*Javascript*

***Value vs. Reference in Javascript :***

For primitive data types value is assigned and copied

For non-primitive data types reference is used

function changeAgeAndReference(person) {  
 person.age = 25;  
 person = {  
 name: 'John',  
 age: 50  
 };  
   
 return person;  
}

var personObj1 = {  
 name: 'Alex',  
 age: 30  
};

var personObj2 = changeAgeAndReference(personObj1);

console.log(personObj1); // -> ?  
console.log(personObj2); // -> ?

The function first changes the property age on the original object it was passed in. It then reassigns the variable to a brand new object and returns that object. Here’s what the two objects are logged out.

console.log(personObj1); // -> { name: 'Alex', age: 25 }  
console.log(personObj2); // -> { name: 'John', age: 50 }

Remember that assignment through function parameters is essentially the same as assignment with =. The variable person in the function contains a reference to the personObj1 object, so initially it acts directly on that object. Once we reassign person to a new object, it stops affecting the original.

**How to clear setTimeout in js ?**

var asd = setTimeout(function(){

console.log('hello')

},5000)

clearTimeout(asd)

**Q What is 'this' keyword in JavaScript?**

It has different values depending on where it is used:

* In a method, this refers to the **owner object**.



This refers to person here

* Alone, this refers to the **global object**.

(for example any where in global scope)

* In a function, this refers to the **global object**.



if function is used using .call()/.apply() / .bind() “then” this will refer to object passed in these 3

* In a function, in strict mode, this is undefined.

Until unless function is called using .call(), .apply() or .bind()

* In an event, this refers to the **element** that received the event.
* Methods like call(), and apply() can refer this to **any object**.

**Q What is the difference between ViewState and SessionState?**

'ViewState' is specific to a page in a session.

'SessionState' is specific to user specific data that can be accessed across all pages in the web application.

**Q What are the different types of errors in JavaScript?**

There are three types of errors:

Load time errors: Errors which come up when loading a web page like improper syntax errors are known as Load time errors and it generates the errors dynamically.

Run time errors: Errors that come due to misuse of the command inside the HTML language.

Logical Errors: These are the errors that occur due to the bad logic performed on a function which is having different operation.

**Q How will you explain closures in JavaScript? When are they used?**

Closure is a locally declared variable related to a function which stays in memory when the function has returned.

**a closure gives you access to an outer function’s scope from an inner function.**

To use a closure, simply define a function inside another function and **expose it**. To expose a function, return it or pass it to another function.

The nested function can access parent functions variable even after the parent function has returned

**closures are commonly used to give objects data privacy.**

**Closures for data pivacy**

Example: Employee Registration



However in this case employeeId can be easily corrupted by outside environment :



to make employeeId unaccessable to outside we create closure



Now the highlighted employeeId is created only once for create employee and can not be change except by the enclosing return function

**Q Describe the properties of an anonymous function in JavaScript?**

A function that is declared without any named identifier is known as an anonymous function. In general, an anonymous function is inaccessible after its declaration.

Anonymous function declaration -

var anon = function() {

alert('I am anonymous');

};

anon();

**Q What is the difference between .call() and .apply()?**

These JavaScript methods allow you to change the value of ‘this’ for a given function.

The function .call() and .apply() are very similar in their usage except a little difference. .call() is used when the number of the function's arguments are known to the programmer, as they have to be mentioned as arguments in the call statement. On the other hand, .apply() is used when the number is not known. The function .apply() expects the argument to be an array.

The basic difference between .call() and .apply() is in the way arguments are passed to the function. Their usage can be illustrated by the given example.



***.bind()***









**First-class functions:**

Functions in JavaScript are utilized as first class objects. This usually means that these functions can be passed as arguments to other functions, returned as values from other functions, assigned to variables or can also be stored in data structures

benefits, of including 'use strict' at the beginning of a JavaScript source file?

strict mode eliminates some JavaScript silent errors by changing them to throw errors. Second, strict mode fixes mistakes that make it difficult for JavaScript engines to perform optimizations: strict mode code can sometimes be made to run faster than identical code that's not strict mode.

**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.

"use strict";

x = 3.14;

**Eliminates this coercion**. Without strict mode, a reference to a this value of null or undefined is automatically coerced to the global. This can cause many headfakes and pull-out-your-hair kind of bugs. In strict mode, referencing a a this value of null or undefined throws an error.

**Disallows duplicate parameter values.** Strict mode throws an error when it detects a duplicate named argument for a function (e.g., function foo(val1, val2, val1){}), thereby catching what is almost certainly a bug in your code that you might otherwise have wasted lots of time tracking down.

Note: It used to be (in ECMAScript 5) that strict mode would disallow duplicate property names (e.g. var object = {foo: "bar", foo: "baz"};) but [as of ECMAScript 2015](https://stackoverflow.com/questions/30617139/whats-the-purpose-of-allowing-duplicate-property-names) this is no longer the case.

**Makes eval() safer.** There are some differences in the way eval() behaves in strict mode and in non-strict mode. Most significantly, in strict mode, variables and functions declared inside of an eval() statement are *not* created in the containing scope (they *are* created in the containing scope in non-strict mode, which can also be a common source of problems).

**Throws error on invalid usage of delete.** The delete operator (used to remove properties from objects) cannot be used on non-configurable properties of the object. Non-strict code will fail silently when an attempt is made to delete a non-configurable property, whereas strict mode will throw an error in such a case.

Property flags

Object properties, besides a **value**, have three special attributes (so-called “flags”):

* **writable** – if true, can be changed, otherwise it’s read-only.
* **enumerable** – if true, then listed in loops, otherwise not listed.
* **configurable** – if true, the property can be deleted and these attributes can be modified, otherwise not.

***What will be the output of the code below?***

*var Employee = { company: 'xyz' }*

*var emp1 = Object.create(Employee);*

*delete emp1.company*

*console.log (emp1.company);*

The output would be xyz. Here, emp1 object has company as its prototype property. **The delete operator doesn't delete prototype property.**

emp1 object doesn't have company as its own property.. However, we can delete the company property directly from theEmployee object using delete Employee.company. Or, we can also delete the emp1 object using the \_\_proto\_\_ property delete emp1.\_\_proto\_\_.company.

***MAP***

***array.map(function(currentValue, index, arr), thisValue)***

const officersIds = officers.map(officer => officer.id);

So how does .map() work? Basically is takes 2 arguments, a callback and an optional context (will be considered as this in the callback) which I did not use in the previous example. The callback runs for each value in the array and returns each new value in the resulting array.

Keep in mind that the resulting array will always be the same length as the original array.

**map vs forEach**

The forEach() method executes a provided function once per array element.

forEach doesn’t return anything. It calls a function once per element, but doesn’t do anything with the results of these calls.

This means forEach is best used for situations where you don’t need a new, modified array in return

***.reduce()***

Just like .map(), .reduce() also runs a callback for each element of an array. What’s different here is that reduce passes the result of this callback (the accumulator) from one array element to the other.

We need to know the total years of experience of all of them. With .reduce(), it’s pretty straightforward

Now let’s say I want to find which pilot is the most experienced one. For that, I can use reduce as well:

***.filter()***

What if you have an array, but only want some of the elements in it? That’s where .filter() comes in!

if the callback function returns true, the current element will be in the resulting array. If it returns false, it won’t be.

* **What is the advantage of using promise over callback?**

error handling in case of promise is easier

if nested calls then only one catch is required

SYNTAX OF PROMISE ON BK



***ES5 vs ES6***

* **Arrow function ( () => console.log() )**

   
this is how "this" was preserved for using inside callback without arrows.   
if direct "this" is called then it will reference to window object and not to the calling one.  
the second pic shows that using arraow function "this" can b called directly inside callbacks

* **Spread operator (...)**

could be used to clone an object (shallow ofcourse!)

let asd = [1,2,3]

cx = [...asd] // clone done

* **Object destructuring**





two new 'const' with "fullname"," place" are created . const age is created and assigned default value of 25

* **templpate strings**



* **Default parameters**



* **Let vs Var**
* Even if the let variable is defined as same as var variable globally, the let variable will not be added to the global window object.





* let variables cannot be re-declared while var variable can be re-declared in the same scope.
* variables declared with let are not accessible before they are declared in their enclosing block
* In case of var, after creating variable definitions, before executing line by line each of the variables is initialized with the undefined value.
* In case of let/const, the initialization to undefined does not happen until the line where the declaration actually happens. And only if there is no assignment immediately. On the lines above the variable is in the Temporal Dead Zone and accessing it results in Reference Error.

console.log(x); // undefined

console.log(y); // error!

var x;

let y

* ***JS Event Loop***

***The Call Stack***

*JavaScript has a single call stack in which it keeps track of what function we’re currently executing and what function is to be executed after that*

***The Event Table & Event Queue***

*Every time you call a setTimeout function or you do some async operation — it is added to the* ***Event Table****. This is a data structure which knows that a certain function should be triggered after a certain event. Once that event occurs (timeout, click, mouse move) it sends a notice. Bear in mind that the Event Table does not execute functions and does not add them to the call stack on it’s own. It’s sole purpose is to keep track of events and send them to the Event Queue.*

*The* ***Event Queue*** *is a data structure similar to the stack — again you add items to the back but can only remove them from the front. It kind of stores the correct order in which the functions should be executed. It receives the function calls from the Event Table, but it needs to* ***somehow send them to the Call Stack****? This is where the Event Loop comes in.*

***The Event Loop***

*We’ve finally reached the infamous Event Loop. This is a constantly running process that checks if the call stack is empty. Imagine it like a clock and every time it ticks it looks at the* ***Call Stack*** *and if it is empty it looks into the* ***Event Queue****. If there is something in the event queue that is waiting it is moved to the call stack. If not, then nothing happens.*

***All `thenable` callbacks of the promise are called first, then the setTimeout callback is called.***

***PROTOTYPES***

* Objects in JavaScript have an internal property known as **prototype.**Itis simply a reference to another object and contains common attributes/properties across all instances of the object
* The **Array** object has a prototype **Array.prototype** This is why you can use a method like **sort()** on an array instance.

**Prototype chain**

When an object gets a request for a property that it does not have, its prototype will be searched for the property, then the prototype’s prototype, and so on.

* **What is hoisting?**

In JavaScript, a variable can be declared after it has been used.

In other words; a variable can be used before it has been declared.

Hoisting is JavaScript's default behavior of moving all declarations to the top of the current scope (to the top of the current script or the current function).

JavaScript only hoists declarations, not initializations.

to avoid bugs, always declare all variables at the beginning of every scope. it is always a good rule.

**function definition hoisting only occurs for function declarations**, not function expressions.

*// Outputs: "Definition hoisted!"*

definitionHoisted();

*// TypeError: undefined is not a function*

definitionNotHoisted();

**function** definitionHoisted() {

console.log("body hoisted!"); //-- - dis is function declaration

}

**var** definitionNotHoisted **=** **function** () {

console.log("body not hoisted!"); //- - dis is function expression

};

* **Output**

**Var name = ‘rana’;**

**(Function(){**

**Console.log(name);**

**Var name = ‘arsal’;**

**})();**

* **For(i=0;i<10000;i++){console.log(i);} this takes more time to run on browser and less time in nodejs. Why?**
* **Output**

**Var a=’abc’;**

**Delete a;**

* **What is currying?**

you can pass all of the arguments a function is expecting and get the result, or pass a subset of those arguments and get a function back that’s waiting for the rest of the arguments.

var greetCurried = function(greeting) {

return function(name) {

console.log(greeting + ", " + name);

};

};

var greetHello = greetCurried("Hello");

greetHello("Heidi"); //"Hello, Heidi"

greetCurried("Hi there")("Howard"); //"Hi there, Howard"

* **Write native xhr request without using jquery.**

$.ajax({

    url : "<https://api.edjunction.com/api/getdata>", // - give example URL ?

    type: "POST",

    data : formData,

    success: function(data, textStatus, jqXHR)

    {

        //data - response from server

    },

    error: function (jqXHR, textStatus, errorThrown)

    {

    }

});

* **What is jquery event bubbling.**

When an event happens on an element, it first runs the handlers on it, then on its parent, then all the way up on other ancestors.

 a focus event does not bubble

* **How to stop event bubbling.**

event.stopPropagation()

* **What is output if click on b3 button.**

**<button id='i-0' >b1</button>**

**<button id='i-1' >b2</button>**

**<button id='i-2' >b3</button>**

**<script type="text/javascript">**

**var v= ['a','b','c'];**

**for(var itr in v){**

**document.getElementById('i-'+itr).onclick = function(){**

**console.log(v[itr]);**

**}**

**}**

* **Is javascript async or sync language?**

**Sync. It means koi operation aa jae to block ho jati ha to block ho jati ha. ajax make it non-bloking or async**

**However AJAX calls are async in JS**

Example of JS behaving async- - The setTimeout function is probably the simplest way to asynchronously schedule code to run in the future:

* **What is callback hell? How to avoid it?**

Nested call bak

a function that is to be executed after another function has finished executing. Any function that is passed as an argument to another function is called a callback function.

Use modular approach to avoid it or use promise library

* **What are promises?**

Same purpose as call bak

A Promise is in one of these states:

pending: initial state, neither fulfilled nor rejected.

fulfilled: meaning that the operation completed successfully.

rejected: meaning that the operation failed.

* **You have two async functions a and b (having unexpected callback time). How you will determine both of them completed their operations.**

*var promise1 = Promise.resolve(3);*

*var promise2 = 42;*

*var promise3 = new Promise(function(resolve, reject) {*

*setTimeout(resolve, 100, 'foo');*

*});*

*Promise.all([promise1, promise2, promise3]).then(function(values) {*

*console.log(values);*

*});*

* **What is jwt?**

**JSON web token, contains users info, JWT has headers, payloads,signatures**



* **How to keep session in angularjs?**

in service of angular

* **What are cookies? What is cookies alternative?**

Cookies are small files which are stored on a user's computer. They are designed to hold a modest amount of data specific to a particular client and website, and can be accessed either by the web server or the client computer. This allows the server to deliver a page tailored to a particular user,

* **What is pure function?**

**For same input gives same output each time. It leave no side effect. Does not impact global variables**

A **pure function** doesn't depend on and doesn't modify the states of variables out of its scope. Concretely, that means a **pure function** always returns the same result given same parameters. Its execution doesn't depend on the state of the system

* **What is ajax?**

**Get post request without page loading.**

load the external data directly into the selected HTML elements of your web page

* **Output**

// interviewer: what will the following code output?

const arr = [10, 12, 15, 21];

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

setTimeout(function() {

console.log('Index: ' + i + ', element: ' + arr[i]);

}, 3000);

}

// index: 4, element: undefined

//prints 4 times

* **What is functional programming?**

(functional programming produces programs by composing mathematical functions and avoids shared state & mutable data.

Pure functions / function purity,

first-class functions,

higher order functions,

functions as arguments/values.

* **What is the use of isNaN function?**

isNaN ('asd') // true

* **Difference between undefined and null.**

**undefined** means a variable has been declared but has not yet been assigned a value

null is an object

* **Difference between == and === .**

What === does, is that it checks not only the equality of the two values, it compares the types of the two values too. Using == compare only values and not type

true === 1 //does the boolean value of true equal 1? (False)

true == 1 //does the boolean value of true equal 1? (True)

* **Output**

**var x = {a:1}, y = {a:1}, z=x;**

**console.log(x==y); console.log(x==z);**

* **Output**

**anon1()**

**var anon1 = function() {**

**alert('I am anonymous');**

**};**

* **What are filters in angular js?**

Filters can be added in AngularJS to format data.

<p>The name is {{ lastName | uppercase }}</p>

<li ng-repeat="x in names | orderBy:'country'">

* **Wat is first classs function?**

 language supports passing functions as arguments to other functions, returning them as the values from other functions, and assigning them to variables or storing them in data structures

* **How will you implement inheritance in javascript?**
* **What is the difference between classical inheritance and prototypal inheritance?**
* **What are different javascript design patterns?**
* **What is nodejs design pattern?**
* **What is OWASP top 10.**
* **What are HTTP verbs.**

**GET,Post**

* **GET and POST can do same things. Why to use POST for form submission?**
* **Difference between DOM and BOM.**
* **Write about the errors shown in JavaScript? (load time, run time, logical errors)**
* **Why it is not advised to use innerHTML in JavaScript?( innerHTML content is refreshed every time and thus is slower. There is no scope for validation in innerHTML and, therefore, it is easier to insert rouge code in the document and, thus, make the web page unstable.)**
* **What is CORS? (node.js)**
* **Write a program that ouput frist 10 positive integers after every seconds. (use of setInterval )**
* **What are decorators?**
* **How to prevent xss attack on web?**
* **What are the pros and cons of functional programming vs object-oriented programming?**
* **What are the pros and cons of monolithic vs microservice architectures?**
* **What are two-way data binding and one-way data flow, and how are they different?**

