λ How to implement a programming language in JavaScript

This is a tutorial on how to implement a programming language. If you ever wrote an interpreter or a compiler, then there is probably nothing new for you here. But, if you're using regexps to "parse" anything that looks like a programming language, then please read at least the section on parsing. Let's write less buggy code!

The ToC on the right is in "simple-to-advanced" order. I'd recommend you not to skip forward, unless you know the subject well. You can always refer back if you don't understand something. Also, questions and feedback are very much appreciated!

The target audience is the average JavaScript / NodeJS programmer.

What are we going to learn

- What is a parser, and how to write one.
- How to write an interpreter.
- Continuations, and why are they important.
- Writing a compiler.
- How to transform code to continuation-passing style.
- A few basic optimization techniques.
- Examples of what our λanguage brings new over plain JavaScript.

In between, I'm going to argue why Lisp is a great programming language. **However, the language we will work on is *not* a Lisp**. It has a richer syntax (classical infix notation that everybody knows) and will be about as powerful as Scheme, except for macros. Sadly or not, macros are the ultimate bastion of Lisp, something that other languages just can't conquer (unless they are called *Lisp dialects*). [Yes, I know about SweetJS… close but no cigar.]

But first, let's dream up a programming language.

How to implement a PL