Part 1.
Recall from class that we discussed the enumerated types (e.g. Bool, Char, and Day). The data
type keyword can be used to define a new enumerated data type where the constructors take no
arguments.

    data Day = Sunday | Monday | Tuesday | Wednesday | Thursday | Friday | Saturday
    deriving (Eq, Show, Ord, Enum)

Type classes can cut down on some of the routine work needed to define a new data type. For
instance in the enumerated data type Day, by deriving from the Eq type class we can define
what it means for elements of this type to be equal. Similarly, by deriving from Enum we get the
default enumeration of the data type (Sunday = 0, Monday = 1, …)

For this assignment you will:
1. Use Haskell to define a data type for Month in the standard ordering
   (January...December).
2. Write a function daysInMonth which has the following type
   daysInMonth :: (Month -> Integer)
   which returns the number of days in the particular month passed in as an argument.
3. Write the functions nextMonth :: Month -> Month and lastMonth:: Month -> Month which
   return the next month and previous month, respectively.

Part 2.
Prove by extensionality the following properties of composition. Recall that composition of two
functions is defined as:

    (f ° g) x = f(g x)

Which is read f composed with g applied to a value x is defined as f applied to g applied to x.
And recall the identity function given in class

id x = x

The identity function has the type:

    id :: a → a

1. Prove ∀f : A → B,f°id = f
2. Prove ∀f : A → B,id°f = f
3. For each proof, what is the resulting type of the identity function.

Part 3.
Infer the types of each of the functions. For instance, given
\[ f \ x \ y \ z = \text{if } x \text{ then } y \text{ else } z \]

We can infer that \( x :: \text{Bool} \), and \( y \) and \( z \) have the same type (because a function can only return an element of one type.) So the function \( f \) has the type:

\[ f :: \text{Bool} \rightarrow a \rightarrow a \rightarrow a \]

In your derivations you can specify the built in type (if it can be inferred) or use a type variable different than the argument instance variable name (e.g. \( a, b, c, \ldots \))

1. \( f [] = [] \)
   \[ f (x:xs) = [x] ++ f \ xs \]
2. \( f \ x \ y = x \mod y \)
3. \( f \ x \ y \ (z:zs) = (g \ x \ y \ z) : (f \ x \ y \ zs) \)
   here \( g \) is some function previously defined and has the type: \( g :: a \rightarrow b \rightarrow c \rightarrow d \)
4. \( f \ x \ y \)
   \[ | y == 0 = \text{Error “No division by zero”} \]
   \[ | y > 0 = (x / y) \]