1. What are the types of the following \( x_1, x_2, \ldots, x_5 \)? Some might have type errors:
\[
b = \text{True}
\]
\[
x_1 = \text{if } b \text{ then putStrLn "hi" else return ()}
\]
\[
x_2 = \text{if } b \text{ then putStrLn "squid" else return "octopus"}
\]
\[
x_3 = \text{if } b \text{ then "squid" else "octopus"}
\]
\[
x_4 = \text{if } b \text{ then "squid" else return ()}
\]
\[
x_5 = \text{do}
\quad \text{putStr "testing"}
\quad x \leftarrow \text{readLn}
\quad \text{return (not x)}
\]

2. Give a recursive definition of a list \( \text{doubles} \) whose first element is 10, and whose \( n \) th element is twice the \( n-1 \) st, i.e., \([10, 20, 40, 80, 160, 320, \ldots]\). To do this, write a helper function \( \text{doubles_from} \) that takes a parameter \( n \) and returns a list of all the doubles starting at \( n \).

3. Give yet another non-recursive definition of \( \text{doubles} \) using the built-in function \( \text{iterate} \) from the Haskell prelude. This is defined as follows:
\[
\text{iterate} :: (a \to a) \to a \to [a]
\]
\[
\text{iterate } f \ x = x : \text{iterate } f \ (f \ x)
\]
4. Define a Haskell list `dollars` that is the infinite list of amounts of money you have every year, assuming you start with $100 and get paid 5% interest, compounded yearly. (Ignore inflation, deflation, taxes, bailouts, the possibility of total economic collapse, and other such details.) So dollars should be equal to [100.0, 105.0, 110.25, ...]

5. Desugar the following actions:

```haskell
lion = do
  putStrLn "What is the color of your mane?"
  color <- getLine
  putStrLn $ "Rawr, nice " ++ color ++ " mane"

parity_repl = do
  putStrLn "Enter a number"
  n <- readLn
  case odd n of
    True -> putStrLn $ (show n) ++ " is odd"
    False -> putStrLn $ (show n) ++ " is even"
  parity_repl

map_reduce = do
  putStrLn "Enter a unary mapping operation"
  A. op <- getLine
  putStrLn "Enter a unary reducing operation"
  B. reduce <- getLine
  putStrLn "Enter a list to evaluate"
  C. lst <- getLine
  let expr = "foldr1 (" ++ reduce ++ ") $ map (" ++ op ++ ") " ++
      lst
       in evaluate expr
```