Week 7 Exercises

FIT2102 Programming Paradigms

Requirements

Complete all the exercises as per the instructions found in the code files. All tests must pass. There must be no compilation warnings or errors.

Recommended order:

- `Parser.hs`
- `Pretty.hs`

Parsing function type

The type of these functions are of the form

```
String -> Maybe (String, a)
```

where `a` is the type of the value we are parsing into (also see https://tgdwyer.github.io/haskell2/#type-parameters-and-polymorphism).

> The order of the tuple `(String, a)` is not obvious at this time, but in later weeks we will see why this is useful.

- The `String` type values represent the input to be parsed, and the remainder of the string, respectively.
- The `Maybe` type is used to represent the possibility of failure in parsing (also see https://tgdwyer.github.io/haskell2/#maybe).

Running Tests

Run `stack test` to run the tests. By default, this will not run tests for the supplementary exercises. To run tests for the supplementary exercises as well, run `stack test --test-arguments supplementary` (or `stack test --ta s` for short).

Web Socket Overview

WebSockets is a communication protocol that allows for persistent, real-time, full-duplex connections between a client (like a browser) and a server. Unlike traditional HTTP (where the client requests data and the server responds), WebSockets keep the connection open, allowing both the client and server to send messages to each other at any time.

GET and POST are common HTTP methods used to request data (GET) or submit data (POST) in traditional web communication.

Over the next few weeks, we will gradually develop a parser to handle WebSocket requests, ultimately building a server-client application using WebSockets. This project will culminate in creating a Wordle game, with the front end (RxJS/JavaScript) communicating over WebSockets (using Haskell) to some server-side logic also written in Haskell.