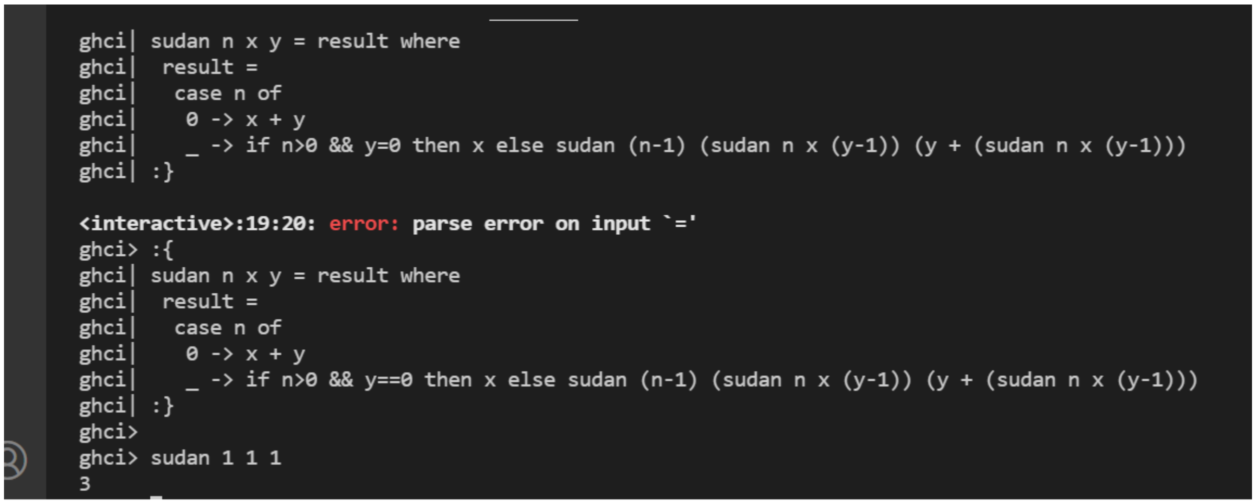
LABORATORY 8

Exercise 8.8.1: Test in GHCi: fact 10000 and fact 100000. What takes longer, calculating the number or printing all of its digits?

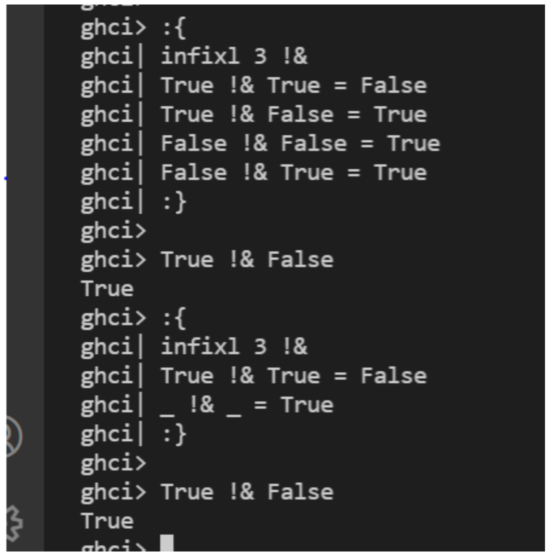
ANSWER: For both cases, it takes longer to print all the digits. When it comes to calculating the number, in the first case the time is very short, while in the second case it takes a couple of seconds.

Exercise 8.14.1: Rewrite the sudan function from lab 1 using where in Haskell.



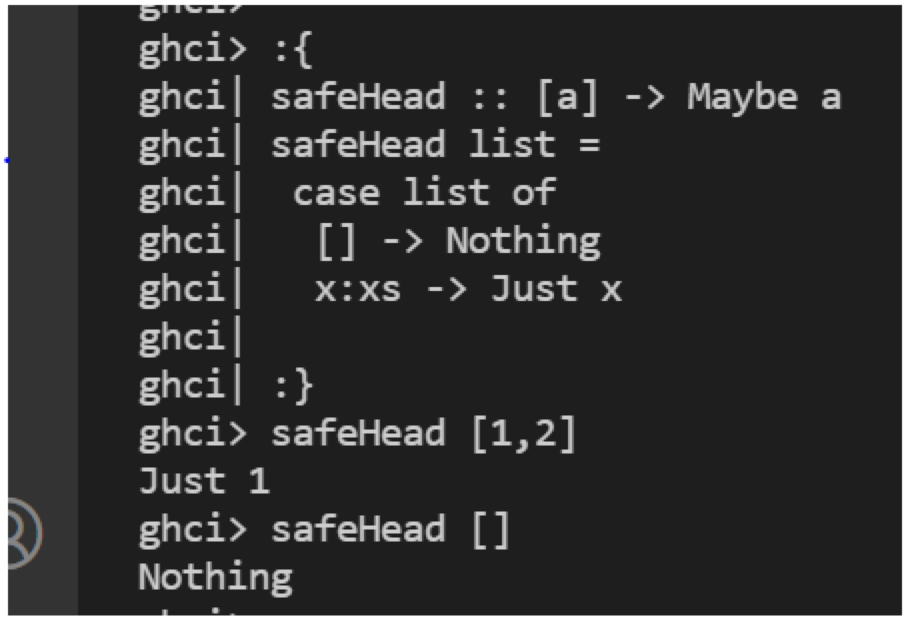
```
ghci| sudan n x y = result where
ghci|  result =
ghci|   case n of
ghci|    0 -> x + y
ghci|    _ -> if n>0 && y=0 then x else sudan (n-1) (sudan n x (y-1)) (y + (sudan n x (y-1)))
ghci| :}

<interactive>:19:20: error: parse error on input `='
ghci> :{
ghci| sudan n x y = result where
ghci|  result =
ghci|   case n of
ghci|    0 -> x + y
ghci|    _ -> if n>0 && y==0 then x else sudan (n-1) (sudan n x (y-1)) (y + (sudan n x (y-1)))
ghci| :}
ghci>
ghci> sudan 1 1 1
3
```

Exercise 8.14.2: Define an infix operator called **!&** in Haskell, which implements the logical function nand (not and).

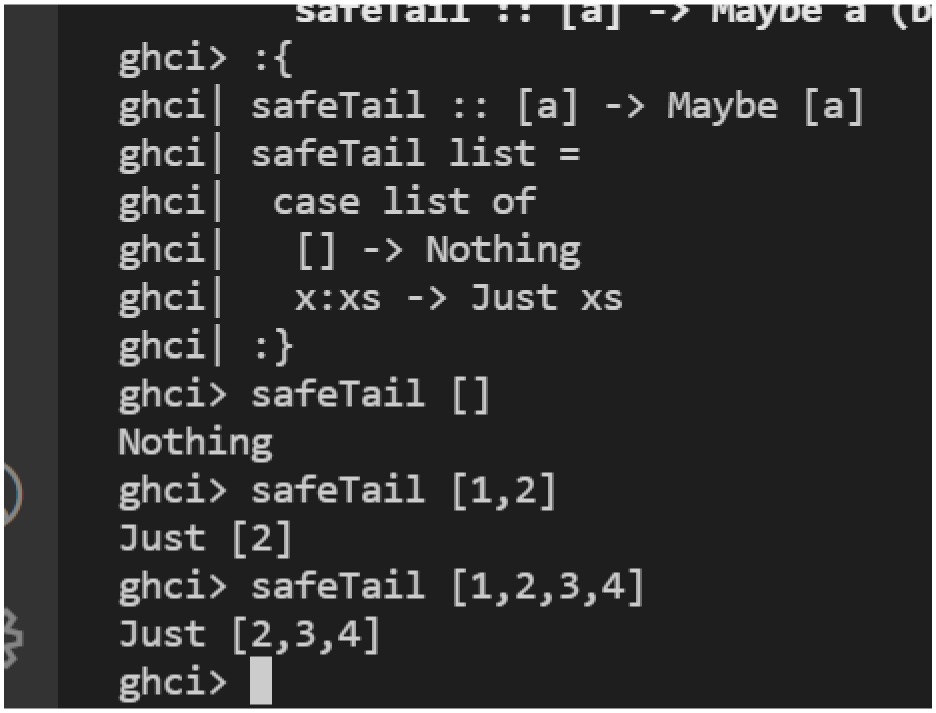


```
ghci> :{
ghci| infixl 3 !&
ghci| True !& True = False
ghci| True !& False = True
ghci| False !& False = True
ghci| False !& True = True
ghci| :}
ghci>
ghci> True !& False
True
ghci> :{
ghci| infixl 3 !&
ghci| True !& True = False
ghci| _ !& _ = True
ghci| :}
ghci>
ghci> True !& False
True
```

Exercise 8.14.3: Implement the safeHead :: [a] -> Maybe a and safeTail :: [a] -> Maybe [a] functions in Haskell which return Nothing if the list is empty and the result wrapped in Just if it isn't empty.

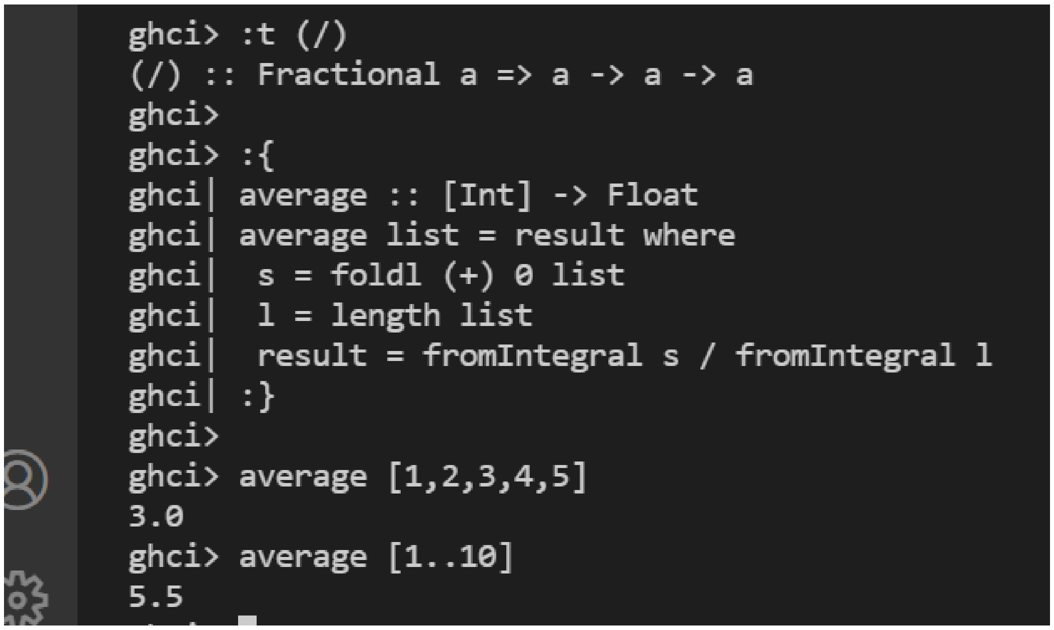


```
ghci> :{
ghci| safeHead :: [a] -> Maybe a
ghci| safeHead list =
ghci|  case list of
ghci|   [] -> Nothing
ghci|   x:xs -> Just x
ghci|
ghci| :}
ghci> safeHead [1,2]
Just 1
ghci> safeHead []
Nothing
```



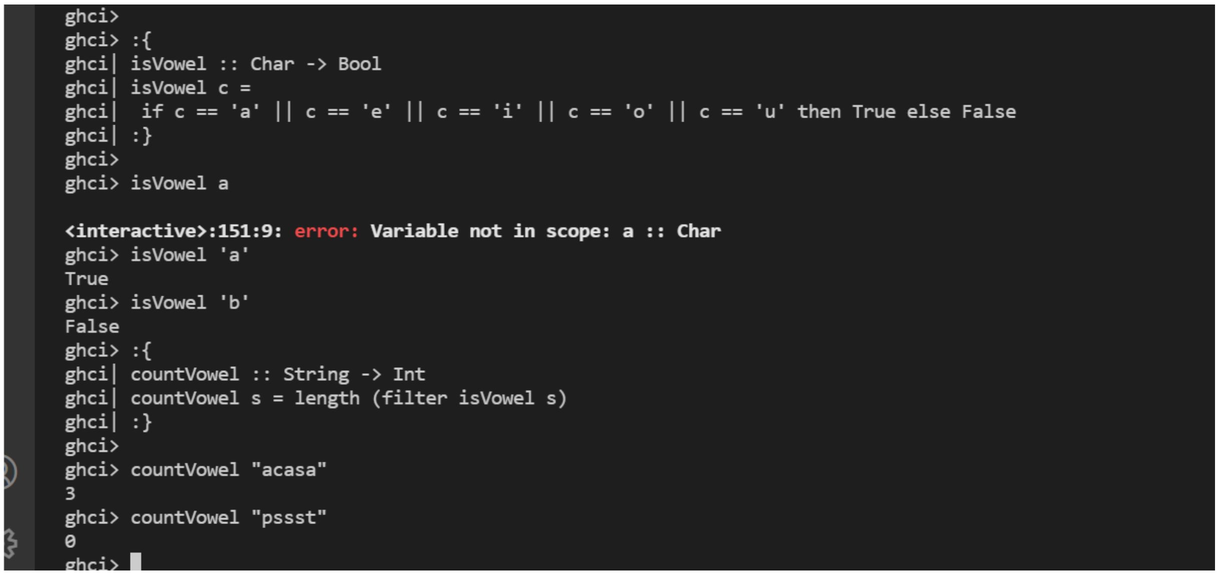
```
          safeTail :: [a] -> Maybe a (b
ghci> :{
ghci| safeTail :: [a] -> Maybe [a]
ghci| safeTail list =
ghci|  case list of
ghci|   [] -> Nothing
ghci|   x:xs -> Just xs
ghci| :}
ghci> safeTail []
Nothing
ghci> safeTail [1,2]
Just [2]
ghci> safeTail [1,2,3,4]
Just [2,3,4]
ghci>
```

Exercise 8.14.4: Write a function called average :: [Int] -> Float , which calculates the average of a list of integers.



```
ghci> :t (/)
(/) :: Fractional a => a -> a -> a
ghci>
ghci> :{
ghci| average :: [Int] -> Float
ghci| average list = result where
ghci|  s = foldl (+) 0 list
ghci|  l = length list
ghci|  result = fromIntegral s / fromIntegral l
ghci| :}
ghci>
ghci> average [1,2,3,4,5]
3.0
ghci> average [1..10]
5.5
```

Exercise 8.14.5: Write a function called countVowels which counts the number of vowels in a string.



```
ghci>
ghci> :{
ghci| isVowel :: Char -> Bool
ghci| isVowel c =
ghci|  if c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u' then True else False
ghci| :}
ghci>
ghci> isVowel a

<interactive>:151:9: error: Variable not in scope: a :: Char
ghci> isVowel 'a'
True
ghci> isVowel 'b'
False
ghci> :{
ghci| countVowel :: String -> Int
ghci| countVowel s = length (filter isVowel s)
ghci| :}
ghci>
ghci> countVowel "acasa"
3
ghci> countVowel "pssst"
0
ghci>
```