Raúl Cátedra Martínez

Basic JavaScript concepts

Raúl Cátedra Martínez

Know what **Javascript** is in general terms

JavaScript is the Programming Language for the Web. JavaScript can update and change both HTML and CSS. JavaScript can calculate, manipulate and validate data.

Briefly know its **history**

JavaScript was created by Brendan Eich in 1995 during his time at Netscape Communications. It was inspired by Java, Scheme and Self.

Netscape, for a time, made the best browser in the world and enjoyed market dominance.

In late 1995, when Microsoft cottoned-on to the competitive threat the Web posed, the Internet Explorer project was started in an all-out attempt to wrestle control of the emerging platform from Netscape.

In so doing Microsoft became a mortal threat, compelling Netscape to respond. First, they started a standardization process to prevent Microsoft gaining control of the JavaScript language. Second, they partnered with Sun to leverage their shared interest in breaking the Microsoft monopoly.

Sun began development of Java in 1990 in an attempt to write a language for “smart appliances”. This approach floundered and in 1994, Sun regrouped and set sights on the Web as the delivery platform of choice.

So the Netscape/Sun partnership meant Sun acquired the use of a competitive browser and a delivery system for their strategic technology.

Netscape, on the other hand found a powerful ally against Microsoft. They also aimed to out-manoeuvre Microsoft by being the official browser of the highly anticipated platform that was Java.

Brendan Eich has said that with Sun on board, they decided to surf the tidal wave of hype surrounding Java and position JavaScript as the companion language to Java, in the same way Visual Basic was to C++. So the name was a straightforward marketing ploy to gain acceptance.

Netscape’s Mocha (later JavaScript) amed to turn the web into a full-blown application platform. Furthermore, when used together with their LiveWire application server product, it would enable isomorphic development, with the same language used on both client and server.

If this sounds familiar, it is because this was exactly what Sun was attempting to pull off with Java. At the time however, the Web was very limited when compared to Java; for example, drawing pixels was not possible in JavaScript as it is now with canvas. So Sun (erroneously, I believe) never saw the language as a competitor and the alliance held.

Unfortunately for JavaScript, its early market positioning outlived its usefulness and later became a brake on market acceptance as it emerged as a viable technology in its own right.

So JavaScript was conceived as a scripting language for the Web for both client and server side. It was then quickly re-positioned as a Web “companion” for Java.

The unique circumstances of the birth of the language, including:

- the aforementioned marketing ploy,
- time-compressed initial development,
- a prejudice that development for the Web was not “serious”,
- the ubiquitous and “unbreakable” deployment environment (the Web), and
- the inclusion of language design elements unfamiliar to most developers

…led to a years-long period of misunderstanding, scorn and, yes, even hatred for the language. In the late 1990s and early 2000s even the authors of some books on JavaScript didn’t understand the fundamental elements of the language.

Douglas Crockford was one of the pioneers of the rediscovery of the language. And the importance of his invention of the JSON data format using a subset of JavaScript syntax should not be underestimated. During the 2000s mindshare slowly shifted to view JavaScript as a serious language: a critical mass of developers emerged who understood the language.

However, the “outsider status” of JavaScript continues to reverberate. Early design choices like automatic semicolon insertion (ASI), the event loop, lack of classes, unusual inheritance (prototypical) and type coercion are laughed at by people who have not taken the time to understand the thinking behind them.

Developers watch other developers laughing at these features and infer that these features are worthy of ridicule and the cycle continues.

Even Brendan Eich, the creator of the language, is occasionally apologetic for design decisions he made for the language.

However, in my view these expressions of apology should not be taken as confirmation that those decisions were wrong: but rather acknowledgement of the necessary inability of one language to please all developers.

As we will see in later posts, criticisms of JavaScript are frequently insubstantial expressions of unfamiliarity or syntactic “taste”, by programmers more familiar with other languages.

This is not to say the language is without flaws: the continuing lack of a decimal number primitive is unfortunate.

Know what is **ECMAScript**

Is a general-purpose programming language, standardized by Ecma International according to the document ECMA-262. It is a JavaScript standard meant to ensure the interoperability of Web pages across different Web browsers. ECMAScript is commonly used for client-side scripting on the Wold Wide Web, and it is increasingly being used for writing server applications and services using Node.js.

Raúl Cátedra Martínez

Understand what **events** are and what they are used for

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

- An HTML web page has finished loading.
- An HTML input field was changed.
- An HTML button was clicked.

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.

Understand how we can execute Javascript

To execute JavaScript in a browser you have two options — either put it inside a script element anywhere inside an HTML document, or put it inside an external JavaScript file (with a .js extension) and then reference that file inside the HTML document using an empty script element with a src attribute.

Know how to create **simple algorithms** with the following characteristics:

- **Variable declaration**

  Before 2015, using the `var` keyword was the only way to declare a JavaScript variable.

  The 2015 version of JavaScript (ES6 - ECMAScript 2015) allows the use of the `const` keyword to define a variable that cannot be reassigned, and the `let` keyword to define a variable with restricted scope.

- Use of **conditional statements**

  Very often when you write code, you want to perform different actions for different decisions.

  You can use conditional statements in your code to do this.

  In JavaScript we have the following conditional statements:

  - Use `if` to specify a block of code to be executed, if a specified condition is true
  - Use `else` to specify a block of code to be executed, if the same condition is false
  - Use `else if` to specify a new condition to test, if the first condition is false
  - Use `switch` to specify many alternative blocks of code to be executed

Raúl Cátedra Martínez

- ## **Use** and **declaration** of **functions**

  A JavaScript function is a block of code designed to perform a particular task.

  A JavaScript function is executed when "something" invokes it (calls it).

  A JavaScript function is defined with the `function` keyword, followed by a name, followed by parentheses ().

  Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

  The parentheses may include parameter names separated by commas: (parameter1, parameter2, ...)

  The code to be executed, by the function, is placed inside curly brackets: {}

- ## Use of simple **iterators**

  In JavaScript an iterator is an object which defines a sequence and potentially a return value upon its termination. Specifically, an iterator is any object which implements the Iterator protocol by having a next() method that returns an object with two properties: value. The next value in the iteration sequence.

  Once created, an iterator object can be iterated explicitly by repeatedly calling next(). Iterating over an iterator is said to consume the iterator, because it is generally only possible to do once. After a terminating value has been yielded additional calls to next() should simply continue to return {done: true}.

  The most common iterator in JavaScript is the Array iterator, which simply returns each value in the associated array in sequence.

- ## Use of **comments**

  You can use // for a single line coment or use /* - */ for multi-line comments.

- ## Import of **Javascript files** into **HTML documents**

  To include an external JavaScript file, we can use the script tag with the attribute src .

- ## Insertion of **Javascript** " **inline** " **codes**

  Inline JavaScript can be achieved by using Script tag inside the body of the HTML, and instead of specifying the source(src="…") of the JavaScript file in the Script tag, we have to write all the JavaScript code inside the Script tag.

- ## Simple **event application**

The change in the state of an object is known as an Event. In html, there are various events which represents that some activity is performed by the user or by the browser. When javascript code is included in HTML, js react over these events and allow the execution. This process of reacting over the events is called Event Handling. Thus, js handles the HTML events via Event Handlers.

For example, when a user clicks over the browser, add js code, which will execute t he task to be performed on the event.

You can also see the the apendix with all the posible events.

Apendix

Window Event Attributes

Attribute	Description
onafterprint	Script to be run after the document is printed
onbeforeprint	Script to be run before the document is printed
onebeforeunload	Script to be run when the document is about to be unloaded
onerror	Script to be run when an error occurs
onhashchange	Script to be run when there has been changes to the anchor part of the a URL
onload	Fires after the page is finished loading
onmessage	Script to be run when the message is triggered
onoffline	Script to be run when the browser starts to work offline
ononline	Script to be run when the browser starts to work online
onpagehide	Script to be run when a user navigates away from a page
onpageshow	Script to be run when a user navigates to a page
onpopstate	Script to be run when the window's history changes
onresize	Fires when the browser window is resized
onstorage	Script to be run when a Web Storage area is updated
onunluad	Fires once a page has unloaded (or the browser window has been closed)

Form Events

Attribute	Description
onblur	Fires the moment that the element loses focus
onchange	Fires the moment when the value of the element is changed
oncontextmenu	Script to be run when a context menu is triggered
onfocus	Fires the moment when the element gets focus
oninput	Script to be run when an element gets user input
oninvalid	Script to be run when an element is invalid
onreset	Fires when the Reset button in a form is clicked
onsearch	Fires when the user writes something in a search field (for <input="search">)
onselect	Fires after some text has been selected in an element
onsubmit	Fires when a form is submitted

Keyboard Events

Attribute	Description
onkeysown	Fires when a user is pressing a key
onkeypress	Fires when a user presses a key
onkeyup	Fires when a user releases a key

Mouse Events

Attribute	Description
onclick	Fires on a mouse click on the element

ondblclick	Fires on a mouse double-click on the element
onmousedown	Fires when a mouse button is pressed down on an element
onmousemove	Fires when the mouse pointer is moving while it is over an element
onmouseout	Fires when the mouse pointer moves out of an element
onmouseover	Fires when the mouse pointer moves over an element
onmouseup	Fires when a mouse button is released over an element
onmousewheel	Deprecated. Use the onwheel attribute instead
onwheel	Fires when the mouse wheel rolls up or down over an element

Drag Events

Attribute	Description
ondrag	Script to be run when an element is dragged
ondragend	Script to be run at the end of a drag operation
ondragenter	Script to be run when an element has been dragged to a valid drop target
ondragleave	Script to be run when an element leaves a valid drop target
ondragover	Script to be run when an element is being dragged over a valid drop target
ondragstart	Script to be run at the start of a drag operation
ondrop	Script to be run when dragged element is being dropped
onscroll	Script to be run when an element's scrollbar is being scrolled

Clipboard Events

Attribute	Description
oncopy	Fires when the user copies the content of an element
oncut	Fires when the user cuts the content of an element
onpaste	Fires when the user pastes some content in an element

Media Events

Attribute	Description
onabort	Script to be run on abort
oncanplay	Script to be run when a file is ready to start playing (when it has buffered enough to begin)
oncanplaythrough	Script to be run when a file can be played all the way to the end without pausing for buffering
oncuechange	Script to be run when the cue changes in a <track> element
ondurationchange	Script to be run when the length of the media changes
onemptied	Script to be run when something bad happens and the file is suddenly unavailable (like unexpectedly disconnects)
onended	Script to be run when the media has reach the end (a useful event for messages like "thanks for listening")
onerror	Script to be run when an error occurs when the file is being loaded
enloadeddata	Script to be run when media data is loaded
onloadedmetadata	Script to be run when meta data (like dimensions and duration) are loaded
onloadstart	Script to be run just as the file begins to load before anything is actually loaded

onpause	Script to be run when the media is paused either by the user or programmatically
onplay	Script to be run when the media is ready to start playing
onplaying	Script to be run when the media actually has started playing
onprogress	Script to be run when the browser is in the process of getting the media data
onratechange	Script to be run each time the playback rate changes (like when a user switches to a slow motion or fast forward mode)
onseeked	Script to be run when the seeking attribute is set to false indicating that seeking has ended
onseeking	Script to be run when the seeking attribute is set to true indicating that seeking is active
onstalled	Script to be run when the browser is unable to fetch the media data for whatever reason
onsuspend	Script to be run when fetching the media data is stopped before it is completely loaded for whatever reason
ontimeupdate	Script to be run when the playing position has changed (like when the user fast forwards to a different point in the media)
onvolumechange	Script to be run each time the volume is changed which (includes setting the volume to "mute")
onwaiting	Script to be run when the media has paused but is expected to resume (like when the media pauses to buffer more data)

Misc Events

Attribute	**Description**
ontoggle	Fires when the user opens or closes the <details> element