Peinding topics:

* is js is sync or async?
* what is single thread,
* oops concepts,
* passing-by-value-vs-by-reference
* prototype
* inheritance
* currying
* diff var & let

**1Q.js is sync or async:**

**JavaScript** is always **synchronous** and single-threaded. ... **JavaScript** is only **asynchronous** in the sense that it can make, for example, Ajax calls.

Sync:

* JS code runs on single thread(can do 1thing at a time)
* synchronous code waits for 1actions to complete before moving on to the next

**2Q. OOPS concepts**

**Inheritance** (objects can inherit features from other objects),

**Polymorphism** (objects can share the same interface—how they are accessed and used—while their underlying implementation of the interface may differ), and

**Encapsulation** (each object is responsible for specific tasks).

**3Q.pass-by-value & pass-by reference**

pass-by-reference- like we have to understand the locations reference where that value assigned in the memory, that has location reference(address)--that is called reference

pass-by-value- assigning direct value Ex: var a = 1;

4q.

1q.

(function(){

  var a = b = 3;

})();

1a. b = 3;

var a = b;

->if var a = 4; defined in if or else then that should be available in outside also it that conditions satisfies.

->same thing if that variable defined in the function, that should be available on that function(like local scope), tested in console.

2q.

var myObject = {  
    foo: "bar",  
    func: function() {  
        var self = this;  
        console.log("outer func:  this.foo = " + this.foo);  
        console.log("outer func:  self.foo = " + self.foo);  
        (function() {  
            console.log("inner func:  this.foo = " + this.foo);  
            console.log("inner func:  self.foo = " + self.foo);  
        }());  
    }  
};

myObject.func();

2a.

outer func:  this.foo = bar

outer func:  self.foo = bar

inner func:  this.foo = undefined

inner func:  self.foo = bar

3q.

console.log(0.1 + 0.2);

console.log(0.1 + 0.2 == 0.3);

3a.

0.30000000000000004

False

4q.

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);

4a.

"122"

"32"

"02"

"112"

"NaN2"

NaN

5q.

for (var i = 0; i < 5; i++) {

       setTimeout(function() { console.log(i); }, i \* 1000 );

}

5a.

5, 5, 5, 5, and 5.

6q.

console.log("0 || 1 = "+(0 || 1));

console.log("1 || 2 = "+(1 || 2));

console.log("0 && 1 = "+(0 && 1));

console.log("1 && 2 = "+(1 && 2));

6a.

0 || 1 = 1

1 || 2 = 1

0 && 1 = 0

1 && 2 = 2

->blind hint, if 0 exist means answer is 0. If any other number me last numer(1&&2), 2 is last number

7q.

console.log(false == '0')

console.log(false === '0')

7a.

true

false

8q.

var a={},

    b={key:'b'},

    c={key:'c'};

a[b]=123;

a[c]=456;

console.log(a[b]);

8a.456

9q.

(function(x) {

    return (function(y) {

        console.log(x);

    })(2)

})(1);

9a. 1

10q.

var hero = {

    \_name: 'John Doe',

    getSecretIdentity: function (){

        return this.\_name;

    }

};

var stoleSecretIdentity = hero.getSecretIdentity;

console.log(stoleSecretIdentity());

console.log(hero.getSecretIdentity());

10a.

undefined

John Doe

11q.

var x = 21;

var girl = function () {

    console.log(x);

    var x = 20;

};

girl ();

11a.

var x = 21;

var girl = function () {

    console.log(x);

    var x = 20;

};

girl ();

11a.

undefined

->bcz of 2nd time defining x makes confusion of x in console. In internally var x and x defined has suppurate defining. So if remove var in 2nd x defining, gets x as 21, bcz that takes global.

12qa.

Void(0) is used to prevent the page from refreshing and parameter "zero" is passed while calling.

13q.

multiply(5)(6);

13a.

function multiply(a) {

  return function(b) {

    return a \* b;

  }

}  
multiply(5)(6);

14q.

How would you create a private variable in JavaScript?

14a.

function func() {  
  var priv = "secret code";  
  return function() {  
    return priv;  
  }  
}  
var getPriv = func();  
console.log(getPriv()); // => secret code

15q.

|  |  |  |
| --- | --- | --- |
| var y = 1;    if (function f(){}) {      y += typeof f;    }    console.log(y); | **var** k = 1;  **if** (1) {     eval(**function foo**(){})      k += **typeof** foo;    }    console.log(k); | **var** k = 1;  **if** (1) {  **function foo**(){}      k += **typeof** foo;    }    console.log(k); |
| Output: 1object | 1undefined | *1function* |

16q.

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

17q.

**var** bar = true;

console.log(bar + 0);

console.log(bar + "xyz");

console.log(bar + true);

console.log(bar + false);

17a.

1, "truexyz", 2, 1

18q.

**var** z = 1, y = z = **typeof** y;

console.log(y);

18a.

Undefined

19q.

**var** foo = **function bar**(){ **return**12; };

**typeof** bar();  
console.log(typeof foo());

19a.

Reference Error.

number

20q.

function test() {  
   console.log(a);  
   console.log(foo());  
   var a = 1;  
function foo() {  
 return 2;  
}  
}

test();

20a.

undefined and 2

21q.

Event delegation

21a.

* Usual in \***ngFor inside element we putting click events, if its exceeds 10k, means will get error/performance issue, to over come this we have to use event delegations, that means one click uses for entire loop(**\*ngFor), we have pass parameter like event, and in .js file we have to write like event.target.node like that….!

22q.1q.console.log('hello'.repeatify(3));

22a.

String.prototype.repeatifys = String.prototype.repeatifys || function(times) {

var str = '';  
 for (var i = 0; i < times; i++) {  
 str += this;  
 }  
 return str;  
};

hellohellohello.

23q.

var fullname = 'John Doe';  
var obj = {  
 fullname: 'Colin Ihrig',  
 prop: {  
 fullname: 'Aurelio De Rosa',  
 getFullname: function() {  
 return this.fullname;  
 }  
 }  
};  
console.log(obj.prop.getFullname());  
var test = obj.prop.getFullname;  
console.log(test());

23a. Aurelio De Rosa and John Doe

24q.

Let vs var

24a.

* Scenarios1

let me = 'go'; // globally scoped

var i = 'able'; // globally scoped

console.log(window.me); // undefined

console.log(window.i); // 'able'

* Scenarios2

|  |  |
| --- | --- |
| **Input**:  console.log(x);  var x=5;  console.log(x);  **Output**:  undefined  5 | **Input**:  console.log(x);  let x=5;  console.log(x);  **Output**:  error |

**var** is scoped to the nearest function block and

**let** is scoped to the nearest enclosing block, which can be smaller than a function block