Official racket documentation: https://docs.racket-lang.org/guide

Big reference was megaparsack: Megaparsack: Practical Parser Combinators (racket-lang.org)
Most of the time spent was just building from this example:
2.2 Parsing sequences (racket-lang.org)

```
(define two-integers/p
    (do [x <- integer/p]
        (many/p space/p #:min
1)
        [y <- integer/p]
        (pure (list x y))))
```

using or/p for multiple paths as needed and used do to chain parsers.
Note: I didn't tokenize the input because most of the examples were just consuming strings and just this line "Parsers that operate on strings, like char/p and integer/p, will not work with tokens from parser-tools/lex because tokens can contain arbitrary data." made it undesirable.

Programming Language Pragmatics by Michael L. Scott
Ch. 2.3 Parsing Figure 2.17 for pseudocode reference

Error handling:
https://docs.racket-lang.org/reference/exns.html#%28def._%28%28quote._~23~25kernel%29._srcloc-line%29%29

Side reference-ish:
ReeceMcMillin/CS441ParserProject: A simple Racket-based parser for a small textbook language. (github.com)

- I found this mid-way through my parser (after building my atoms and expressions) and I liked the way he did the error handling but I didn't like the two indicators on the top and bottom. I previously had the bare minimum for the error message but then I added in the indicator just on the bottom.
- I also had absolutely no idea why he did the gen:custom-write but I also didn't read the documentation on that.
- His parser was a good reference/check to make sure I was using megaparsack do, try/p, many/p, and or/p to its full extent.