**Javascript Sources**

[**https://www.youtube.com/watch?v=wstwjQ1yqWQ**](https://www.youtube.com/watch?v=wstwjQ1yqWQ)

[**https://medium.com/quick-code/javascript-promises-in-twenty-minutes-3aac5b65b887**](https://medium.com/quick-code/javascript-promises-in-twenty-minutes-3aac5b65b887)

**Q. How to find the second max in an array**

**var arr = [12, 245, 45, 98, 12, 768];**

**let max= arr[0];**

**let secondMax= arr[0];**

**for (var i = 1; i < arr.length; i++) {**

**if (arr[i] > max) {**

**secondMax = max;**

**max = arr[i];**

**} else if (arr[i] > secondMax ) {**

**secondMax = arr[i];**

**}**

**}**

**Q. How can we achieve polymorphism in javascript**

[**https://www.youtube.com/watch?v=zdovG9cuEBA**](https://www.youtube.com/watch?v=zdovG9cuEBA) **- Checkout after 7 minutes**

**Q. What is session strorage and local storage**

**LocalStorage:**

**Web storage can be viewed simplistically as an improvement on cookies, providing much greater storage capacity. Available size is 5MB which considerably more space to work with than a typical 4KB cookie.**

**The data is not sent back to the server for every HTTP request (HTML, images, JavaScript, CSS, etc) - reducing the amount of traffic between client and server.**

**The data stored in localStorage persists until explicitly deleted. Changes made are saved and available for all current and future visits to the site​**

**sessionStorage:**

**It is similar to localStorage.**

**Changes are only available per window (or tab in browsers like Chrome and Firefox). Changes made are saved and available for the current page, as well as future visits to the site on the same window. Once the window is closed, the storage is deleted**

**The data is available only inside the window/tab in which it was set​**

**Q. Context in react**

**Context provides a way to pass data through the component tree without having to pass props down manually at every level.**

**const MyContext = React.createContext(defaultValue);**

**<MyContext.Provider value={/\* some value \*/}>**

**Q. How to pass a value using the react router**

**Using Render method it can pass params**

**<Route**

**path='/dashboard'**

**render={(props) => <Dashboard {...props} isAuthed={true} />}**

**/>**

**,**

**Q.How to call Concurrent API calls in javascript**

**Q. What it will do if i pass 0 setTimeout ({function(){**

**},0);**

**// Async , await**

**// What are the promises**

**// What is currying function**

**// What are the react hooks**

**// How to center a div**

**// what is event loooping , Event Que**

**// What is \_\_proto\_\_**

**// What is prototype pollution**

**// How the javascript engine implimented Let variable to have the scope particulary to block**

**// Object.assign is shallow copy or deep copy**

**// What is box model , What are the layout types in bootstrap**

**// What is th use of less , HOw to call class inside a class**

**// How to declare variables inside a less file**

**// How to extend a class using constructor?**

**// Solve - arr = ["Sravani", "1", "2", "false", "4"]; return only numbers array == [1,2,4]**

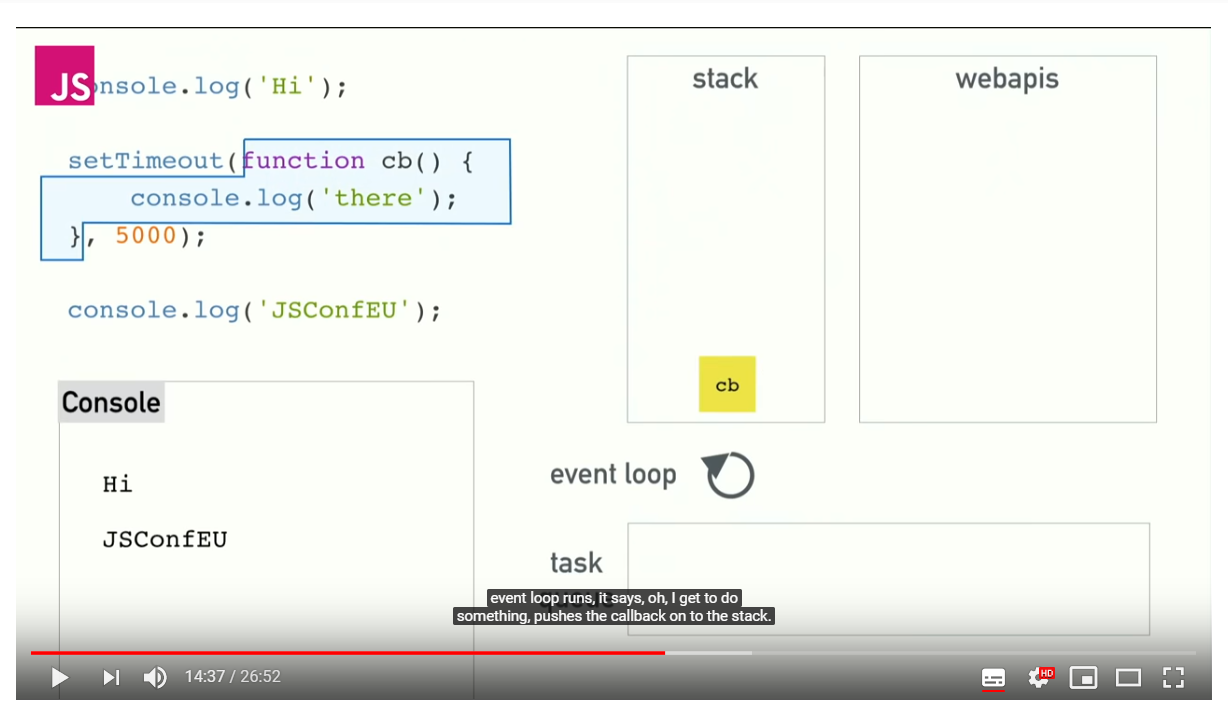
**// Solve input Array = [[1,2],[1,2,3],[1,2,3,4]] , Output should be = [1,2,3,4]; using spread parameter**

**// How garbage collector works**

**// why we need threads**

**// What is prototype chaining**

**Q. What is event looping**

****

**Here first the main() -> hi-> timer callback will be running by web apis concurently and while it is runing by webapi , the other things will be run in the stack and pops up after it is done. WEb api will push settimeout callback to task que after it is done .**

**Event loop continiously checks stack and task que, stack is empty it will take from task q and push it to stack**

**Q. write a function to give out put sum(1)(2)(3)......... (n);**

**Var sum = function(a){**

**var *sum* = function(a){**

**return function(b){**

**if(a+b) {**

**return *sum*(a + b)**

**}**

**else{**

**return a;**

**}**

**}**

**}**

1. difference between this & $(this)

Ans : this is the DOM object, whereas $(this) is the jQuery wrapper around same.

When using this, you can call DOM methods on it, but not jQuery methods. When using $(this), you can call jQuery methods on it, but not DOM methods.

2. What is Cross site Scripting

Source Link : <https://www.youtube.com/watch?v=cbmBDiR6WaY&t=297s>

<https://www.youtube.com/watch?v=LcXLjj92xz4>

4. [] == true  
Ans : False

When you call if (arr == false) you compare *values* of this object and the primitive false value. Internally, arr.toString() is called, which returns an empty string "".

This is because toString called on Array returns Array.join(), and empty string is one of falsy values in JavaScript.

5. typeof null  
Ans : Object

6. isNaN(undefined)  
Ans: true. Any value other than number will return true for isNaN.

8. Different positions in CSS:   
 Static : Bydefault static will apply and we cant set left,top values

* Relative.: The origin of the each element is top left corner , and when we say relative moves the position accourding its own element oriign
* :fixed.: Fixed take the origin as body of the element
* Absolute.: When we say absolute it will take the origin from the parent which is having position “Relative”
* sticky.:The element is positioned based on the user's scroll position. A **sticky**element toggles between relative and fixed , depending on the scroll position.

9. How to apply multiple backgrounds  
**document.body.style.background = color;**

**<div class="multi-bg-example"></div>**

**.multi-bg-example { width: 100%; height: 400px;**

**background-image: url(https://mdn.mozillademos.org/files/11305/firefox.png), url(https://mdn.mozillademos.org/files/11307/bubbles.png), linear-gradient(to right, rgba(30, 75, 115, 1), rgba(255, 255, 255, 0)); background-repeat: no-repeat, no-repeat, no-repeat; background-position: bottom right, left, right;**

10. What is functional programming. How it is different from normal Javascript programming.

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which are data structures that contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods -- Wikipedia <https://en.wikipedia.org/wiki/Object-oriented_programming>

Functional programming is a programming paradigm, a style of building the structure and elements of computer programs, that treats computation as the evaluation of mathematical functions and avoids changing-[state](https://en.wikipedia.org/wiki/Program_state) and mutable data -- Wikipedia <https://en.wikipedia.org/wiki/Functional_programming>

In all programs, there are two primary components: the data (the stuff a program knows) and the behaviors (the stuff a program can do to/with that data). OOP says that bringing together data and its associated behavior in a single location (called an “object”) makes it easier to understand how a program works. FP says that data and behavior are distinctively different things and should be kept separate for clarity.

11. Function currying  
**Ex:1** **function add(num1){**

**return function (num2){**

**return function (num3){**

**return num1+num2+num3;**

**}**

**}**

**}**

**add(10)(20)(30);**

**Ex:2 const curry = x => y => z => x + y + z; // a curry function**

17. Explain real time examples of prototypal inheritance. How did you implement in your projects.

**What is Prototype: When the javascript engine create an object , it will automatically puts hidden property into the object and attaches to the original Properties and methods.**

**Prototype Chaining: Prototype chaining is something like everthying is Object in Js**

*Var arr = [“Sravani”, “Swetha”];*

*Arr.\_\_proto\_\_ == Array.prototype;*

*Arr.\_\_proto\_\_.\_\_proto\_\_ = Objec.prototye*

*Arr.\_\_proto\_\_.\_\_proto\_\_.\_\_proto\_\_ = null*

*var fun = function(){}*

*func.\_\_proto\_\_*

*ƒ () { [native code] }*

*func.\_\_proto\_\_.\_\_proto\_\_*

*{constructor: ƒ, \_\_defineGetter\_\_: ƒ, \_\_defineSetter\_\_: ƒ, hasOwnProperty: ƒ, \_\_lookupGetter\_\_: ƒ, …}*

*Func.\_\_proto\_\_.\_\_proto\_\_.\_\_proto\_\_ = null*

*Means everything is object and after prototype chaining it becomes null;*

*Lets create a sample and see how inheritance works ?*

***var object*** *= {*

***name****:****"Sravani"****,*

***city****:****"Hyderabad"****,*

*getIntro(){*

***console****.log(****`****${****this****.****name****}* ***came from*** *${****this****.****city****}****`****);*

*}*

*}*

***var object1*** *= {*

***name****:****"Yogitha"***

*}*

*Now, Object1.\_\_proto\_\_ will be pointing to Object.****prototype****;*

*If we set Object1.\_\_proto\_\_ =* ***object****;*

*When you try to check the any property on object1 / Child object and then it checks parent object*

*object1.\_\_proto\_\_ = object;*

*object1.getIntro(); // “Yogitha came from Hyderabad”*

**19. What is call , apply, bind method in javascript ?**

***let name****={*

***firstName****:* ***"Sravani"****,*

***lastName****:****"Cheripalli"****,*

*printFullName:* ***function****(){*

***console****.log(****this****.****firstName****+****" "****+* ***this****.****lastName****);*

*}*

*}*

***name****.printFullName();*

***let name2*** *={*

***firstName****:****'yogita'****,*

***lastName****:****"kalidindi"***

*}*

***name****.printFullName.call(****name2****);*

*Call is like borrwoing the methods from another method*

*In real life we will modify printFullName method like this*

***let name****={*

***firstName****:* ***"Sravani"****,*

***lastName****:****"Cheripalli"****,*

*}*

*printFullName =* ***function****(hometown){*

***console****.log(****this****.****firstName****+****" "****+* ***this****.****lastName+ “Hometown is”+hometown****);*

*}*

*printFullName.call(name,”Miryalaguda”); // Sravani Cheripalli hometown is Miryalaguda*

***let name2*** *={*

***firstName****:****'yogita'****,*

***lastName****:****"kalidindi"***

*}*

*printFullName.call(name2,”Vizag”); // Yogitha kalidindi hometown is vizag*

***Apply Also works same but it will pass arguements as array***

*printFullName.call(name2,[”Vizag”]); // Yogitha kalidindi hometown is vizag*

***Call and Apply will directly call the method but bind method returns the function which can be invoked later.***

*Let bound = printFullName.call(name,”Miryalaguda”);*

*bound() // Sravani Cheripalli hometown is Miryalaguda*

*19. Pollifyl for bind method*

***Polyfils are like browser fallbacks , If our browser does not support any of the features we will write the compatible own function.***

***Traditional way of bind works***

***let name1 = {***

***firstName :"sravani",***

***lastName : "Cheripalli"***

***}***

***let printName = function(hometowm){***

***console.log(this.firstName+" "+this.lastName+""+hometowm);***

***}***

***let printMyName = printName.bind(name1,"hyd");***

***printMyName(); //sravani Cheripalli***

***Function.prototype.mybind = function (...args){***

***let obj = this;***

***let params = args.slice(1);// it will remove the 1st item and return remaining in array***

***return function(...args2){***

***obj.apply(args[0], [...params,...args2]);***

***}***

***}***

***let printMyName2 = printName.mybind(name1,"Hyd");***

***printName("India");***

18. Explain how to do autosuggestions component?  
Debounce  
  
searchFunction = function(){  
//search logic  
}

function debounce(fn, delay){

let timeout;

return function(){

clearTimeout(timeout);

timeout = setTimeout(function(){

fn.call(this);

}, delay);

}

}

21. What is the value of window.document === document?

Ans : true

22. What is the difference between document.ready and window.ready?

Ans : The ready event occurs after the HTML document has been loaded, while the onload event occurs later, when all content (e.g. images) also has been loaded.

The onload event is a standard event in the DOM, while the ready event is specific to jQuery. The purpose of the ready event is that it should occur as early as possible after the document has loaded, so that code that adds functionality to the elements in the page doesn't have to wait for all content to load.

23. How to retrieve query params from the url?   
Ans : using window.location.search

24. Write poly-fill of Object.create and explain why it worked:

if(typeof Object.create != "function") {

Object.create = function(param) {

var Fun = function(){};

Fun.prototype = param;

return new Fun();

}

}  
  
function createPolyfill(o){

if(!Object.create){

function F(){}  
 F.prototype = o;  
 return new F();

}

}

25. What new does when we create instance, what if we do not add it etc.

Ans : new will set the \_\_proto\_\_ property of the object being created to the prototype of constructor function.

26. What is the output of the followin code

var obj = { x: 1, y: 2 };

function sum (arg1, arg2, arg3) {

return this.x + this.y + arg1 + arg2 + arg3;

}

var myCaller = sum.bind(obj, 4);

myCaller(5, 6); //18 is the answer

3. Flat an n level array - [[1, 2], [[3], [4]], 5] should return [1, 2, 3, 4, 5]

Ans :

function flatten(arr, res=[]){

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

if(Array.isArray(arr[i])){

arr.contact(flatten(arr[i], res));

} else {

res.push(arr[i]);

}

}

return res;

}

4. Implement curry function:

var temp = curry(avg, 1, 2, 3);

temp(10); //4 - stores 1, 2, 3 in closures and adds 10 for average

temp(1, 2); //1.8 - stores 1, 2, 3 in closures and add 1, 2 for average

//Average function:

var avg = function (...param) {

var sum = 0;

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

sum += param[i];

}

return param.length > 0 ? sum / param.length : 0;

}

var curry = function (fn, ...m) {

return function (...n) {

return fn.apply(this, m.concat(n));

}

}

13. Object.freeze and Object.seal. Another way of asking is, how will you prevent object property value to be updated or added?  
Ans : Similarity: both of them are used for creating **non extensible objects**.

* Difference : In Freeze ***configurable , enumerable and writable*** attributes of the object are set to false. where as in Sealed ***writable*** attribute is set to true and rest of the attributes are false.

In case of freeze, we cant update prop values. Also cannot extend object with new properties.  
In case of seal, we can change prop values but cannot add new properties.

15. In what order will the numbers 1-4 be logged to the console when the code below is executed? Why?

(function() {

console.log(1);

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

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

console.log(4);

})();

Ans :

1  
4  
3  
2  
Bacause, set time out will push the function at the top of execution stack after the given time interval.

16. What is the output of the following code? Explain your answer.

var a={},

b={key:'b'},

c={key:'c'};

a[b]=123;

a[c]=456;

console.log(a[b]);

a[Object Object] = 123;  
  
Because if we set any property for an object in javascript, it will stringify the prop name and then assign some value.

22. lib.add(10).sub(20).mul(2).div(1) - Implement a function with use of closure  
Similar question Implement jQuery, underscore like library?

Ans : (function(global){

var lib = function(number){

return new lib.init(number);

}

lib.prototype = {

add : function(n){

this.number += n;

return this;

},

sub : function(n){

this.number -= n;

return this

},

multiply : function(n){

this.number\*= n;

return this;

},

divide : function(n){

this.number/=n

return this;

}

}

lib.init = function(number){

this.number = number;

}

lib.init.prototype = lib.prototype;

global.lib = lib;

})(window);

31. What is the output of following? What is the reason for output, how many closures are created in following example?

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

setTimeout(function () {

console.log(i);

}, 1000\*i);

}

Fix:

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

(function (i) {

setTimeout(function () {

console.log(i);

}, 1000\*i);

})(i);

}

32. Output of following

(function() {

x = 1;

function x() {};

var x;

console.log('var x => ', x);

})()

Ans : 1  
It's important to keep a few things in mind when declaring JavaScript functions and variables.

1. Variable assignment takes precedence over function declaration
2. Function declarations take precedence over variable declarations

Function declarations are hoisted over variable declarations but not over variable assignments.  
So order is :  
Variable declaration < Function declaration < variable assignment.

33. Question on static function:

function A() {}

// Static

A.getSmth = function() {

return 1;

}

// Public

A.prototype.getAnother = function() {

return 2;

}

var a = new A();

try {

console.log(a.getSmth());

} catch(e) {

console.log('a.getSmth() TypeError => ', e.constructor.name);

}

34. Output of following:

var head = {};

var newNode = {};

head.next = newNode;

newNode.next = head;

head = null;

console.log('newNode.next => ', newNode.next);

Array of arrays - flatten out as a single array of values. ex: [1, [2, [3, 4]]] => [1, 2, 3, 4]

function flatten(arr, res=[]){

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

if(Array.isArray(arr[i])){

flatten(arr[i], res);

} else {

res.push(arr[i]);

}

}

return res;

}

function flatten(arr, res=[]){

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

Array.isArray(arr[i])? res.concat(flatten(arr[i], res)) : res.push(arr[i])

}

return res;

}

Write a function to get factorial and cache the result. If it's called next time, return from cache.

a. How can you improve the performance of factorial using memoization?

b. How can you improve memoization approach since it takes more space?

c. We can ask which one is better recursive or iterative version of factorial?  
Ans :   
let lookup = [];  
function factorial(n){

if(lookup[n]){

return lookup[n];

}

lookup[n] = n > 1 ? n \* factorial(n-1) : 1;

return lookup[n];

}

**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.   
The process is called “bubbling”, because events “bubble” from the inner element up through parents like a bubble in the water.  
By default, events are bubbled out.  
**Stopping bubbling :**

A bubbling event goes from the target element straight up. Normally it goes upwards till <html>, and then to document object, and some events even reach window, calling all handlers on the path.

But any handler may decide that the event has been fully processed and stop the bubbling.

The method for it is event.stopPropagation().

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

<body onclick="alert(`the bubbling doesn't reach here`)">

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

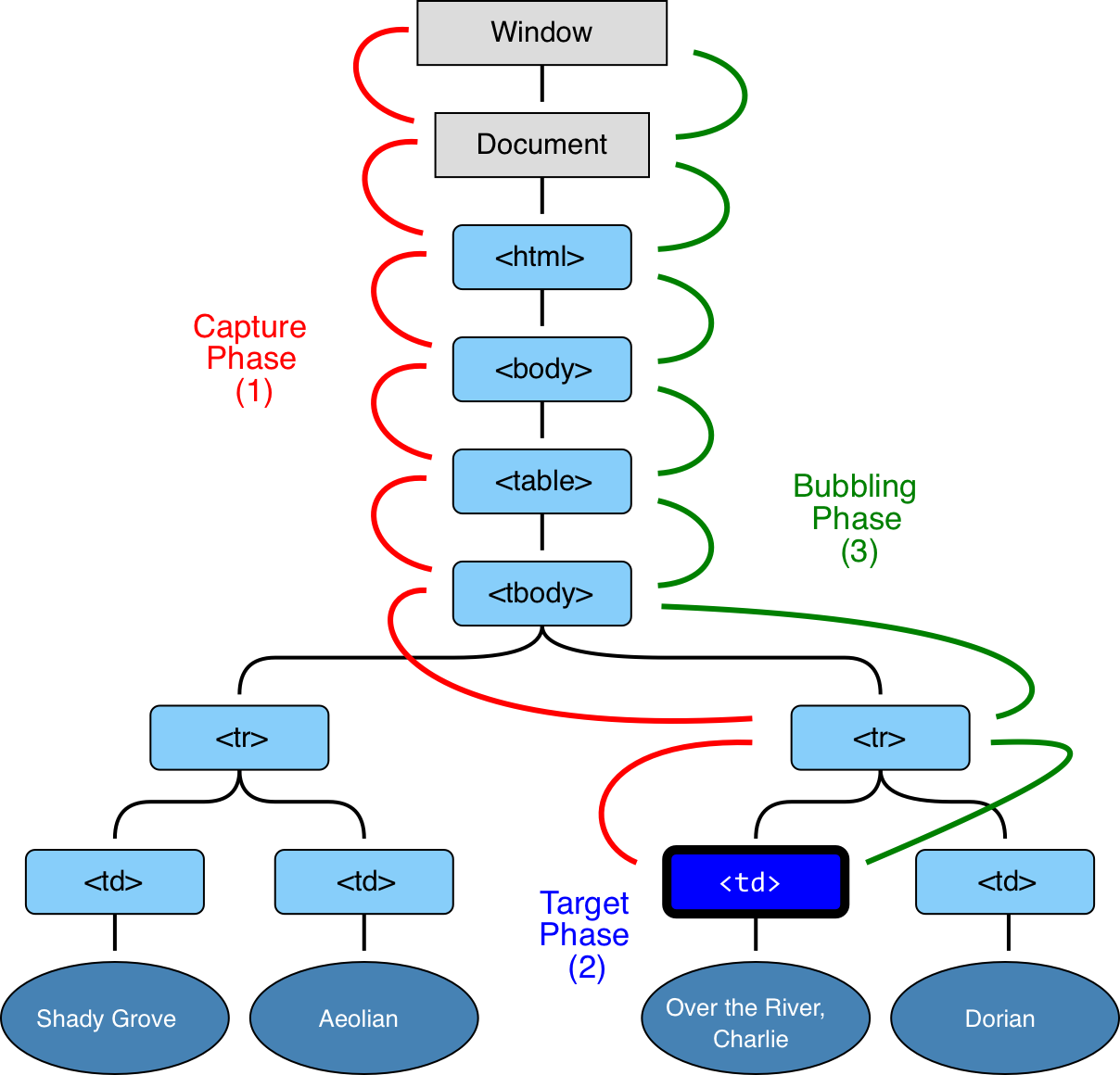
</body>

**Event Capturing :  
There’s another phase of event processing called “capturing”. It is rarely used in real code, but sometimes can be useful.**

**The standard** [**DOM Events**](http://www.w3.org/TR/DOM-Level-3-Events/) **describes 3 phases of event propagation:**

1. **Capturing phase – the event goes down to the element.**
2. **Target phase – the event reached the target element.**
3. **Bubbling phase – the event bubbles up from the element.**

**Here’s the picture of a click on <td> inside a table, taken from the specification:**

****

**That is: for a click on <td> the event first goes through the ancestors chain down to the element (capturing), then it reaches the target, and then it goes up (bubbles), calling handlers on its way.**

**elem.addEventListener(..., {capture: true})**

**// or, just "true" is an alias to {capture: true}**

**elem.addEventListener(..., true)**

**Different positions in CSS:**   
Relative, absolute, fixed.

|  |  |  |
| --- | --- | --- |
| static | Default value. Elements render in order, as they appear in the document flow |  |
| absolute | The element is positioned relative to its first positioned (not static) ancestor element |  |
| fixed | The element is positioned relative to the browser window |  |
| relative | The element is positioned relative to its normal position, so "left:20px" adds 20 pixels to the element's LEFT position |  |
| sticky | The element is positioned based on the user's scroll position  A sticky element toggles between relative and fixed, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position:fixed). |  |
| initial | Sets this property to its default value. [Read about *initial*](https://www.w3schools.com/cssref/css_initial.asp) |  |
| inherit | Inherits this property from its parent element. [Read about *inherit*](https://www.w3schools.com/cssref/css_inherit.asp) |  |

**Hoisting in javascript.**

Ans : variable declarations are moved to top of the scope. First variable declarations are moved. Then function declarations are moved. Assignment will happen at the place where its assigned.

**const obj = { "a" : 1 }**

**Can I change obj.a and why can I change it if const means that variable cannot be reassigned or re-declared?**

Ans : because objects are call by reference. So here we cannot change reference of the object but we can assign new values to it.  
  
**What will be output of following:**

var a = '1';

var b = 'c';

console.log(a + b); //1c

console.log(a - b); //NaN

console.log(a \* b); //NaN

console.log(a / b); //NaN

var a = 1;

var b = '2';

console.log(a + b); //12

console.log(a - b); //-1

console.log(a \* b); //2

console.log(a / b); //.5

**JavaScript is pass by value or pass by reference?**

Ans : Primitive type (string, number, etc.) are passed by value and objects are passed by reference

Memory leaks in closures?

A closure is a combination of a function and the lexical environment within which that function was declared. A closure is an inner(enclosed) function that has access to the outer (enclosing) function’s variables(scope). Also the inner function will continue to have access to the outer function’s scope even after the outer function is executed.

A memory leak occurs in a closure if a variable is declared in outer function becomes automatically available to the nested inner function and continues to reside in memory even if it is not being used/referenced in the nested function.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | <script>  var newElem;    function outer() {  var someText = new Array(1000000);  var elem = newElem;    function inner() {  if (elem) return someText;  }    return function () {};  }    setInterval(function () {  newElem = outer();  }, 5);  </script> |

In the above example, function inner is never called but keeps a reference to elem. But as all inner functions in a closure share the same context, inner(line 7) shares the same context as function(){} (line 12)which is returned by outer function. Now in every 5ms we make a function call to outer and assign its new value(after each call) to newElem which is a global variable. As long a reference is pointing to this function(){}, the shared scope/context is preserved and someText is kept because it is part of the inner function even if inner function is never called. Each time we call outer we save the previous function(){} in elem of the new function. Therefore again the previous shared scope/context has to be kept. So in the nth call of outer function, someText of the (n-1)th call of outer cannot be garbage collected. This process continues until your system runs out of memory eventually.

**SOLUTION:** The problem in this case occurs because the reference to function(){} is kept alive. There will be no javascript memory leak if the outer function is actually called(Call the outer function in line 15 like newElem = outer()();). A small isolated javascript memory leak resulting from closures might not need any attention. However a periodic leak repeating and growing with each iteration can seriously damage the performance of your code.

What is the output of following? What is the reason for output, how many closures are created in following example?

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

setTimeout(function () {

console.log(i);

}, 1000\*i);

}

Fix:

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

(function (i) {

setTimeout(function () {

console.log(i);

}, 1000\*i);

})(i);

}

Question on static function:

function A() {}

// Static

A.getSmth = function() {

return 1;

}

// Public

A.prototype.getAnother = function() {

return 2;

}

var a = new A();

try {

console.log(a.getSmth());

} catch(e) {

console.log('a.getSmth() TypeError => ', e.constructor.name);

}

Flatten out JSON data - like the flatten array   
Solution :

function flattenObj(obj, parent, res = {}){

for(let key in obj){

let propName = parent ? parent + '\_' + key : key;

if(typeof obj[key] == 'object'){

flattenObj(obj[key], propName, res);

} else {

res[propName] = obj[key];

}

}

return res;

}

Implement Design pattern - Pub/Sub, Mixin, Observer, revealing module pattern?

Ans :  
Observer and pub/sub are same.  
class MyChannel{

constructor(content=[], subscribers=[]){

this.content = content;

this.subscribers = subscribers;

}

update(content){

this.content = content;

this.notifySubscribers();

}

subscribe(subscriber){

this.subscribers.push(subscriber);

}

notifySubscribers(){

this.subscribers.forEach(sub=>{

sub.message = 'new Content added to my channel';

})

}

}

class Subscriber{

constructor(subscriptions=[]){

this.message = '';

this.subscriptions = subscriptions;

}

subscribe(channel){

this.subscriptions.push(channel);

}

}

**Revealing module Pattern**Javascript does not have the typical 'private' and 'public' specifiers of more traditional object oriented languages like C# or Java. However, you can achieve the same effect through the clever application of Javascript's function-level scoping. The Revealing Module pattern is a design pattern for Javascript applications that elegantly solves this problem.

The central principle of the Revealing Module pattern is that all functionality and variables should be hidden *unless deliberately exposed*.

**var musicModule = (function () {**

**// Let's make sure no one can directly access our songList**

**var songList = ['California Girls', 'California Dreaming', 'Hotel California'];**

**// We'll expose all these functions to the user**

**function play () {**

**console.log('Im playing the next song!');**

**}**

**function pause () {**

**console.log('Im paused!');**

**}**

**function addTrackToMusicQueue (track) {**

**songList.push(track);**

**console.log('I added a song');**

**}**

**function showNextTrack () {**

**console.log('My next track is', songList[0]);**

**}**

**// Let's hide this function**

**function loadSong() {**

**filesystem.loadNextSong();**

**}**

**return {**

**playMusic: play,**

**pauseMusic: pause,**

**showUpNext: showNextTrack,**

**addTrack: addTrackToMusicQueue**

**}**

**})(); // our IIFE function (surrounded with parens) is invoked here**

**musicModule.playMusic(); // 'Im playing the next song!'**

**musicModule.pauseMusic(); // 'I'm paused!'**

**musicModule.showUpNext(); // 'The next track is California Girls'**

**musicModule.loadSong(); // error: not a function**

**musicModule.songList.push('White Rabbit'); // undefined**

Different levels of scopes in javascript  
Blockscope and function scope  
  
Ans : I think about the best I can do is give you a bunch of examples to study. Javascript programmers are practically ranked by how well they understand scope. It can at times be quite counter-intuitive.

1. **A globally-scoped variable**// global scope var a = 1; function one() { alert(a); // alerts '1' }
2. **Local scope**// global scope var a = 1; function two(a) { // passing (a) makes it local scope alert(a); // alerts the given argument, not the global value of '1' } // local scope again function three() { var a = 3; alert(a); // alerts '3' }
3. **Intermediate**: *No such thing as block scope in JavaScript* (ES5; ES6 introduces [let](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let))  
   a.  
   var a = 1; function four() { if (true) { var a = 4; } alert(a); // alerts '4', not the global value of '1' }  
   b.  
   var a = 1; function one() { if (true) { let a = 4; } alert(a); // alerts '1' because the 'let' keyword uses block scoping }
4. **Intermediate**: *Object properties*var a = 1; function Five() { this.a = 5; } alert(new Five().a); // alerts '5'
5. **Advanced**: *Closure*var a = 1; var six = (function() { var a = 6; return function() { // JavaScript "closure" means I have access to 'a' in here, // because it is defined in the function in which I was defined. alert(a); // alerts '6' }; })();
6. **Advanced**: *Prototype-based scope resolution*var a = 1; function seven() { this.a = 7; } // [object].prototype.property loses to // [object].property in the lookup chain. For example... // Won't get reached, because 'a' is set in the constructor above. seven.prototype.a = -1; // Will get reached, even though 'b' is NOT set in the constructor. seven.prototype.b = 8; alert(new seven().a); // alerts '7' alert(new seven().b); // alerts '8'
7. **Global+Local**: *An extra complex Case*var x = 5; (function () { console.log(x); var x = 10; console.log(x); })();  
   This will print out undefined and 10 rather than 5 and 10 since JavaScript always moves variable declarations (not initializations) to the top of the scope, making the code equivalent to:  
   var x = 5; (function () { var x; console.log(x); x = 10; console.log(x); })();
8. **Catch clause-scoped variable**var e = 5; console.log(e); try { throw 6; } catch (e) { console.log(e); } console.log(e);  
   This will print out 5, 6, 5. Inside the catch clause e shadows global and local variables. But this special scope is only for the caught variable. If you write var f; inside the catch clause, then it's exactly the same as if you had defined it before or after the try-catch block.

What is closure

Ans : <https://medium.com/javascript-scene/master-the-javascript-interview-what-is-a-closure-b2f0d2152b36>

How will you handle date - time in your product?  
Ans : Using moment.js

What will be output of below:

X = [1,2]

X.push(X)

console.log(X)

Ans : A circular array with [1,2,[1,2,[1,2, [...]]]];

Output of following:

var head = {};

var newNode = {};

head.next = newNode;

newNode.next = head;

head = null;

console.log('newNode.next => ', newNode.next);

Ans : Circular Object

19. How can I get the depth of the deepest li tag in an unordered list?

http://stackoverflow.com/questions/22248432/calculate-depth-of-unordered-list

What is CORS.  
Ans : Cross Origin Request Sharing  
<https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>  
  
Difference between Webpack, gulp and Grunt.  
Ans : Webpack is a module bundler whereas gulp and grunt are task runner.  
  
Difference between classical and prototype inheritence? If you design a language, which one will you choose?  
  
Ans : <https://medium.com/javascript-scene/master-the-javascript-interview-what-s-the-difference-between-class-prototypal-inheritance-e4cd0a7562e9>

What is functional programming. How it is different from normal Javascript programming.  
Ans : There are a lot of ideas in the innocent looking definition above which all need to be understood before you can begin to grasp the meaning of functional programming:

* Pure functions
* Function composition
* Avoid shared state
* Avoid mutating state
* Avoid side effects

Custom array reduce method :  
  
Array.prototype.myReduce = function(reducer){

let accumulator = this[0];

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

accumulator = reducer(accumulator, this[i])

}

return accumulator;

}

Custom Array map method :  
  
Array.prototype.myMap = function(cb){

let res = [];

this.forEach(val=>{

res.push(cb(val));

});

return res;

}

Custom Array filter method :

Array.prototype.myFilter = function(cb){

let filteredArr = [];

this.forEach(val=>{

if(cb(val)) filteredArr.push(val);

})

return filteredArr;

}

What is the difference between document.load and window .load

What happens in the browser when we access a page

What is difference between inline and inline block

How this value works in case of arrow functions

Can we use arrow function in the constructor

How to use the sibling selectors in css

Life cycle of react

How to pass the values from child to parent

How to pass props to nested levels of context

How to call an API