The Rise of JavaScript

Massimo DiPierro
DePaul University

Only a few years ago, JavaScript (JS) was a programming language disdained by many. Today, JS is very popular and one of the fastest-growing programming languages. It cannot be ignored, whether one develops for the web or not. In fact, as we will argue here, JS is an excellent development language for a certain class of scientific applications.

Brendan Eich invented JS in 1995 as a way to program one of the first popular web browsers, Netscape Navigator. Over time, other browsers have adopted JS as a primary language to add logic to webpages. The various implementations have since converged into a single, enhanced standard.

Even if originally developed as an interpreted language, modern browsers are smart, and if they observe that a certain JS function is called often, they compile it using a just-in-time (JIT) compiler. This makes JS a very fast language, almost as fast as C/C++ in some instances and usually faster than Java (which isn't related to JS, despite the name similarity). As a language designed to run primarily in the browser, JS provides a standard API to process HTML and JSON; handle network protocols; draw on a canvas; and interface with devices such as the computer's camera, audio, GPS, gyroscope, and even the GPU. The latest version of JS is called ECMAScript 6, or ES6 for short.

In 2009, Ryan Dahl invented node.js, an interpreter for running JS code server-side without the need for a browser. Since then, there has been a proliferation of command-line tools based on JS. Node.js includes a package manager called Node Package Manager (npm), which makes it easy to download additional modules and manage dependencies. One example of such modules is Babel, which can compile modern ES6 to previous versions of JS for portability. Another example is Browserify, which allows packaging of multiple JS files and their dependencies into a single portable .js file for ease of distribution.

Another major step in the history of JS was the invention of asm.js in 2013. This intermediate programming language allows C to compile into a portable and optimized subset of JS. Thanks to asm.js, it is possible to run an entire virtual machine (VM) in the browser and run any operating system in it. For example, to run Linux, just visit jslinux.org. Today, many languages can be compiled into JS, including Python.

If we have convinced you to try JS, there are a couple of features that you need to be aware of. These features can be confusing to programmers with experience in other languages:

- Until 2015, JS did not have a "class" keyword. Despite that, JS was always an object-oriented programming language with support for the basic functionalities of encapsulation, inheritance, polymorphism, and abstraction. The so-called *prototype interface* allows the creation of a prototype object from which other objects can be made. The keyword "class" was added purely as syntactic sugar.
- An important feature is the execution model. In JS, the code always runs single threaded, and it is impossible for two functions to run at the same time. That one thread maintains a queue of asynchronous tasks to be executed. Every time a function is called,

Copublished by the IEEE CS and the AIP
1521-9615/18/$33 ©2018 IEEE

it is placed in the queue as a closure (together with a copy of the calling environment), and functions in the queue are executed in the order in which they were called. Functions can also be scheduled to run at a later time or after another function has run (*promises*).

In this special issue of *CiSE*, we focus on a small sample of libraries and tools written in JS that might interest scientists and engineers. Our goal is, of course, not to be exhaustive nor representative—given the huge and rapidly expanding field—but to spark the reader's curiosity and show how certain tasks are remarkably easy in JS. In particular, we will cover the following tasks:

- **Compile C/C++ code to JS using Emscripten**. In his article, Alon Zakai reviews the performance of asm.js, WebAssembly, and Emscripten. He also demonstrates how to use the latter to compile C++ code implementing the Ising model into JS code, without having to read or write one line of JS.
- **Perform symbolic math**. Jos de Jong and Eric Mansfield provide a tutorial for math.js, an extensible library for symbolic computations implemented in JS. Math.js comes with a notepad that is like a lightweight version of the Mathematica Notebook or the Jupyter Notebook, except that it runs completely in the user's browser.
- **Program the GPU**. Fazli Sapuan, Matthew Saw, and Eugene Cheah contributed a tutorial for gpu.js, a library that provides a programming interface to the GPU. Very much like CUDA and OpenCL, in gpu.js, one develops a kernel program using JS that is compiled in real time and deployed to the multiple threads/cores of the GPU. Because this is all done in JS within the browser, it allows for real GPU programming without the need to install any third-party development tools.
- **Experiment with a camera and machine vision**. In my paper, I demonstrate how to write code using JS ES6 to access the computer web camera, process the frames to detect movement, and use the observed movements to control a simulated 3D robotic arm. For the latter, one can use three.js, a powerful library that can take advantage of the computer GPU to build and render complex 3D objects such as textures and lighting.
- **Create complex interfaces**. In the final paper, Ibrahim Tanyalcin, Carla Al Assaf, Julien Ferte, François Ancien, Taushif Khan, Guillaume Smits, Marianne Rooman, and Wim Vranken provide an introduction to Mutaframe, an extensible framework for the visualization of DNA sequences and mutations.

I hope you are convinced that JS is a serious, powerful, and fast language. And more importantly, that JS is an indispensable language for building complex and modern interfaces (to new and existing code) that can run in the user's browser without the need to install third-party tools. Too often, excellent scientific code is hidden behind arcane user interfaces and long chains of dependencies. Making the code interactive and accessible on the web should be considered a priority for scientists.

ABOUT THE AUTHOR

Massimo DiPierro is a professor in the School of Computing at DePaul University and co-director of the MS program in computational finance. Contact him at massimo.dipierro@depaul.edu.