The World's Most Misunderstood Programming Language

JavaScript, aka Mocha, aka LiveScript, aka JScript, aka ECMAScript, is one of the world's most popular programming languages. Virtually every personal computer in the world has at least one JavaScript interpreter installed on it and in active use. JavaScript's popularity is due entirely to its role as the scripting language of the WWW.

Despite its popularity, few know that JavaScript is a very nice dynamic object-oriented general-purpose programming language. How can this be a secret? Why is this language so misunderstood?

The Name

The *Java-* prefix suggests that JavaScript is somehow related to Java, that it is a subset or less capable version of Java. It seems that the name was intentionally selected to create confusion, and from confusion comes misunderstanding. JavaScript is not interpreted Java. Java is interpreted Java. JavaScript is a different language.

JavaScript has a syntactic similarity to Java, much as Java has to C. But it is no more a subset of Java than Java is a subset of C. It is better than Java in the applications that Java (fka Oak) was originally intended for.

JavaScript was not developed at Sun Microsystems, the home of Java. JavaScript was developed at Netscape. It was originally called LiveScript, but that name wasn't confusing enough.

The *-Script* suffix suggests that it is not a real programming language, that a scripting language is less than a programming language. But it is really a matter of specialization. Compared to C, JavaScript trades performance for expressive power and dynamism.

Lisp in C's Clothing

JavaScript's C-like syntax, including curly braces and the clunky `for` statement, makes it appear to be an ordinary procedural language. This is misleading because JavaScript has more in common with functional languages like Lisp or Scheme than with C or Java. It has arrays instead of lists and objects instead of property lists. Functions are first class. It has closures. You get lambdas without having to balance all those parens.

Typecasting

JavaScript was designed to run in Netscape Navigator. Its success there led to it becoming standard equipment in virtually all web browsers. This has resulted in typecasting. JavaScript is the George Reeves of programming languages. JavaScript is well suited to a large class of non-Web-related applications

Moving Target

The first versions of JavaScript were quite weak. They lacked exception handling, inner functions, and inheritance. In its present form, it is now a complete object-oriented programming language. But many opinions of the language are based on its immature forms.

The ECMA committee that has stewardship over the language is developing extensions which, while well intentioned, will aggravate one of the language's biggest problems: There are already too many versions. This creates confusion.

Design Errors

No programming language is perfect. JavaScript has its share of design errors, such as the overloading of + to mean both addition and concatenation with type coercion, and the error-prone `with` statement should be avoided. The reserved word policies are much too strict. Semicolon insertion was a huge mistake, as was the notation for literal regular expressions. These mistakes have led to programming errors, and called the design of the language as a whole into question. Fortunately, many of these problems can be mitigated with a good lint program.

The design of the language on the whole is quite sound. Surprisingly, the ECMAScript committee does not appear to be interested in correcting these problems. Perhaps they are more interested in making new ones.

Lousy Implementations

Some of the earlier implementations of JavaScript were quite buggy. This reflected badly on the language. Compounding that, those implementations were embedded in horribly buggy web browsers.

Bad Books

Nearly all of the books about JavaScript are quite awful. They contain errors, poor examples, and promote bad practices. Important features of the language are often explained poorly, or left out entirely. I have reviewed dozens of JavaScript books, and **I can only recommend one**: *JavaScript: The Definitive Guide (5th Edition)* by David Flanagan. (Attention authors: If you have written a good one, please send me a review copy.)

Substandard Standard

The official specification for the language is published by ECMA. The specification is of extremely poor quality. It is difficult to read and very difficult to understand. This has been a contributor to the Bad Book problem because authors have been unable to use the standard document to improve their own understanding of the language. ECMA and the TC39 committee should be deeply embarrassed.

Amateurs

Most of the people writing in JavaScript are not programmers. They lack the training and discipline to write

good programs. JavaScript has so much expressive power that they are able to do useful things in it, anyway. This has given JavaScript a reputation of being strictly for the amateurs, that it is not suitable for professional programming. This is simply not the case.

Object-Oriented

Is JavaScript object-oriented? It has objects which can contain data and methods that act upon that data. Objects can contain other objects. It does not have classes, but it does have constructors which do what classes do, including acting as containers for class variables and methods. It does not have class-oriented inheritance, but it does have prototype-oriented inheritance.

The two main ways of building up object systems are by inheritance (is-a) and by aggregation (has-a). JavaScript does both, but its dynamic nature allows it to excel at aggregation.

Some argue that JavaScript is not truly object oriented because it does not provide information hiding. That is, objects cannot have private variables and private methods: All members are public.

But it turns out that JavaScript objects *can* have private variables and private methods. (Click here now to find out how.) Of course, few understand this because JavaScript is the world's most misunderstood programming language.

Some argue that JavaScript is not truly object oriented because it does not provide inheritance. But it turns out that JavaScript supports not only classical inheritance, but other code reuse patterns as well.