EloquentJavaScript01.Rmd

Robert A. Stevens

July 4, 2014

Eloquent JavaScript: An opinionated guide to programming

Chapter 1: Introduction

by Marijn Haverbeke

http://eloquentjavascript.net/Eloquent%20JavaScript.pdf

When personal computers were first introduced, most of them came equipped with a simple programming language, usually a variant of BASIC. Interacting with the computer was closely integrated with this language, and thus every computer-user, whether he wanted to or not, would get a taste of it. Now that computers have become plentiful and cheap, typical users don't get much further than clicking things with a mouse. For most people, this works very well. But for those of us with a natural inclination towards technological tinkering, the removal of programming from every-day computer use presents something of a barrier.

Fortunately, as an effect of developments in the World Wide Web, it so happens that every computer equipped with a modern web-browser also has an environment for programming JavaScript. In today's spirit of not bothering the user with technical details, it is kept well hidden, but a web-page can make it accessible, and use it as a platform for learning to program.

That is what this (hyper-)book tries to do.

> I do not enlighten those who are not eager to learn, nor arouse those who are not anxious to give an explanation themselves. If I have presented one corner of the square and they cannot come back to me with the other three, I should not go over the points again.
>
> Confucius

Besides explaining JavaScript, this book tries to be an introduction to the basic principles of programming. Programming, it turns out, is hard. The fundamental rules are, most of the time, simple and clear. But programs, while built on top of these basic rules, tend to become complex enough to introduce their own rules, their own complexity. Because of this, programming is rarely simple or predictable. As Donald Knuth, who is something of a founding father of the field, says, it is an art.

To get something out of this book, more than just passive reading is required. Try to stay sharp, make an effort to solve the exercises, and only continue on when you are reasonably sure you understand the material that came before.

> The computer programmer is a creator of universes for which he alone is responsible. Universes of virtually unlimited complexity can be created in the form of computer programs.
>
> Joseph Weizenbaum, Computer Power and Human Reason

A program is many things. It is a piece of text typed by a programmer, it is the directing force that makes the computer do what it does, it is data in the computer's memory, yet it controls the actions performed on this same memory. Analogies that try to compare programs to objects we are familiar with tend to fall short, but a superficially fitting one is that of a machine. The gears of a mechanical watch fit together ingeniously, and if the watchmaker was any good, it will accurately show the time for many years. The elements of a program fit together in a similar way, and if the programmer knows what he is doing, the program will run without crashing.

A computer is a machine built to act as a host for these immaterial machines. Computers themselves can only do stupidly straightforward things. The reason they are so useful is that they do these things at an incredibly high speed. A program can, by ingeniously combining many of these simple actions, do very complicated things.

To some of us, writing computer programs is a fascinating game. A program is a building of thought. It is costless to build, weightless, growing easily under our typing hands. If we get carried away, its size and complexity will grow out of control, confusing even the one who created it. This is the main problem of programming. It is why so much of today's software tends to crash, fail, screw up.

When a program works, it is beautiful. The art of programming is the skill of controlling complexity. The great program is subdued, made simple in its complexity.

Today, many programmers believe that this complexity is best managed by using only a small set of well-understood techniques in their programs. They have composed strict rules about the form programs should have, and the more zealous among them will denounce those who break these rules as bad programmers.

What hostility to the richness of programming! To try to reduce it to something straightforward and predictable, to place a taboo on all the weird and beautiful programs. The landscape of programming techniques is enormous, fascinating in its diversity, still largely unexplored. It is certainly littered with traps and snares, luring the inexperienced programmer into all kinds of horrible mistakes, but that only means you should proceed with caution, keep your wits about you. As you learn, there will always be new challenges, new territory to explore. The programmer who refuses to keep exploring will surely stagnate, forget his joy, lose the will to program (and become a manager).

As far as I am concerned, the definite criterion for a program is whether it is correct. Efficiency, clarity, and size are also important, but how to balance these against each other is always a matter of judgement, a judgement that each programmer must make for himself. Rules of thumb are useful, but one should never be afraid to break them.

In the beginning, at the birth of computing, there were no programming languages. Programs looked something like this:

```
00110001 00000000 00000000
00110001 00000001 00000001
00110011 00000001 00000010
01010001 00001011 00000010
00100010 00000010 00001000
01000011 00000001 00000000
01000001 00000001 00000001
00010000 00000010 00000000
01100010 00000000 00000000
```