paulfranco / **JavaScript Interview Questions.md**

Created 4 years ago

JavaScript Interview Questions.md

JavaScript Interview Questions

Q1: Explain equality in JavaScript ☆

Answer: JavaScript has both strict and type–converting comparisons:

- **Strict comparison (e.g., ===)** checks for value equality without allowing *coercion*
- **Abstract comparison (e.g. ==)** checks for value equality with *coercion* allowed

```
var a = "42";
var b = 42;

a == b;                 // true
a === b;                // false
```

Some simple equalityrules:

- If either value (aka side) in a comparison could be the `true` or `false` value, avoid `==` and use `===`.
- If either value in a comparison could be of these specific values (`0`, `""`, or `[]` -- empty array), avoid `==` and use `===`.
- In all other cases, you're safe to use `==`. Not only is it safe, but in many cases it simplifies your code in a way that improves readability.

Q2: Explain Null and Undefined in JavaScript ☆☆

Answer: JavaScript (and by extension TypeScript) has two bottom types: `null` and `undefined`. They are *intended* to mean different things:

- Something hasn't been initialized : `undefined`.

- Something is currently unavailable: `null`.

Q3: What is Scope in JavaScript? ☆

Answer: In JavaScript, each function gets its own *scope*. Scope is basically a collection of variables as well as the rules for how those variables are accessed by name. Only code inside that function can access that function's scoped variables.

A variable name has to be unique within the same scope. A scope can be nested inside another scope. If one scope is nested inside another, code inside the innermost scope can access variables from either scope.

Q4: Explain Values and Types in JavaScript ☆☆

Answer: JavaScript has typed values, not typed variables. The following built-in types are available:

- `string`
- `number`
- `boolean`
- `null` and `undefined`
- `object`
- `symbol` (new to ES6)

Q5: What is typeof operator? ☆

Answer: JavaScript provides a `typeof` operator that can examine a value and tell you what type it is:

```
var a;
typeof a;                               // "undefined"

a = "hello world";
typeof a;                               // "string"

a = 42;
typeof a;                               // "number"

a = true;
typeof a;                               // "boolean"

a = null;
typeof a;                               // "object" -- weird, bug

a = undefined;
typeof a;                               // "undefined"
```

```
a = { b: "c" };
typeof a;                                   // "object"
```

Q6: What is the object type? ☆

Answer: The object type refers to a compound value where you can set properties (named locations) that each hold their own values of any type.

```
var obj = {
        a: "hello world", // property
        b: 42,
        c: true
};

obj.a;          // "hello world", accessed with doted notation
obj.b;          // 42
obj.c;          // true

obj["a"];       // "hello world", accessed with bracket notation
obj["b"];       // 42
obj["c"];       // true
```

Bracket notation is also useful if you want to access a property/key but the name is stored in another variable, such as:

```
var obj = {
        a: "hello world",
        b: 42
};

var b = "a";

obj[b];                 // "hello world"
obj["b"];               // 42
```

Q7: Explain arrays in JavaScript ☆

Answer: An `array` is an object that holds values (of any type) not particularly in named properties/keys, but rather in numerically indexed positions:

```
var arr = [
        "hello world",
        42,
        true
];

arr[0];                 // "hello world"
```

```
arr[1];                 // 42
arr[2];                 // true
arr.length;             // 3

typeof arr;             // "object"
```

Q8: What is Coercion in JavaScript?

Answer: In JavaScript conversion between different two build-in types called `coercion` . Coercion comes in two forms in JavaScript: *explicit* and *implicit*.

Here's an example of explicit coercion:

```
var a = "42";

var b = Number( a );

a;                              // "42"
b;                              // 42 -- the number!
```

And here's an example of implicit coercion:

```
var a = "42";

var b = a * 1;  // "42" implicitly coerced to 42 here

a;                              // "42"
b;                              // 42 -- the number!
```

Q9: What is strict mode? ☆☆

Answer: *Strict Mode* is a new feature in ECMAScript 5 that allows you to place a program, or a function, in a "strict" operating context. This strict context prevents certain actions from being taken and throws more exceptions.

```
// Non-strict code...

(function(){
  "use strict";

  // Define your library strictly...
})();

// Non-strict code...
```

Q10: What is let keyword in JavaScript? ☆☆

Answer: In addition to creating declarations for variables at the function level, ES6 lets you declare variables to belong to individual blocks (pairs of { .. }), using the `let` keyword.

Source: *github.com/getify*

Q11: What is a Polyfill? ☆☆

Answer: A polyfill is essentially the specific code (or plugin) that would allow you to have some specific functionality that you expect in current or "modern" browsers to also work in other browsers that do not have the support for that functionality built in.

- Polyfills are not part of the HTML5 standard
- Polyfilling is not limited to Javascript

Source: *programmerinterview.com*

Q12: Being told that an unsorted array contains (n - 1) of n consecutive numbers (where the bounds are defined), find the missing number in O(n) time ☆☆

Answer:

```
// The output of the function should be 8
var arrayOfIntegers = [2, 5, 1, 4, 9, 6, 3, 7];
var upperBound = 9;
var lowerBound = 1;

findMissingNumber(arrayOfIntegers, upperBound, lowerBound); // 8

function findMissingNumber(arrayOfIntegers, upperBound, lowerBound) {
  // Iterate through array to find the sum of the numbers
  var sumOfIntegers = 0;
  for (var i = 0; i < arrayOfIntegers.length; i++) {
    sumOfIntegers += arrayOfIntegers[i];
  }

  // Find theoretical sum of the consecutive numbers using a variation of
  // Formula: [(N * (N + 1)) / 2] - [(M * (M - 1)) / 2];
  // N is the upper bound and M is the lower bound

  upperLimitSum = (upperBound * (upperBound + 1)) / 2;
  lowerLimitSum = (lowerBound * (lowerBound - 1)) / 2;

  theoreticalSum = upperLimitSum - lowerLimitSum;
```

```
  return theoreticalSum - sumOfIntegers;
}
```

Source: *https://github.com/kennymkchan*

Q13: Remove duplicates of an array and return an array of only unique elements ☆☆

Answer:

```
// ES6 Implementation
var array = [1, 2, 3, 5, 1, 5, 9, 1, 2, 8];

Array.from(new Set(array)); // [1, 2, 3, 5, 9, 8]

// ES5 Implementation
var array = [1, 2, 3, 5, 1, 5, 9, 1, 2, 8];

uniqueArray(array); // [1, 2, 3, 5, 9, 8]

function uniqueArray(array) {
  var hashmap = {};
  var unique = [];

  for(var i = 0; i < array.length; i++) {
    // If key returns undefined (unique), it is evaluated as false.
    if(!hashmap.hasOwnProperty(array[i])) {
      hashmap[array[i]] = 1;
      unique.push(array[i]);
    }
  }

  return unique;
}
```

Source: *https://github.com/kennymkchan*

Q14: Given a string, reverse each word in the sentence ☆☆

Details: For example `Welcome to this Javascript Guide!` should be become `emocleW ot siht tpircsavaJ !ediuG`

Answer:

```
var string = "Welcome to this Javascript Guide!";

// Output becomes !ediuG tpircsavaJ siht ot emocleW
var reverseEntireSentence = reverseBySeparator(string, "");
```

```
// Output becomes emocleW ot siht tpircsavaJ !ediuG
var reverseEachWord = reverseBySeparator(reverseEntireSentence, " ");

function reverseBySeparator(string, separator) {
  return string.split(separator).reverse().join(separator);
}
```

Source: *https://github.com/kennymkchan*

Q15: Implement enqueue and dequeue using only two stacks ☆☆

Answer: *Enqueue* means to add an element, *dequeue* to remove an element.

```
var inputStack = []; // First stack
var outputStack = []; // Second stack

// For enqueue, just push the item into the first stack
function enqueue(stackInput, item) {
  return stackInput.push(item);
}

function dequeue(stackInput, stackOutput) {
  // Reverse the stack such that the first element of the output stack is
  // last element of the input stack. After that, pop the top of the outpu
  // get the first element that was ever pushed into the input stack
  if (stackOutput.length <= 0) {
    while(stackInput.length > 0) {
      var elementToOutput = stackInput.pop();
      stackOutput.push(elementToOutput);
    }
  }

  return stackOutput.pop();
}
```

Source: *https://github.com/kennymkchan*

Q16: Explain event bubbling and how one may prevent it ☆☆

Answer: Event bubbling is the concept in which an event triggers at the deepest possible element, and triggers on parent elements in nesting order. As a result, when clicking on a child element one may exhibit the handler of the parent activating.

One way to prevent event bubbling is using `event.stopPropagation()` or `event.cancelBubble` on IE < 9.

Source: *https://github.com/kennymkchan*

Q17: Write a "mul" function which will properly when invoked as below syntax. ☆☆

Details:

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

Answer:

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

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

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

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

Source: *github.com/ganqqwerty*

Q18: How to empty an array in JavaScript? ☆☆

Details:

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

How could we empty the array above?

Answer: Method 1

```
arrayList = [];
```

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

For Instance:

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

Method 2

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

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

For Instance:

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

Method 3

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

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

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

Method 4

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

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

Source: *github.com/ganqqwerty*

Q19: How to check if an object is an array or not? Provide some code. ☆☆

Answer:

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

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

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

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

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

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

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

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

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

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

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

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

In modern browser, you can also use:

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

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

Source: *github.com/ganqqwerty*

Q20: How would you use a closure to create a private counter? ☆☆

Answer: You can create a function within an outer function (a closure) that allows you to update a private variable but the variable wouldn't be accessible from outside the function without the use of a helper function.

```
function counter() {
  var _counter = 0;
  // return an object with several functions that allow you
  // to modify the private _counter variable
  return {
    add: function(increment) { _counter += increment; },
    retrieve: function() { return 'The counter is currently at: ' + _count
  }
}

// error if we try to access the private variable like below
// _counter;

// usage of our counter function
var c = counter();
c.add(5);
c.add(9);

// now we can access the private variable in the following way
c.retrieve(); // => The counter is currently at: 14
```

Source: *coderbyte.com*