: 1**

#### Above example Explanation:

* function declaration function a(){} is hoisted first and it behaves like var a = function () {};. Hence in local scope variable a is created.
* If you have two variables with same name (one in global another in local), local variable always get precedence over global variable.
* When you set a = 10, you are setting the local variable a , not the global one. Hence, the value of global variable remain same and you get, 1 in the log.
* **Extra:** If you didn’t have a function named as "a", you will see 10 in the log.

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

## 79) Closures Inside Loops

**Question:** You have for loop and within for loop, you have setTimeout function.

What will be the output for below code?

for(var i = 0; i < 10; i++) {

setTimeout(function() {

console.log(i);

}, 10);

}

**Answer:** The above will not output the numbers 0 through 9, but will simply print the number 10 ten times.

**Explanation:** The console log is inside the anonymous function of setTimeout and setTimeout is executed when current call stack is over. So, the loop finishes and before setTimeout get the chance to execute. However, anonymous functions keep a reference to i by creating a closure. Since, the loop is already finished, the value i has been set to 10. When setTimeout use the value of i by reference, it gets the value of i as 10. Hence, you see 10 ten times.

**Solution:**You can fix it by avoiding closure. Just create a IIFE (Immediately Invoked Function Expression), it will create its own scope and you can pass i to the function. In that case i will be a local variable (will not refer to i in the closure) and value of i in every loop will be preserved.

for(var i = 0; i < 10; i++) {

setTimeout((function(i) {

console.log(i);

})(i), 10)

}

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

# The `delete` Operator in JavaScript

The purpose of delete is to delete things. More specifically, it deletes object properties. For example:

**var** multiverse **=** {

earth1**:**"Silver Age",

earth2**:**"Golden Age"

};

**delete** multiverse.earth2;

console.log(multiverse);

***Output: { earth1: "Silver Age" }***

\*\*\*\*The delete operator **cannot delete ordinary variables**

**For Ex:**

**var** alex **=**"Alexander Luthor";

**delete** alex;

console.log(alex);

***// Output: "Alexander Luthor" // delete unable to delete ordi*nary var**

Delete operator:

var anObject = { left : 1, right : 2};

console.log(anObject.left) ; // 1

delete anObject.left; // left property deleted here

console.log(anObject.left); // undefined

console.log("left" in anObject) ; //false, since left is not anObject’s property, it’s deleted already

console.log("right" in anObject) ; // true

Output:

1

undefined

false

true

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

‘delete’ Operator:

Check below example:

var P = { prop: 42 };

var O = Object.create(P); // P is O's prototype.

When you retrieve O.prop, you get the value of prop from O if O has a property with that name (even if its value is undefined), but if O doesn't have the property at all, then the value will be retrieved from P.prop instead.

alert(O.prop); // "42" since O doesn't have its own prop, but P does.

O.prop = undefined;

alert(O.prop); // "undefined" since O has its own prop.

delete O.prop;

alert(O.prop); // "42" since the delete "unmasked" P.prop.

**80) Question:** Look at the code below, I have a property in object and I am creating a new object where I am setting it to a new value. If I delete that property what will i get if I try to access that property?

var myObject = {

price: 20.99,

get\_price : function() {

return this.price;

}

};

var customObject = Object.create(myObject);

customObject.price = 19.99;

delete customObject.price;

console.log(customObject.get\_price());

**Output: 20.99**

**82) Question:** **What is curring**

**OR**

**How would you implement currying for any functions in JavaScript?**

Curring is *partial* invocation of a function. Currying means first few arguments of a function is pre-processed and a function is returned. The returning function can add more arguments to the curried function. It's like if you have given one or two spice to the curry and cooked little bit, still you can add further spice to it.

**Ex 1)**

<script>

function addBase(base){ // this is a base function..

return function(num){

console.log(base + num)

}

}

var addTen = addBase(10); // the value 10 will add up every time ..

addTen(5); //output 15 // we are calling object here…

addTen(80); //Output 90 , calling object here…

addBase(2)(3) // calling base function, another way to call curried function, first argument is passed to curried function & another argument is passed to inner function..

</script>

**Output: 15**

**90**

**5**

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

**Ex. 2) Currying with 2 inner functions:**

function parent(base){

return function(num1){

return function(num2) {

console.log(base + num1 + num2)

}

}

}

parent(2)(3)(4) **Output: 9**

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

83) How to find whether your input number is prime or not?

Ans:

**function Prime\_find(n)**

**{**

**if (n < 2) return false; // 1 is not prime by default so less than 2 here..**

**var q = Math.floor(Math.sqrt(n));**

**for (var i = 2; i <= q; i++)**

**{**

**if (n % i == 0)**

**{**

**return false;**

**}**

**}**

**return true;**

**}**

**Prime\_find(17);**

**Output: true**

84) Fibonacci series with simple way and with recursive function:

**function fibonacci(n){**

**var fibo = [0, 1]; // at index zero value zero we are storing and at index 1, value 1 is stored..**

**if (n <= 2) return 1;**

**for (var i = 2; i <=n; i++ ){**

**fibo[i] = fibo[i-1]+fibo[i-2];**

**}**

**return fibo[n];**

**}**

**fibonacci(12)**

**Output: 144**

**Fibonacci with recursive function:  
function fibonacci(n){**

**if(n<=1)**

**return n;**

**else**

**return fibonacci(n-1) + fibonacci (n-2);**

**}**

**fibonacci(14);**

**Output: 377**

## 86) swap numbers without temp

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

**console.log('before swap: ','a: ', a, 'b: ', b);**

**b = b -a;**

**a = a+ b;**

**b = a-b;**

**console.log('after swap: ','a: ', a, 'b: ', b);**

**}**

**swapNumb(2, 3);**

**Output:**

**before swap: a: 2 b: 3**

**after swap: a: 3 b: 2**

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

**87) Write a function that increase value of a variable by 1, at each second.**

Ans: You will need setInterval() method here

function Counter() {

var num = 0;

setInterval(function() {

num++;

console.log(num);

}, 1000);

}

Counter();

**Note: \*\*\*\*setInterval function is working like closure here..**

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

# Object oriented concepts in JavaScript: Abstraction: Abstraction is a process where you show only “relevant” data and “hide” unnecessary details from the user. Consider your mobile phone, you just need to know what buttons are to be pressed to send a message or make a call, What happens when you press a button, how your messages are sent, how your calls are connected is all abstracted away from the user.

# Encapsulation: Encapsulation is the process of combining data and functions into a single unit called class. In Encapsulation, the data is not accessed directly; it is accessed through the functions present inside the class. In simple words, attributes of the class are kept private, and public getter and setter methods are provided to manipulate these attributes. Thus, encapsulation makes the concept of data hiding possible.

Inheritance: refers to an object being able to inherit methods and properties from a parent object (a class in other OOP languages or a function in JavaScript.)

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

**Toptal.com ques starts**

What is a potential pitfall with using typeof bar === "object" to determine if bar is an object? How can this pitfall be avoided?

\*\***Pitfall**: unexpected or surprising difficulty

**Although typeof bar === "object" *is* a reliable way of checking if bar is an object, the surprising gotcha in JavaScript is that null, functions and arrays are *also* considered as an object!**

For Ex:  
**var bar = []; // var bar = function() {}; // var bar = null;**

**console.log((bar !== null) && (typeof bar === "object") && (bar.constructor !== Array));**

Que: What is the significance, and what are the benefits, of including 'use strict' at the beginning of a JavaScript source file?

Ans: Some of the key benefits of strict mode include:

* **Makes debugging easier.** Code errors that would otherwise have been ignored or would have failed silently will now generate errors or throw exceptions, alerting you sooner to problems in your code and directing you more quickly to their source.
* **Prevents accidental globals.** Without strict mode, assigning a value to an undeclared variable automatically creates a global variable with that name. This is one of the most common errors in JavaScript. In strict mode, attempting to do so throws an error.
* **Disallows duplicate property names or parameter values.** Strict mode throws an error when it detects a duplicate named property in an object (e.g., var object = {foo: "bar", foo: "baz"};) or a duplicate named argument for a function (e.g., function foo(val1, val2, val1){}).
* **Throws error on invalid usage of delete.**

What will the code below output? Explain your answer.

console.log(0.1 + 0.2);

console.log(0.1 + 0.2 == 0.3);

An educated answer to this question would simply be: “You can’t be sure. it might print out “0.3” and “true”, or it might not. Numbers in JavaScript are all treated with floating point precision, and as such, may not always yield the expected results.”

The example provided above is classic case that demonstrates this issue. Surprisingly, it will print out:

0.30000000000000004

false

Write a simple function (less than 80 characters) that returns a boolean indicating whether or not a string is a [palindrome](http://www.palindromelist.net/).

**Ans:**

The following one line function will return true if str is a palindrome; otherwise, it returns false.

function isPalindrome(str) {

str = str.toLowerCase();

// **String(str).toLowerCase();** if you are passing number then convert number to string first

return str == str.split('').reverse().join('');

}

// The join() method joins the elements of an array into a string, and returns the string. And remember we cannot use reverse method with string type so converting string into an array

console.log(isPalindrome("level")); // logs 'true'

console.log(isPalindrome("levels")); // logs 'false'

console.log(isPalindrome("Nitin")); // logs 'true', N is different than last character ‘n’ that is why we are converting our string to lowercase first

Write a sum method which will work properly when invoked using either syntax below.

console.log(sum(2,3)); // Outputs 5

console.log(sum(2)(3)); // Outputs 5

**Solution:**

function sum(x) {

if (arguments.length == 2) {

return arguments[0] + arguments[1];

}

// currying in else block

else {

return function(y) { return x + y; };

}

}

sum(2)(3) // sum(2, 3) is also acceptable

In JavaScript, functions provide access to an arguments object which provides access to the actual arguments passed to a function. This enables us to use the length property to determine at runtime the number of arguments passed to the function.

If two arguments are passed, we simply add them together and return.

Otherwise, we assume it was called in the form sum(2)(3), so we return an anonymous function that adds together the argument passed to sum() (in this case 2) and the argument passed to the anonymous function (in this case 3).

What will the code below output to the console and why ?

console.log(1 + "2" + "2");

console.log(1 + +"2" + "2");

console.log(1 + -"1" + "2");

console.log(+"1" + "1" + "2");

console.log( "A" - "B" + "2");

console.log( "A" - "B" + 2);

Hide answer

The above code will output the following to the console:

"122"

"32"

"02"

"112"

"NaN2"

NaN

What will the following code output to the console and why:

var hero = {

\_name: 'John Doe',

getSecretIdentity: function (){

return this.\_name;

}

};

var stoleSecretIdentity = hero.getSecretIdentity;

console.log(stoleSecretIdentity());

console.log(hero.getSecretIdentity());

The code will output:

undefined

John Doe

The first console.log prints undefined because we are extracting the method from the hero object, so stoleSecretIdentity()is being invoked in the global context (i.e., the window object) where the \_name property does not exist.

One way to fix the stoleSecretIdentity() function is as follows:

var stoleSecretIdentity = hero.getSecretIdentity.bind(hero);

# 1. What are JavaScript types?

There are 7 data-types in JavaScript. They are as follows:

1. Number // typeof 2;
2. String // typeof “2”;
3. Boolean // typeof true
4. Function // typeof function f(){}
5. Object // var a = {}; typeof a; [Array is also object]
6. Null // Remember, typeof null returns object
7. Undefined // typeof undefined

**4. What is the use of isNaN function?**

isNan function returns true if the argument is not a number otherwise it is false.