1 List Prefix

- Write a recursive function \texttt{begin xs ys} that returns true if \texttt{xs} is a prefix of \texttt{ys}. Both lists are lists of integers. Include the type signature.

\begin{verbatim}
> begin [] []
True
> begin [1] []
False
> begin [1,2] [1,2,3,4]
True
> begin [1,2] [1,1,2,3,4]
False
> begin [1,2,3,4] [1,2]
\end{verbatim}

2 List Containment

- Write a recursive function \texttt{subsequence xs ys} that returns true if \texttt{xs} occurs anywhere within \texttt{ys}. Both lists are lists of integers. Include the type signature.

- Hint: reuse \texttt{begin} from the previous exercise.

\begin{verbatim}
> subsequence [] []
True
> subsequence [1] []
False
> subsequence [1] [0,1,0]
True
> subsequence [1,2,3] [0,1,0,1,2,3,5]
True
\end{verbatim}
3 Mystery

- Consider the following function:

\[
\text{mystery} :: [a] \rightarrow [[[a]]]
\]

\[
\text{mystery} [] = [[[\ ]]]
\]

\[
\text{mystery} (x:xs) = \text{sets} ++ (\text{map } (x:) \text{ sets})
\]

where \( \text{sets} = \text{mystery} \text{ xs} \)

- What would \( \text{mystery} [1,2] \) return? \( \text{mystery} [1,2,3] \)?
- What does the function compute?

4 foldr

- Explain what the following expressions involving \texttt{foldr} do:

\[
1. \text{foldr } (:) \ [\ ] \ xs
2. \text{foldr } (:) \ xs \ ys
3. \text{foldr } (y \ ys \rightarrow \ ys ++ [y]) \ [\ ] \ xs
\]

5 shorter

- Define a function \texttt{shorter} \( \text{xs} \ \text{ys} \) that returns the shorter of two lists.

\[
> \text{shorter} \ [1,2] \ [1]
[1]
> \text{shorter} \ [1,2] \ [1,2,3]
[1,2]
\]

6 stripEmpty

- Write function \texttt{stripEmpty} \( \text{xs} \) that removes all empty strings from \( \text{xs} \), a list of strings.

\[
> \text{stripEmpty} \ ["", \ "Hello", \ ",", \ "", \ "World!""]
["Hello",\ "World!"]
> \text{stripEmpty} \ ["""]
[
]
> \text{stripEmpty} \ [\ ]
[
]
\]

7 merge

- Write function \texttt{merge} \( \text{xs} \ \text{ys} \) that takes two ordered lists \( \text{xs} \) and \( \text{ys} \) and returns an ordered list containing the elements from \( \text{xs} \) and \( \text{ys} \), without duplicates.

\[
> \text{merge} \ [1,2] \ [3,4]
[1,2,3,4]
> \text{merge} \ [1,2,3] \ [3,4]
[1,2,3,4]
> \text{merge} \ [1,2] \ [1,2,4]
[1,2,4]
\]

2
8 Data Types

- Consider the following type:

  ```haskell
data Shape = Circle Float |
              Rectangle Float Float
  ```

- Define a function `shapeLength` that computes the length of the perimeter of a shape.

- Add an extra constructor to `Shape` for triangles.

- Define a function which decides whether a shape is regular: a circle is regular, a square is a regular rectangular, and being equilateral makes a triangle regular.

9 Function Composition

- Rewrite the expression

  ```haskell
  map f (map g xs)
  ```

  so that only a single call to map is used

10 Reduce

- Let the Haskell function `reduce` be defined by

  ```haskell
  reduce f []     v = v
  reduce f (x:xs) v = f x (reduce f xs v)
  ```

- Reconstruct the Haskell functions `length`, `append`, `filter`, and `map` using `reduce`. More precisely, complete the following schemata (in the simplest possible way):

  ```haskell
  mylength xs   = reduce ___ xs ___
  myappend xs ys = reduce ___ xs ___
  myfilter p xs = reduce ___ xs ___
  mymap f xs    = reduce ___ xs ___
  ```