Week 8 Exercises

FIT2102 Programming Paradigms

- Requirements
- Type holes

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:

1. `Maybe.hs`
2. `ApplicativeFunctions.hs`
3. `Parser.hs`

Finding useful functions

Haskell comes with a very powerful in built library named 'Prelude', which has a whole bunch of useful functions. Some functions in the Prelude you will have to implement over the coming weeks.

However, it is also very useful to know how to explore the functions already use them to create some very useful functions!

There are a couple of standard approaches that we will be recommending in this unit. Have a try using the following techniques and tools, and if you get stuck, please make sure to let your tutor know or make an Ed post.

Type holes

A useful GHC feature that will help you with this is type holes which allow the compiler to tell you the expected type at a location, and suggestions

Suppose we have identified the `even` function. We can then use a type hole in the implementation to find the type of function we are trying to find. What this looks like in GHCi:

```
*Main Instances JSON Parser Paths_exercises
ghci> (_ even [1]) :: Bool
```

```
<interactive>:30:2: error:
    • Found hole: _ :: (Integer -> Bool) -> [Integer] -> Bool
    • In the expression: _
      In the expression: (_ even [1]) :: Bool
      In an equation for ‘it’: it = (_ even [1]) :: Bool
    • Relevant bindings include
        it :: Bool (bound at <interactive>:30:1)
      Valid hole fits include
        all :: forall (t :: * -> *) a.
               Foldable t =>
               (a -> Bool) -> t a -> Bool
          with all @[] @Integer
        any :: forall (t :: * -> *) a.
               Foldable t =>
               (a -> Bool) -> t a -> Bool
          with any @[] @Integer
```

Importantly, the compiler has told us `Found hole: _ :: (Integer -> Bool) -> [Integer] -> Bool` meaning that we want a function of the type `(Integer -> Bool) -> [Integer] -> Bool`.

Then, there are the suggested functions `all` and `any`.

Try putting type holes in your code implementations as well! Your HLS and compiler should pick up the type holes and give similar suggestions and type hints.