INTRODUCTION TO

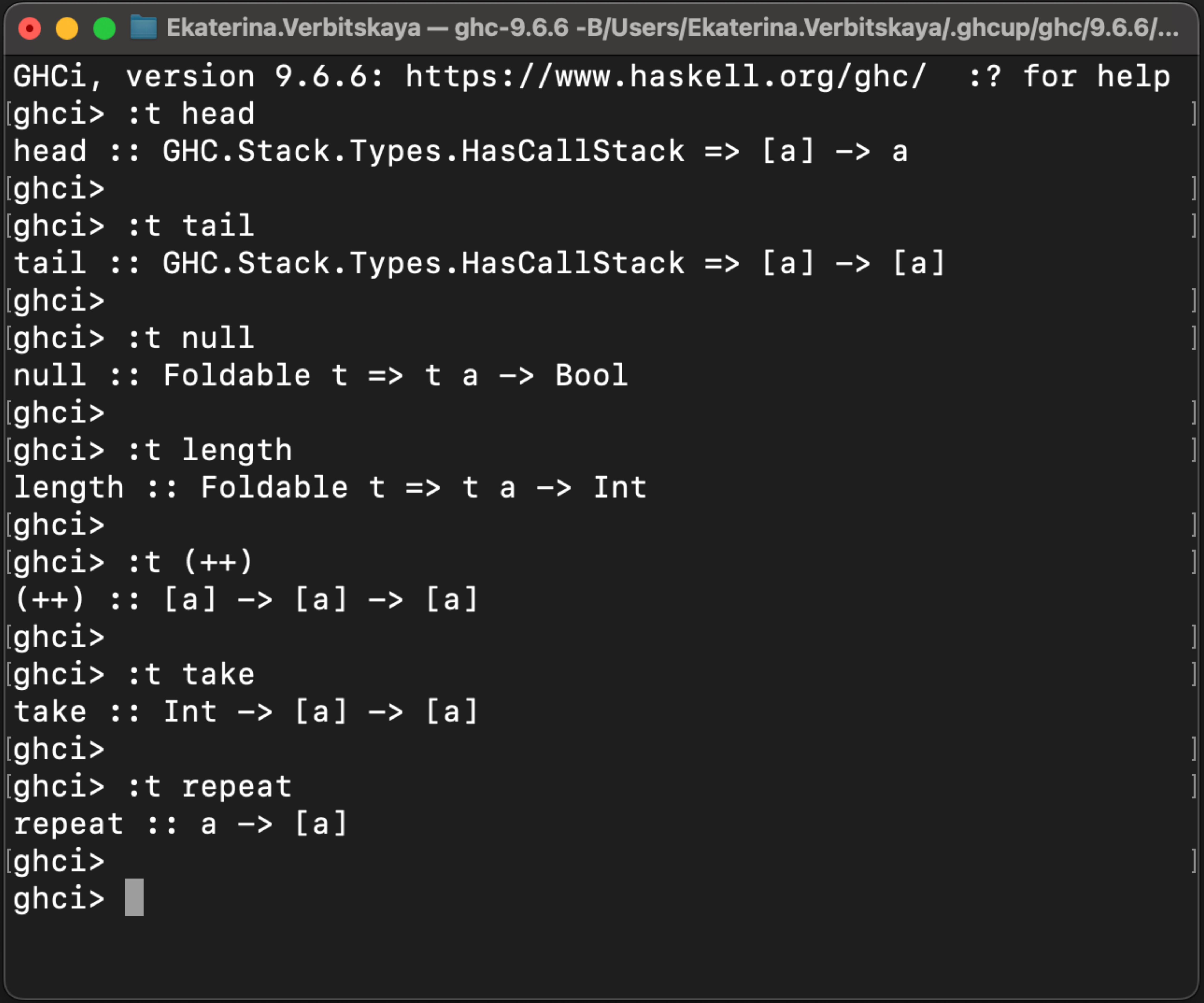
FUNCTIONAL PROGRAMMING

OUTLINE

- Functions over Lists
- Recursion

MODULE DATA.LIST

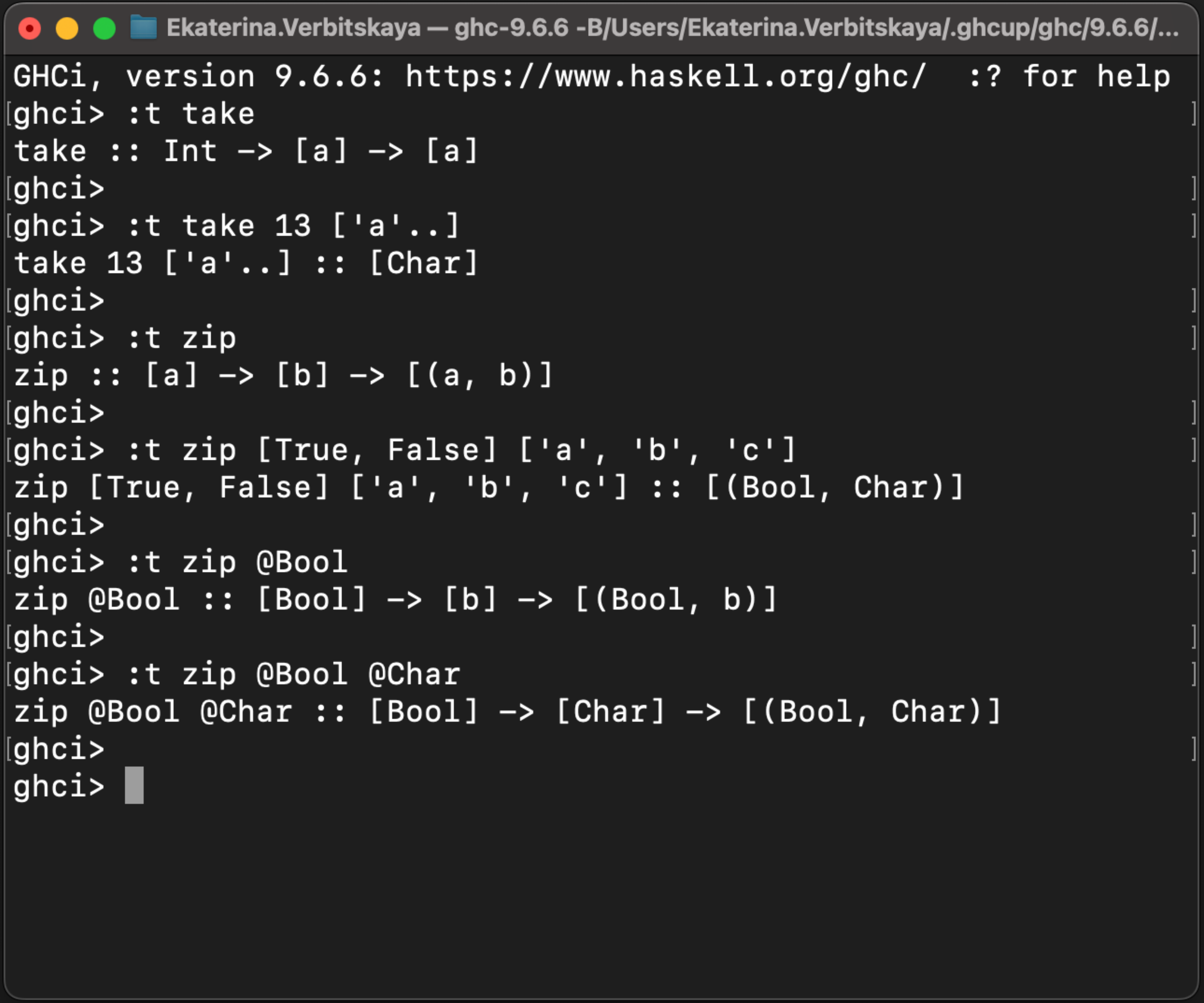
- null checks if a list is empty
- length finds the length of a list
- (++) concatenates two lists
- take n returns first n elements of a list
- repeat x constructs an infinite list by repeating the value x


Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/...

```
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
ghci> :t head
head :: GHC.Stack.Types.HasCallStack => [a] -> a
ghci>
ghci> :t tail
tail :: GHC.Stack.Types.HasCallStack => [a] -> [a]
ghci>
ghci> :t null
null :: Foldable t => t a -> Bool
ghci>
ghci> :t length
length :: Foldable t => t a -> Int
ghci>
ghci> :t (++)
(++) :: [a] -> [a] -> [a]
ghci>
ghci> :t take
take :: Int -> [a] -> [a]
ghci>
ghci> :t repeat
repeat :: a -> [a]
ghci>
ghci>
```

PARAMETRIC POLYMORPHISM

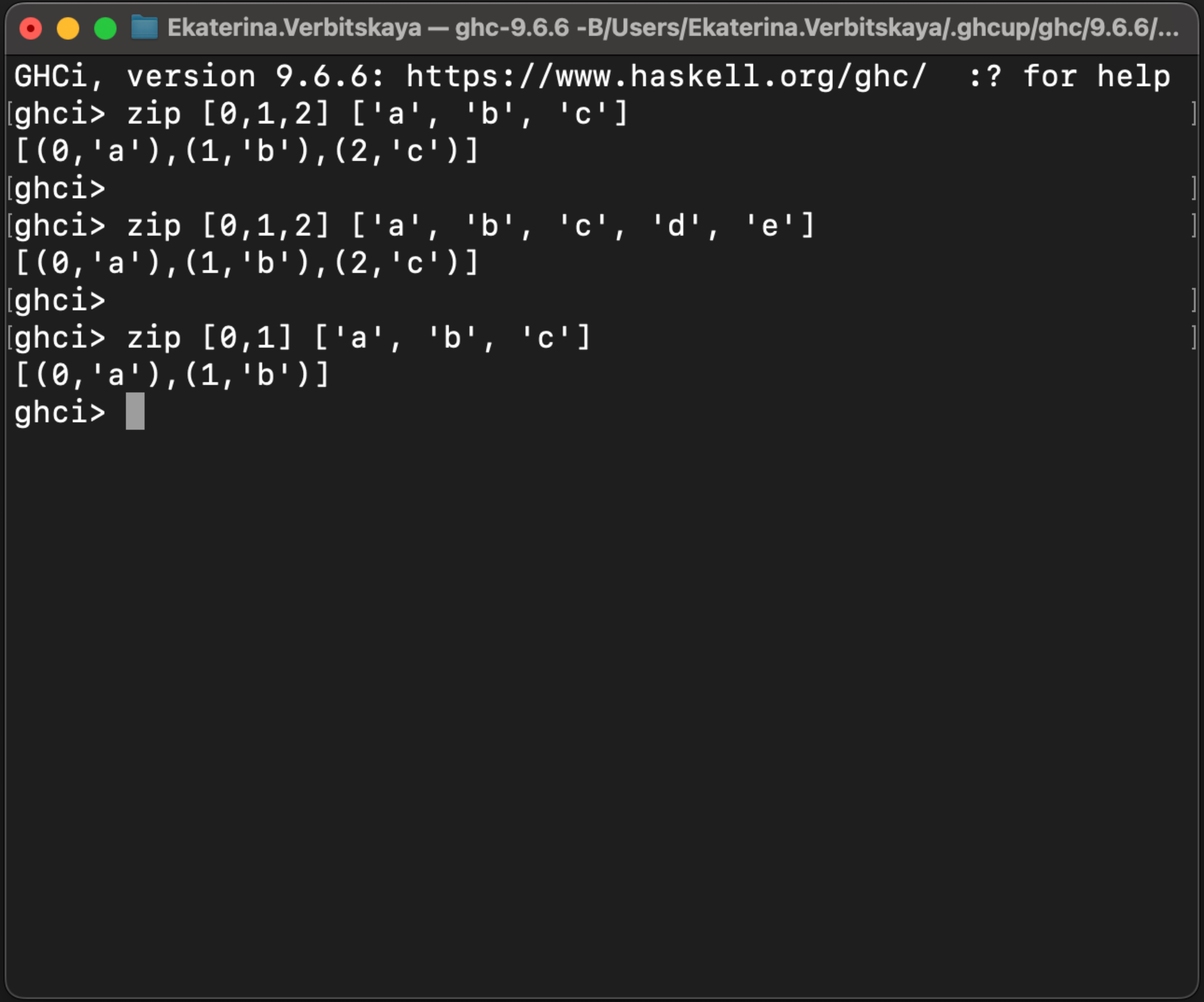
- A type contains one or more type variables
- Any value can be substituted for a type variable
- Simultaneously substitute all of the same variables
- Different variables can have different types substituted for them


Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/...

```
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
ghci> :t take
take :: Int -> [a] -> [a]
ghci>
ghci> :t take 13 ['a'..]
take 13 ['a'..] :: [Char]
ghci>
ghci> :t zip
zip :: [a] -> [b] -> [(a, b)]
ghci>
ghci> :t zip [True, False] ['a', 'b', 'c']
zip [True, False] ['a', 'b', 'c'] :: [(Bool, Char)]
ghci>
ghci> :t zip @Bool
zip @Bool :: [Bool] -> [b] -> [(Bool, b)]
ghci>
ghci> :t zip @Bool @Char
zip @Bool @Char :: [Bool] -> [Char] -> [(Bool, Char)]
ghci>
ghci> 
```

ZIP

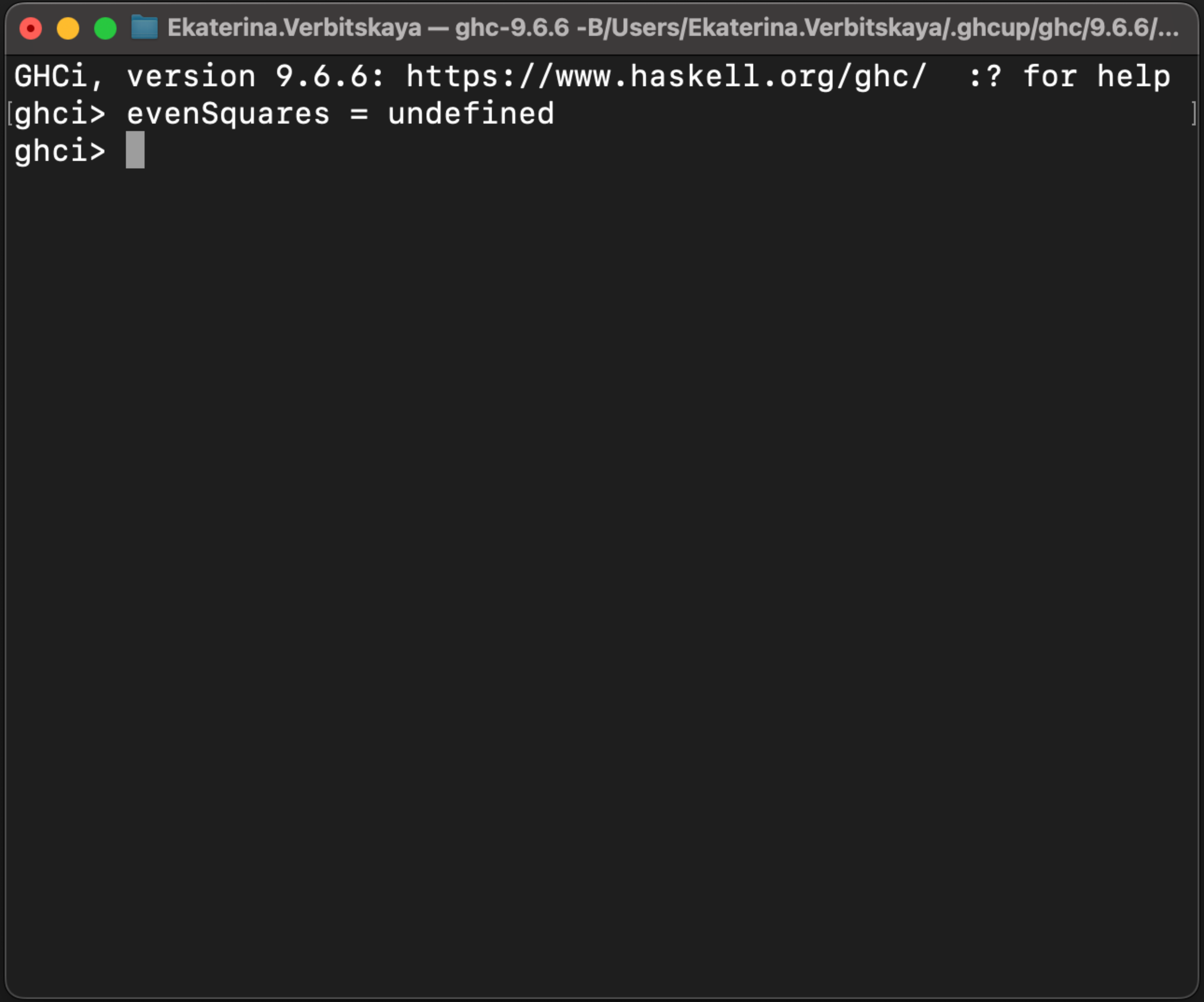
- “Zips” two lists together
- The head of one list is joined by the head of the other
- The length of the resulting list is equal to the length of the shortest list


Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/...

```
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
ghci> zip [0,1,2] ['a', 'b', 'c']
[(0,'a'),(1,'b'),(2,'c')]
ghci>
ghci> zip [0,1,2] ['a', 'b', 'c', 'd', 'e']
[(0,'a'),(1,'b'),(2,'c')]
ghci>
ghci> zip [0,1] ['a', 'b', 'c']
[(0,'a'),(1,'b')]
ghci>
```

EXERCISES

- Implement evenSquares using zip


Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/...

```
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
ghci> evenSquares = undefined
ghci> 
```

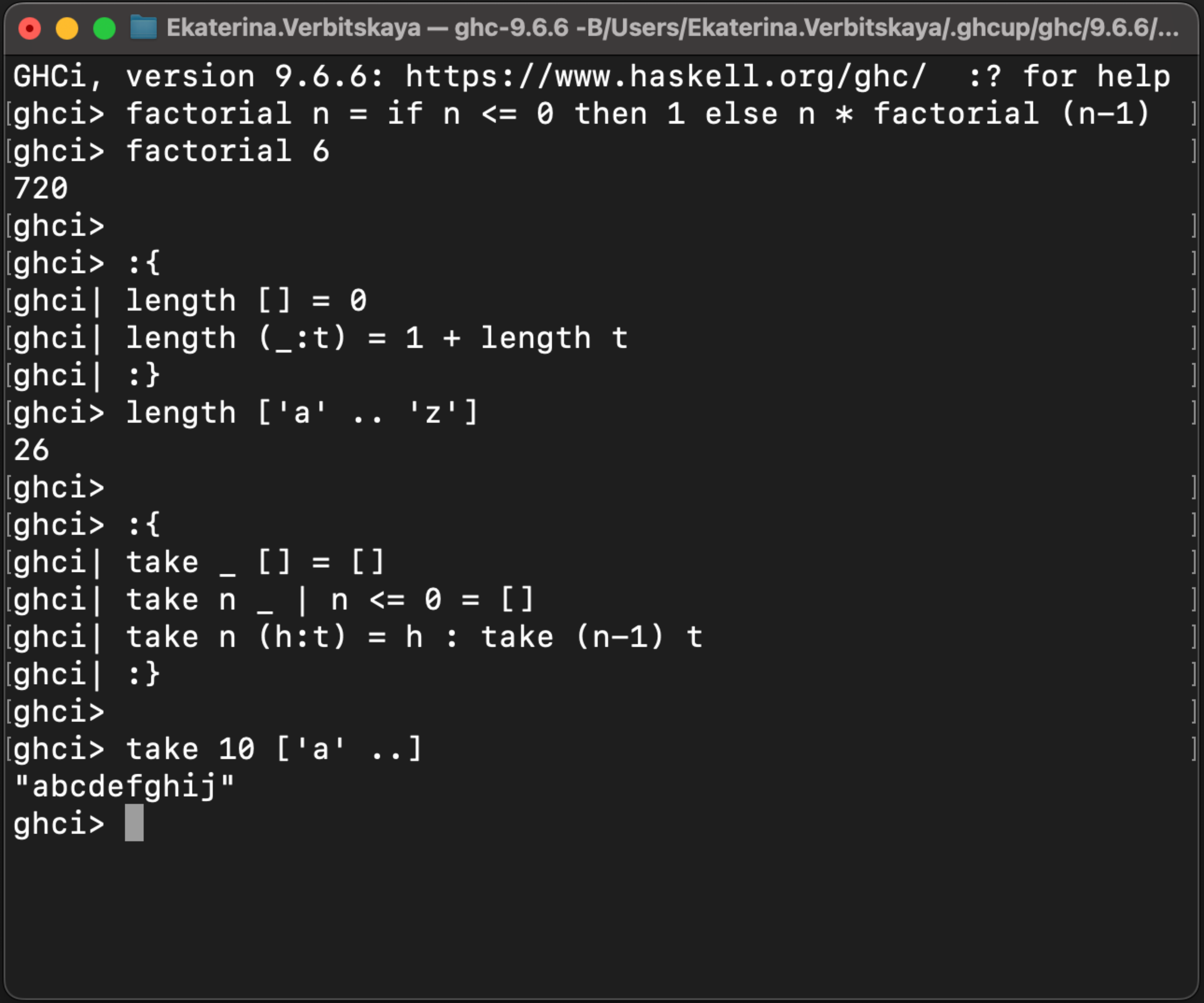
OUTLINE

- Functions over Lists
- Recursion

RECURSION

- In the body of a function, there is a call to the function itself

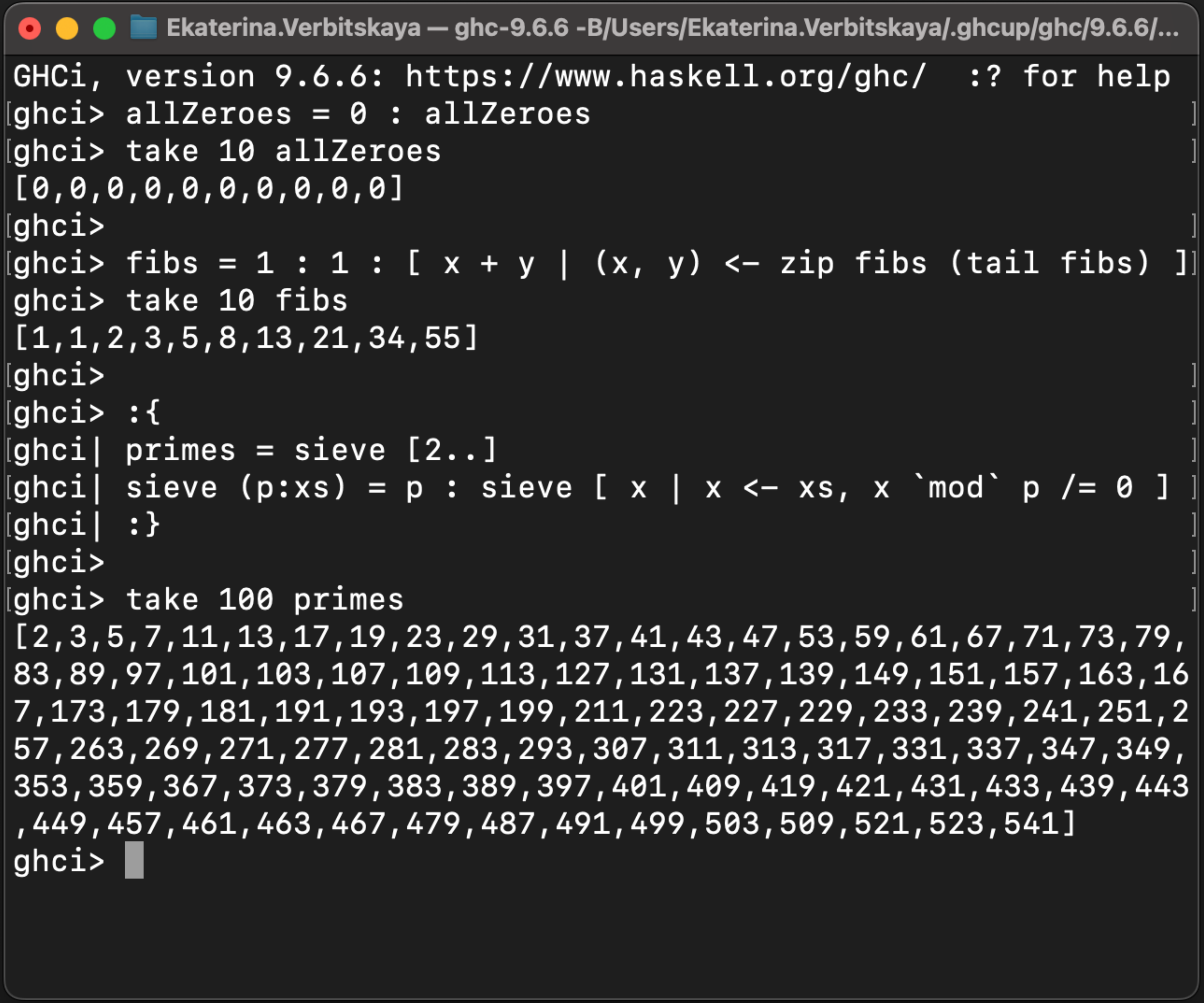
- If you want your recursion to terminate, don't forget about the base case


Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/...

```
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
ghci> factorial n = if n <= 0 then 1 else n * factorial (n-1)
ghci> factorial 6
720
ghci>
ghci> :{
ghci| length [] = 0
ghci| length (_:t) = 1 + length t
ghci| :}
ghci> length ['a' .. 'z']
26
ghci>
ghci> :{
ghci| take _ [] = []
ghci| take n _ | n <= 0 = []
ghci| take n (h:t) = h : take (n-1) t
ghci| :}
ghci>
ghci> take 10 ['a' ..]
"abcdefghij"
ghci>
```

NEAT TRICKS WITH LAZINESS

- Haskell is lazy
  - Values are not computed until they are needed
- Laziness enables infinite data structures
- Fibs
  - [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ..]
  - [1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ..]
  - [2, 3, 5, 8, 13, 21, 34, 55, 89, 104, ..]

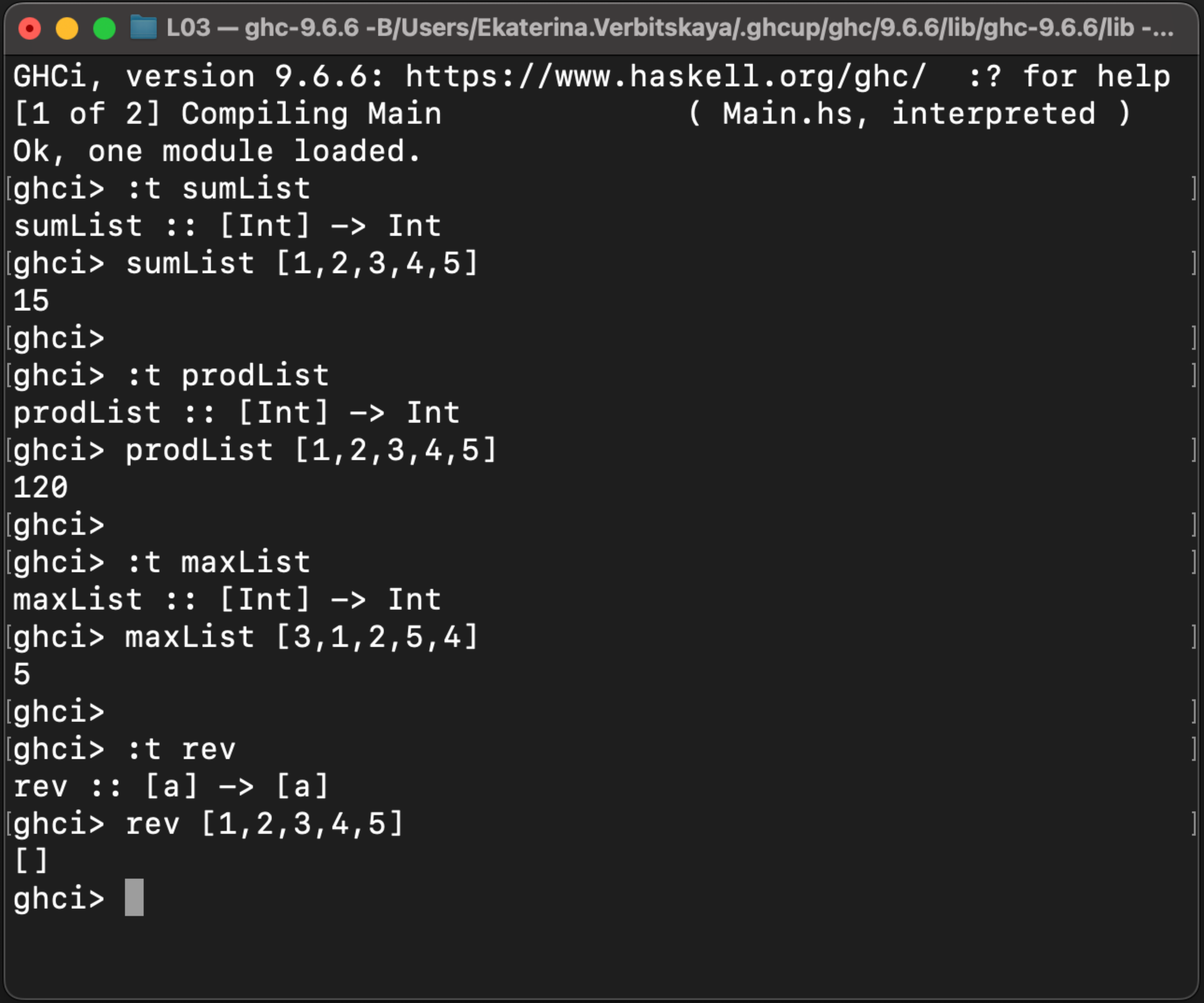


Ekaterina.Verbitskaya — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/...

```
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
ghci> allZeroes = 0 : allZeroes
ghci> take 10 allZeroes
[0,0,0,0,0,0,0,0,0,0]
ghci>
ghci> fibs = 1 : 1 : [ x + y | (x, y) <- zip fibs (tail fibs) ]
ghci> take 10 fibs
[1,1,2,3,5,8,13,21,34,55]
ghci>
ghci> :{
ghci| primes = sieve [2..]
ghci| sieve (p:xs) = p : sieve [ x | x <- xs, x `mod` p /= 0 ]
ghci| :}
ghci>
ghci> take 100 primes
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,
83,89,97,101,103,107,109,113,127,131,137,139,149,151,157,163,16
7,173,179,181,191,193,197,199,211,223,227,229,233,239,241,251,2
57,263,269,271,277,281,283,293,307,311,313,317,331,337,347,349,
353,359,367,373,379,383,389,397,401,409,419,421,431,433,439,443
,449,457,461,463,467,479,487,491,499,503,509,521,523,541]
ghci>
```

EXERCISES

- Implement functions:
  - sumList which sums up the elements of a list of Ints
  - prodList which multiplies them
  - maxList which finds the maximum number of a list of Ints
  - rev which reverses a list


L03 — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup/ghc/9.6.6/lib/ghc-9.6.6/lib -...

```
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
[1 of 2] Compiling Main             ( Main.hs, interpreted )
Ok, one module loaded.
ghci> :t sumList
sumList :: [Int] -> Int
ghci> sumList [1,2,3,4,5]
15
ghci>
ghci> :t prodList
prodList :: [Int] -> Int
ghci> prodList [1,2,3,4,5]
120
ghci>
ghci> :t maxList
maxList :: [Int] -> Int
ghci> maxList [3,1,2,5,4]
5
ghci>
ghci> :t rev
rev :: [a] -> [a]
ghci> rev [1,2,3,4,5]
[]
ghci> 
```