

## CHALLENGES

1. There are at least six domain-specific languages used in the [little system I cobbled together](#) to write and publish this book. What are they?
2. Get a “Hello, world!” program written and running in Java. Set up whatever makefiles or IDE projects you need to get it working. If you have a debugger, get comfortable with it and step through your program as it runs.
3. Do the same thing for C. To get some practice with pointers, define a [doubly linked list](#) of heap-allocated strings. Write functions to insert, find, and delete items from it. Test them.

1. Java and C are used to build the two interpreters described in the book, while Dart is used behind the scenes for build and testing scripts that help automate the process. The actual content of the book is written in Markdown, which then gets turned into a website using HTML, with CSS and SCSS handling the styling. All of these count as domain specific languages because each one is designed to handle a specific task really well, rather than being used for everything.

2.

```
1 public class HelloWorld { new *
2     public static void main(String[] args) { new *
3         System.out.println("Hello, world!");
4     }
5 }
```

```
Run HelloWorld x
/opt/homebrew/opt/java/libexec/openjdk.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=6372...
Hello, world!
Process finished with exit code 0
```

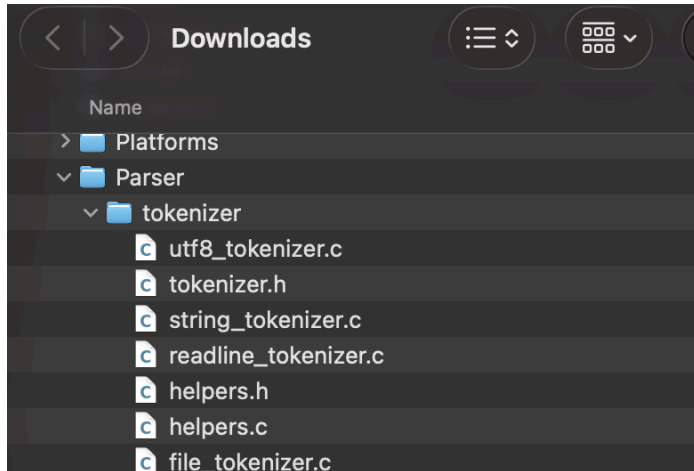
```
valeriaurzua@Valerias-MacBook-Pro craftinginterpreters % cd c
gcc ch1challenge3.c -o ch1challenge3
./ch1challenge3

valeria cs4080
Found:cs4080
valeria
```

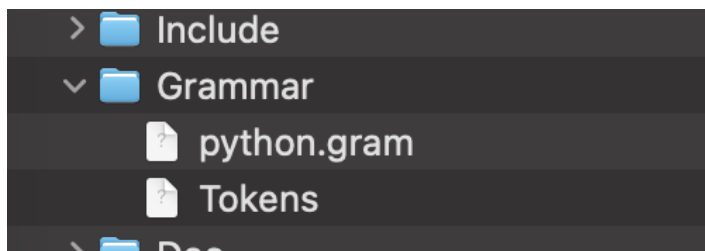
3.

## CHALLENGES

1. Pick an open source implementation of a language you like. Download the source code and poke around in it. Try to find the code that implements the scanner and parser. Are they handwritten, or generated using tools like Lex and Yacc? ( `.l` or `.y` files usually imply the latter.)
2. Just-in-time compilation tends to be the fastest way to implement dynamically typed languages, but not all of them use it. What reasons are there to *not* JIT?
3. Most Lisp implementations that compile to C also contain an interpreter that lets them execute Lisp code on the fly as well. Why?



1.



I downloaded the CPython

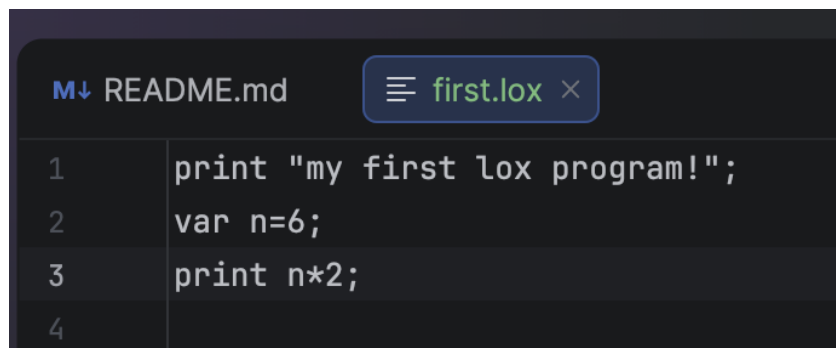
source code from GitHub and looked at the tokenizer and grammar files, the scanner is handwritten in C and the parser is generated from grammar files

2. Even though just-in-time compilation can make things run faster, a lot of dynamically typed languages don't use it because JITs are pretty complicated to build and keep working. Also, they also take up more memory and can slow down startup since the program has to compile while it's running. Also, JITs can make debugging harder and don't always work as smoothly across different machines. Due to this, many languages just stick with simpler interpreters or bytecode virtual machines.
3. Lisp implementations often include both an interpreter and a compiler because interpreters make interactive testing and debugging easier, while compiling to C is better for performance in finished programs.

## CHALLENGES

1. Write some sample Lox programs and run them (you can use the implementations of Lox in [my repository](#)). Try to come up with edge case behavior I didn't specify here. Does it do what you expect? Why or why not?
2. This informal introduction leaves a *lot* unspecified. List several open questions you have about the language's syntax and semantics. What do you think the answers should be?
3. Lox is a pretty tiny language. What features do you think it is missing that would make it annoying to use for real programs? (Aside from the standard library, of course.)

1.

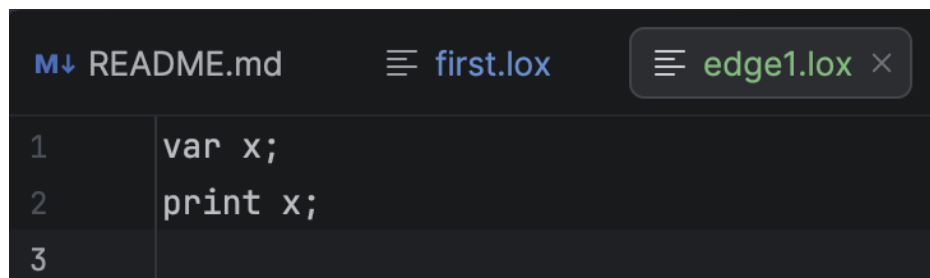


A screenshot of a code editor with a dark theme. At the top, there are two tabs: 'README.md' and 'first.lox'. The 'first.lox' tab is active. The code in the editor is as follows:

```
1 print "my first lox program!";
2 var n=6;
3 print n*2;
4
```

```
valeriaurzua@Valerias-MacBook-Pro craftinginterpreters % ./jlox loxCh3-tests/first.lox

my first lox program!
12
```



A screenshot of a code editor with a dark theme. At the top, there are three tabs: 'README.md', 'first.lox', and 'edge1.lox'. The 'edge1.lox' tab is active. The code in the editor is as follows:

```
1 var x;
2 print x;
3
```

```
valeriaurzua@Valerias-MacBook-Pro craftinginterpreters % java -cp build/java com.craftinginterpreters.lox.Lox loxCh3-tests/edge1.lox

nil
```

An edge case I tested was declaring a variable without giving it a value and then printing it. I expected it might error, but instead Lox prints nil. This shows that uninitialized variables default to nil, which is helpful but also something programmers need to be aware of so they don't accidentally use variables before assigning them.

2. Question 1: What happens if you declare a variable without initializing it and then use it right away?

Answer: From what I tested, the variable defaults to nil. This makes sense since Lox is dynamically typed and doesn't force every variable to be initialized.

Question 2: How do logical operators handle boolean operands?

Answer: After running a few small programs, it looks like logical operators treat false and nil as falsey values, while everything else is treated as truthy. This also matches the short-circuit behavior described in the chapter.

3. One thing Lox is missing that would make it annoying to use for real programs is user input handling. Right now, programs can print output, but they can't easily take input from the user, which limits what you can actually build. It also doesn't have lists or arrays, so there's no easy way to store or work with collections of values. Without lists or arrays, even basic tasks like looping over data or keeping track of multiple items would be much harder than they need to be.