Chapter 1: JavaScript and Politics

> *We build our computer systems the way we build our cities: over time, without a plan, on top of ruins.* – Ellen Ullman, Author and Programmer

The most important programming languages are the ones that manage to capitalize on emerging frontiers in computing. C rode the rise of operating systems, Lisp rode the rise of artificial intelligence research and Java rode the rise of business logic applications. But which caused which? Did C become popular because Unix did? Or was Unix so explosive because C was so ground-breaking? Like many questions relating to human behavior, it is difficult to tease out causation from correlation. And ultimately these questions are not that interesting. Any programmer worth his paycheck can tell you that C and Unix are hopelessly intertwined: C was designed to write Unix and Unix was designed to be written in C. Influential tools and their defining creations are often this way. The tool and the creation frame problems through the same lens. And because they happen to provide the right lens at the right time, their fates become tangled.

There are 2.5 billion Internet users in the world and hundreds of millions of websites. Both of those figures have experienced enormous growth since JavaScript was introduced in 1995. Like C is to Unix, JavaScript is to the Web. More pages, more content and more Internet users have meant more problems JavaScript was designed to solve. What has been good for the web has been good for JavaScript; and what has been good for JavaScript has been good for the Web. The last decade has been extremely good to both.

Although it is difficult to quantify how many lines of JavaScript are written or how many people are writing them, JavaScript consistently ranks near the top of modern programming language popularity metrics.[1] In fact, JavaScript has been called the world's most popular programming language. More important however, than how many people are writing JavaScript, is how many people are running JavaScript programs. And the answer is: nearly everyone. In 2013, the number of mobile Internet devices in use is expected to exceed the number of people in the world.[2] Many of these new devices will run an operating system written in C, many will support applications written in Java, but for all intents and purposes, *all* of these devices will ship with a JavaScript interpreter. In the browser, JavaScript is king. And on the personal device, the browser is increasingly king.

Yet strangely, for all its popularity, JavaScript is surrounded by controversy and conflict. Some JavaScript knowledge is required to write even basic content for the Web, meaning that many people's first experience with programming is in JavaScript and a massive amount of JavaScript code is written each day by complete novices. At the same time, JavaScript experts are pushing the language far beyond what people thought was possible. It is the language of amateur coders *and* professional software engineers. JavaScript is at home in the copy and paste culture of high-school blogs, and in the highest levels of academic Computer Science. JavaScript is ubiquitous and popular. Yet it is chastised and criticized. Is JavaScript a toy language? Or is it the language of the future?

Even five years ago, the answers to these questions were obvious: JavaScript was a toy language on a toy platform. The Web was built for viewing pages of static content. HTML is an excellent tool for describing documents; HTTP is an excellent protocol for fetching those documents. But the Web was not built for interacting with data and performing heavy computations. Today, however, the answers are murkier. Clever engineers have pushed the Web towards loftier goals. We now ask HTML to describe interactive tools and HTTP to manage the input and output of complex programs. JavaScript, a language designed to register browser events and tie together HTML components, has been pushed the furtherest: we ask it to define the behavior of increasingly complex

1 http://redmonk.com/sograd y/2013/02/28/language-rankings-1-13/

2 http://www.theguardian.com/technology/2013/feb/07/mobile-internet-outnumber-people

systems.

The Web today is not just a series of documents strung together with links. The Web is a software platform—perhaps even the beginnings of a distributed operating system. Companies are developing real software for the Web, and Web technologies are creeping into other platforms. Programmers are writing JavaScript as if it was C++ and surprisingly, though admittedly with the help of some clever hacks, the language has largely stood the tests of these new responsibilities. As it turns out, the core of the language is not a toy.

The story of how JavaScript ended up in this situation is strange. Although perhaps we got lucky, and (arguably) the language is not as bad as was thought, why did we choose to rely on JavaScript in the first place? Why take a chance on the construction of the most culturally central pieces of technologies we have made? The answer is that we did not choose to use JavaScript. The Web was not built or decided by anyone; it was built by everyone. The series of events that lead to JavaScript's current position within software development communities was truly a series of accidents. Political forces with a myriad of agendas have shaped the history, community and syntax of the language. Only looking back can we see why JavaScript has risen to its position. JavaScript, the most important language of the modern era, is the accidental king.

The future of the Web and the future of JavaScript are tangled. And for millions of web developers—as well as for companies like Google, Amazon and Microsoft—the question of what tools will be used for web application development in the coming years could not be more important. Some have argued that Web technologies need to be replaced. JavaScript (and maybe even HTML or HTTP) should be thrown out and we should start from scratch with a plan. Others have argued JavaScript is the right language to move forward with, but it needs to be fixed in a big way. Others see no problem with the current trajectory of the Web; maybe we should accept that the path forward for these technologies is difficult to predict or control. It is not clear which argument holds is right, or how the voices of will shape the language or its platform. What is clear is that the possibilities of the Web as a software platform rest on on the future of JavaScript.