Workshop exercises set 5.

QUESTION 1
Recall the discussion of the Maybe monad in lectures, and the definitions
of maybe_head, maybe_sqrt and maybe_sqrt_of_head. In a similar style, write
Haskell code for the function

```
        maybe_tail :: [a] -> Maybe [a]
```

which returns the tail of a list if the list is not empty, and

```
        maybe_drop :: Int -> [a] -> Maybe [a]
```

which is like the prelude function drop ("drop n xs" drops the first n elements
of the list xs), but returns a Maybe type. If n is greater than the length
of xs, it should return Nothing (drop returns [] in this case), otherwise
it should return Just the resulting list.

Code two versions of maybe_drop. Both should use maybe_tail. One should
explicitly check for Nothing and the other should use >>=.

QUESTION 2
Given the tree data type defined below, write the Haskell function

```
    print_tree :: Show a => Tree a -> IO ()
```

which does an inorder traversal the tree, printing the contents of each node
on a separate line. What are the advantages and disadvantages of this approach
compared to traversing the tree and returning a string, and then printing
the string?

```
>data Tree a = Empty | Node (Tree a) a (Tree a)
```

QUESTION 3
Write a Haskell function

```
     str_to_num :: String -> Maybe Int
```

that converts a string containing nothing but digits to Just the number they
represent, and any other string to Nothing. Hint: the standard library module
Data.Char has a function isDigit that tests whether a character is a decimal
digit, and another function digitToInt that converts such characters to a
number between between 0 and 9.

QUESTION 4
Write two versions of a Haskell function that reads in a list of lines
containing numbers, and returns their sum. The function should read in lines
until it finds one that contains something other than a number.

The first version of the function should sum up the numbers as it read them in.
The second should collect the entire list of numbers before it starts summing
them up.