JavaScript interview questions and answers

Tony Patton

My first adventures in Web development were brightened with the introduction of JavaScript (anybody remember LiveScript?). For many years, the language was shunned by self-proclaimed real developers, but there has been an about-face within the industry, with JavaScript now viewed as a major vehicle for building powerful Web applications and beyond (think Node.js).

When the time comes to hire JavaScript developers, the following interview questions and talking points might help. The depth and focus of an interview will depend on the position's requirements, so you can choose to go further with some questions or avoid topics based on your needs. I hope these questions will stimulate a discussion that gives you a peek into what the candidate knows and how the person works.

Note: This content is also available as a downloadable PDF.

What are global variables? How are they declared? What are the problems with using globals?

Global variables are available throughout your code: that is, the variables have no scope. Local variables scope, on the other hand, is restricted to where it is declared (like within a function). The var keyword is used to declare a local variable or object, while omitting the var keyword creates a global variable.

Most JavaScript developers avoid globals. One reason why is they're averse to naming conflicts between local and globals, Also, code that depends on globals can be difficult to maintain and test.

```
// Declare a local variable

var localVariable = "TechRepublic"
```

```
// Declare a global


globalVariable = "CNet"
```

How do you organize your JavaScript code?

The key concept here is to get an idea of how the candidate maintains and designs code. Do they design code that is specific to an application with no possible reuse? Do they use class inheritance or the module pattern to build reusable code? These approaches allow multiple developers to work on a project without stepping on their coworkers' toes. In addition, testing modular code or classes is easier to approach than a jumbled mess of code thrown together (look around the Web, and you'll find plenty of examples).

What are JavaScript types?

Unlike Java or C#, JavaScript is a loosely-typed language (some call this weakly typed); this means that no type declarations are required when variables are created. Strings and numbers can be intermixed with no worries. JavaScript is smart, so it easily determines what the type should be. The types supported in JavaScript are: Number, String, Boolean, Function, Object, Null, and Undefined.

```
var fName = "Mary";   //Declare a String

var total = 100.32;    //Declare a number


var fName = new String; //Another way to declare a string


fName = "Mary";


var total = new Number;


var isIt = new Boolean;

var names = new Array;

var car = new Object;
```

What is the difference between undefined and null?

The value of a variable with no value is undefined (i.e., it has not been initialized). Variables can be emptied by setting their value to null. You can test for each using the === (three equal signs) or == (two equal signs) for comparison checking. The big difference is the latter uses coercion, which can have some odd results — it returns true for a null or undefined comparison if they are either.

```
if (nullExample === null) { // executes this block only if null }

if (undExample ===Undefined) { // executes this block only if Undefined }


if (bothExampe == null) { // executes this block if Undefined or null }
```

You can be more exact with a comparison by using the typeof to return an object's type.

```
If (typeof variable ==="undefined")  { // executes this block of if undefined }
```

What is JavaScript's this keyword?

JavaScript's *this* keyword normally refers to the object that owns the method, but it depends on how a function is called. Basically, it points to the currently in scope object that owns where you are in the code. When working within a Web page, this usually refers to the Window object. If you are in an object created with the new keyword, the this keyword refers to the object being created. When working with event handlers, JavaScript's *this* keyword will point to the object that generated the event.

What is event bubbling?

Event bubbling describes the behavior of events in child and parent nodes in the Document Object Model (DOM); that is, all child node events are automatically passed to its parent nodes. The benefit of this method is speed, because the code only needs to traverse the DOM tree once. This is useful when you want to place more than one event listener on a DOM element since you can put just one listener on all of the elements, thus code simplicity and reduction. One application of this is the creation of one event listener on a page's body element to respond to any click event that occurs within the page's body.

Do you have a JavaScript framework preference? What are your thoughts on using frameworks?

This open-ended question has the potential to spawn a good conversation. There are the vastly popular frameworks like jQuery, although you might be surprised when the person tells you about the framework they developed or even played the contributor role.

The answer to the second question gives you an idea of the candidate's feelings about open source (well, that is the way I see it). There are so many open source options available today (Knockout, postal.js, jQuery, etc.), and a developer's time is very valuable, so why reinvent the wheel? These open source options provide robust code that has been thoroughly tested by an army of developers. From my perspective, I want developers who will use whatever is available to meet a project's demands. Plus, the interviewee might introduce you to something you've never used.

How are errors gracefully handled in JavaScript?

Exceptions that occur at runtime can be handled via try/catch/finally blocks; this allows you to avoid those unfriendly error messages. The finally block is optional, as the bare minimum to use is try/catch. Basically, you try to run code (in the try block between the braces), and execution is transferred to the catch block of code when/if runtime errors occur. When the try/catch block is finally done, code execution transfers to the finally code block. This is the same way it works in other languages like C# and Java.

```
try {

// do something


} catch (e) {

// do something with the exception

} finally {

// This code block always executes whether there is an exception or not.

}
```

You can give bonus points to any candidate who discusses the onerror event handler tied to the Window object in the browser — this allows it to monitor all errors on a page. This allows you to properly handle code syntax errors and runtime exceptions.

Can you explain how inheritance works in JavaScript?

This subject confuses many developers, and I would expect a candidate to stammer on this question or throw up their hands and say "can anybody?" Instead of trying to explain, take a look at this overview on the Mozilla Developer Network.

How do JavaScript timers work? What is a drawback of JavaScript timers?

Timers allow you to execute code at a set time or repeatedly using an interval. This is accomplished with the setTimeout, setInterval, and clearInterval functions. The setTimeout(function, delay) function initiates a timer that calls a specific function after the delay; it returns an id value that can be used to access it later. The setInterval(function, delay) function is similar to the setTimeout function except that it executes repeatedly on the delay and only stops when cancelled. The clearInterval(id) function is used to stop a timer. Timers can be tricky to use since they operate within a single thread, thus events queue up waiting to execute.

More in our development interview Q&A series