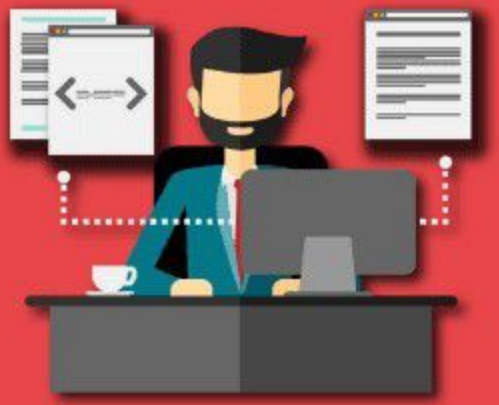

JAVASCRIPT INTERVIEW QUESTIONS

Q1: EXPLAIN VALUES AND TYPES IN JAVASCRIPT

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)*

Q2: WHAT IS LET KEYWORD IN JAVASCRIPT?

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.

Q3: EXPLAIN THE SAME-ORIGIN POLICY WITH REGARDS TO JAVASCRIP

The same-origin policy prevents JavaScript from making requests across domain boundaries. An origin is defined as a combination of URI scheme, hostname, and port number. This policy prevents a malicious script on one page from obtaining access to sensitive data on another web page through that page's Document Object Model.

Q4: WHAT IS THE DIFFERENCE BETWEEN == AND ===?

== is the abstract equality operator while === is the strict equality operator. The == operator will compare for equality after doing any necessary type conversions. The === operator will not do type conversion, so if two values are not the same type === will simply return false. When using ==, funky things can happen, such as:

```
1 == '1'; // true
1 == [1]; // true
1 == true; // true
```

My advice is never to use the == operator, except for convenience when comparing against null or undefined, where a == null will return true if a is null or undefined.


```
var a = null;
console.log(a == null); // true
console.log(a == undefined); // true
```

Q5: WHY WOULD YOU USE SOMETHING LIKE THE LOAD EVENT? DOES THIS EVENT HAVE DISADVANTAGES? DO YOU KNOW ANY ALTERNATIVES, AND WHY WOULD YOU USE THOSE?

The load event fires at the end of the document loading process. At this point, all of the objects in the document are in the DOM, and all the images, scripts, links and sub-frames have finished loading.
The DOM event DOMContentLoaded will fire after the DOM for the page has been constructed, but do not wait for other resources to finish loading. This is preferred in certain cases when you do not need the full page to be loaded before initializing.

Q6: WHAT'S THE DIFFERENCE BETWEEN HOST OBJECTS AND NATIVE OBJECTS?

- Native objects are objects that are part of the JavaScript language defined by the ECMAScript specification, such as String, Math, RegExp, Object, Function, etc.

- Host objects are provided by the runtime environment (browser or Node), such as window, XMLHTTPRequest, etc.

Q7: WHAT IS STRICT MODE?

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.

Q8: IS THERE ANYWAY TO FORCE USING STRICT MODE IN NODE.JS?

you can place

"use strict";

at the top of your file in node >= 0.10.7, but if you want your whole app to run in strict (including external modules) you can do this

node --use_strict

swipe

Q9: WHAT ARE SOME OF THE ADVANTAGES/DISADVANTAGES OF WRITING JAVASCRIPT CODE IN A LANGUAGE THAT COMPILES TO JAVASCRIPT?

Some examples of languages that compile to JavaScript include CoffeeScript, Elm, ClojureScript, PureScript, and TypeScript.

Advantages:

- Fixes some of the longstanding problems in JavaScript and discourages JavaScript anti-patterns.
- Enables you to write shorter code, by providing some syntactic sugar on top of JavaScript, which I think ES5 lacks, but ES2015 is awesome.
- Static types are awesome (in the case of TypeScript) for large projects that need to be maintained over time.

Disadvantages:

- Require a build/compile process as browsers only run JavaScript and your code will need to be compiled into JavaScript before being served to browsers.
- Debugging can be a pain if your source maps do not map nicely to your pre-compiled source.

Q9: WHAT ARE SOME OF THE ADVANTAGES/DISADVANTAGES OF WRITING JAVASCRIPT CODE IN A LANGUAGE THAT COMPILES TO JAVASCRIPT?

Disadvantages:

- Most developers are not familiar with these languages and will need to learn it. There's a ramp up cost involved for your team if you use it for your projects.
- Smaller community (depends on the language), which means resources, tutorials, libraries, and tooling would be harder to find.
- IDE/editor support might be lacking.
- These languages will always be behind the latest JavaScript standard.
- Developers should be cognizant of what their code is being compiled to—because that is what would actually be running, and that is what matters in the end.

Q10: EXPLAIN EVENT BUBBLING AND HOW ONE MAY PREVENT IT

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.

Q11: WHY IS IT, IN GENERAL, A GOOD IDEA TO LEAVE THE GLOBAL SCOPE OF A WEBSITE AS-IS AND NEVER TOUCH IT?

Every script has access to the global scope, and if everyone uses the global namespace to define their variables, collisions will likely occur. Use the module pattern (IIFEs) to encapsulate your variables within a local namespace.

Q12: WHAT IS A POLYFILL?

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

Q13: WHAT'S THE DIFFERENCE BETWEEN THROW ERROR('MSG') VS THROW NEW ERROR('MSG')?

Problem

```
var err1 = Error('message');
var err2 = new Error('message');
```

Which one is correct and why?

Both are fine; the function call Error(...) is equivalent to the object creation expression new Error(...) with the same arguments.

Q14: EXPLAIN NULL AND UNDEFINED IN JAVASCRIPT

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

- Something hasn't been initialised : undefined.
- Something is currently unavailable: null.

Non-zero value

0

null

undefined

JS

swipe

Q15: WHAT DOES USE STRICT DO?

The use strict literal is entered at the top of a JavaScript program or at the top of a function and it helps you write safer JavaScript code by throwing an error if a global variable is created by mistake. For example, the following program will throw an error:

```
function doSomething(val) {
  "use strict";
  x = val + 10;
}`
```

It will throw an error because x was not defined and it is being set to some value in the global scope, which isn't allowed with use strict The small change below fixes the error being thrown:

```
function doSomething(val) {
  "use strict";
  var x = val + 10;
}
```

Q16: WHAT IS SCOPE IN JAVASCRIPT?

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.

Q17: EXPLAIN ARRAYS IN JAVASCRIPT

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"
```

Q18: WHAT IS THE OBJECT TYPE?

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
```