The #12in23 Challenge - Common Lisp - Trying Common Lisp in January 2023

Written by Seth Corker on Benevolent Bytes

Cover photo by Milad Fakurian on Unsplash

Why Common Lisp?

Lisp is a family of languages that I've always heard about, but never really taken the next step. It's a technology that's influenced so much and yet hasn't really been in the mainstream. Maybe you've heard about it in Beating the Averages by Paul Graham or taken a look at the syntax and been put off — it can appear quite foreign compared with popular programming languages today. I thought this would be the perfect time to experience what Lisp has to offer and figure out if it's something I'd like to use in 2023 and beyond.

I've been exposed to Lisp languages in the past, but mostly though reading and talks. It comes up in the history of computing, but not often in a modern context. I think that over time, I've absorbed enough through repeated exposure to know a little about Lisp, I've just never sat down and used it until now. For the 12 in 23 challenge I chose Common Lisp for January as it's a popular language with numerous resources and the Steel Bank Common Lisp compiler because it's a high performance compiler that runs on most systems, and it receives regular updates.

Initial thoughts and challenges

Syntax

Lisp is scary at first. It has a quite unfamiliar syntax than most languages out there that follow a C-style syntax. C-style languages like JavaScript, C#, Ruby, Python etc. all look different, but the structure is very similar.

A class in JavaScript might look like this:

```
class MyClass {
    saySomething(message) {
        const prefix = "[SAY]"
        console.info(`${prefix}: ${message}`)
    }
}
```

The same in Ruby might look like this:

```
class MyClass
    def say_something(message)
        prefix = '[SAY]'
        puts "#{prefix}: #{message}"
    end
end
```

There are differences in how the body of a class for function are defined, the convention for naming is camel case vs. snake case but overall, the structure is the same. Lisp doesn't look like this at all.

This snippet does the same things as before (it's a contrived example, there's no need for a class in any of these code snippets, but I wanted to show the equivalent concepts).

```
(defclass my-class () ())

(defmethod say-something ((obj my-class) &key message)
  (let ((prefix "[SAY]"))
  (format t "~a: ~a~&" prefix message)))
```

What Lisp mostly breaks down to is an operator, operands and *lots* of parentheses.

Here are a few examples: Something like `(+ 1 2 3)` is the same as `1 + 2 + 3` in a C-style language.

Defining a function In Lisp, it looks like this: `(defun hello-world () "Hello World")` The operator is `defun` followed by the operands `hello-world`, `()`, `"Hello World"` which are the name, arguments, and body of the function.

Versus the equivalent in JavaScript:

```
function helloWorld() {
    return "Hello World"
}
```

Conditionals In Lisp, it looks like this: `(if t "true" "false")`

Versus the equivalent in JavaScript:

```
if (true) {
    return "true"
} else {
    return "false"
}
```

The challenge with syntax in Lisp is that it takes some time to get used to. It's consistent, but it's a big shift for the languages I've used in the past. One major takeaway I've had from using Common Lisp for the past month is that creating plenty of small functions makes it more readable. That might sound obvious, but the number of parenthesis makes it difficult to delineate expressions easily.

Finding resources

The first week was challenging in learning about the ecosystem. When you learn a new language, it's not just the language itself but the conventions, where good resources are, how to use the documentation, etc. As part of the initial session, I've learned more about what exists and doesn't as part of the core language. Common Lisp has a small set of functions, which I'm slowly working my way through.

The big challenge for learning right now is that documentation can be challenging to use, Lisp Works is a common resource, but it's not easy to navigator understand for a beginner. The best part of the documentation is the examples, as they illuminate a lot and the rest can be inferred. I'm sure I'll get used to it as I learn more along the way.

Interesting language features

Consistency

The syntax is boring. I've noticed that most things are straightforward and easy to understand, and it's not the syntax that really changes. The only challenge is figuring out what arguments are needed in macros and the shape of data being passed in. There's not much to it, once you know a small amount you have all the tools you need to muddle your way through.

Along these same lines, documentation defined in code is done elegantly, simply add a string after the args for a function, and then it can be accessed using `(documentation 'my-function-name 'function)`. This is something I've seen in numerous modern languages — most likely inspired from Lisp.

REPL

I use the REPL for other languages. It's great for trying things out in Ruby and Elixir, but each real has varying degrees of usefulness. Ruby's IRB is great for prototyping or testing functions manually, but that's about it. Elixir's IEx has the ability to recompile modules and change the system while it's running, which makes it more useful. Common Lisp takes this even further and allows for redefining functions. The REPL becomes invaluable, it's used to prototype, and you can adjust anything and update it while the system is running. This feels mod like interacting with an environment than a simple CLI used for evaluating simple expressions.

Developer experience

Wrestling with IDEs

IDE support is mixed. Visual Studio Code is my editor of choice, but the extensions could be better. They are just very basic. I couldn't quite get formatting working or suggestions that went beyond simple function lists. Everything was just a bit clunky, to the point where even the build in editor on Exercism was better in some ways.

I moved to emacs which I occasionally use but am still a beginner in, it's a big learning curve but seems like the best experience for Lisp. The REPL becomes a companion in the editor, and shortcuts for recompiling functions and evaluating in the REPL make it the most productive choice.

Errors Error messages can be difficult to parse when using SBCL, it's hard to see where an error happened as my experience with reading error messages isn't attuned. I just don't have the experience in the language to know what an error is trying to communicate.

Documentation Documentation is still a struggle. Common-Lisp.net has a good quick start which is useful for getting up and running and gives some context on common tooling in the ecosystem. The Common Lisp Cookbook is a nice, guided walkthrough for teaching Common Lisp, and I want to work my way through it more, but in terms of quick reference, I haven't found anything satisfactory.

Takeaways

I hope to keep using Common Lisp, it's an interesting language that has a long history and has introduced me to an entirely different way of programming. Doing a few exercises has been great, but I really want a project to work on and see how the ecosystem works as a whole. The most difficult challenge I had in the beginning has been eased with more exposure to common tooling and given more experience with SLIME, working with Common Lisp will be a far more productive experience. I can't wait to jump into another language, but I will also try to work my way through more exercises in Lisp and continue learning.

Resources

Here are some resources to help you on your journey that I've found helpful so far. - Common-Lisp.net — Getting started guide with installation instructions and some context on common tooling. - The Common Lisp Cookbook — Online book with examples and explanation on various aspects of the language. - Learn X in Y — A quick cheatsheet/reference for learning though examples - Common Lisp Track on Exercism — Exercises you can do to sharpen your skills and learn while solving problems