INTRODUCTION TO

FUNCTIONAL PROGRAMMING

OUTLINE

- Pattern matching
- Lists
- List comprehensions
- Functions over Lists
- Recursion

PATTERN MATCHING: TUPLES

- Pattern matching a tuple is reminiscent of destructuring declarations in Kotlin
  - ~~But better~~
  - Passing a triple to a function which pattern matches a pair leads to a type error

2024 — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcu...

```
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
ghci> :{
ghci| f pair =
ghci|   let (fst, snd) = pair
ghci|   in  fst + snd
ghci| :}
ghci> f (13, 42)
55
ghci> :{
ghci| f (fst, snd) =
ghci|   fst + snd
ghci| :}
ghci> f (13, 42)
55
ghci> f (13, 42, 777)

<interactive>:12:3: error: [GHC-83865]
    • Couldn't match expected type: (a, a)
                  with actual type: (a0, b0, c0)
    • In the first argument of 'f', namely '(13, 42, 777)'
      In the expression: f (13, 42, 777)
      In an equation for 'it': it = f (13, 42, 777)
    • Relevant bindings include it :: a (bound at <interactive
>:12:1)
ghci>
```

PATTERN MATCHING: LISTS

- You can pattern match on:
  - Empty list
  - Non-empty list
- Pattern matches can be embedded into each other
- Wildcards match anything

L01 — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcu...

```
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
ghci> isEmpty [] = True
ghci> isEmpty []
True
ghci> isCons (h:t) = True
ghci> isCons "Hello"
True
ghci> unCons (h:t) = (h, t)
ghci> unCons "Hello"
('H',"ello")
ghci> exactly3 [x, y, z] = (x, y, z)
ghci> exactly3 "Hoy"
('H','o','y')
ghci> deep [x, (h:t), _] = (x, h, t)
ghci> deep ["Hello", "World", "!"]
("Hello",'W',"orld")
ghci>
```

NON-EXHAUSTIVE PATTERNS

- We want functions to be **total**
- Make sure your patterns are exhaustive, i.e. they cover all possible values
- Be cautious when defining patten-matching functions in GHCi
  - Each line is its own definition
  - GHCi redefines functions
  - Use :{ :}

L01 — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcu...

```
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
ghci> isEmpty [] = True
ghci> isEmpty []
True
ghci> isEmpty [1,2,3]
*** Exception: <interactive>:1:1-17: Non-exhaustive patterns i
n function isEmpty

ghci> :{
ghci| isEmpty [] = True
ghci| isEmpty (h:t) = False
ghci| :}
ghci> isEmpty []
True
ghci> isEmpty [1,2,3]
False
ghci> isEmpty [] = True
ghci> isEmpty (h:t) = False
ghci> isEmpty []
*** Exception: <interactive>:11:1-21: Non-exhaustive patterns
in function isEmpty
```

MATCHING ORDER

- From top to bottom
- From left to right
- Once one match succeeds, the evaluation stops

L01 — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcu...

```
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
ghci> :{                                                        ]
ghci| isEmpty [] = True                                         ]
ghci| isEmpty _ = False                                         ]
ghci| :}                                                        ]
ghci> isEmpty [] && not (isEmpty [1,2,3])                       ]
True
ghci> :{                                                        ]
ghci| isEmpty _ = False                                         ]
ghci| isEmpty [] = True                                         ]
ghci| :}                                                        ]

<interactive>:8:1: warning: [GHC-53633] [-Woverlapping-pattern
s]
    Pattern match is redundant
    In an equation for 'isEmpty': isEmpty [] = ...
ghci> isEmpty [] && not (isEmpty [1,2,3])                       ]
False
ghci> isEmpty []                                                ]
False
ghci>
```

GUARDS

- You can add additional conditions for pattern matching (guards)
- otherwise or True can be used as a catch-all guard – only use it as the last guard
- May be used only for some patterns
- One guard cannot be shared between multiple patterns

L01 — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcu...

```
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
ghci> :{
ghci| describeAge age
ghci|   | age < 0 = "Invalid age"
ghci|   | age == 0 = "Newborn"
ghci|   | age < 13 = "Child"
ghci|   | age < 20 = "Teenager"
ghci|   | age < 65 = "Adult"
ghci|   | otherwise = "Senior"
ghci| :}
ghci> :{
ghci| describeList [] = "Empty list"
ghci| describeList [_] = "Singleton list"
ghci| describeList xs
ghci|   | length xs <= 10 = "Short list"
ghci|   | otherwise = "Long list"
ghci| :}
ghci> describeList [1,2,3]
"Short list"
ghci> describeAge 15
"Teenager"
ghci>
```

CASE-EXPRESSIONS

- Another patten-matching construction
- Can be written in one line
- What to use is a matter of style

L01 — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcu...

```
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
ghci> :{
ghci| isEmpty xs = case xs of
ghci|   [] -> True
ghci|   _ -> False
ghci| :}
ghci> isEmpty [] && not (isEmpty [1,2,3])
True
ghci> isEmpty xs = case xs of [] -> True; _ -> False
ghci> isEmpty [] && not (isEmpty [1,2,3])
True
ghci> :{
ghci| take n xs = case (n, xs) of
ghci|   (0, _) -> []
ghci|   (_, []) -> []
ghci|   (n, (h:t)) -> h : take (n-1) t
ghci| :}
ghci> take 5 [0..]
[0,1,2,3,4]
ghci> take 5 [0]
[0]
ghci>
```

WHERE CLAUSE

- Another way to bind a variable
- Can only be used on the level of function definition
- Is shared between multiple guard branches
  - let is not shared
- Avoid repeating computations

L01 — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcu...

```
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
ghci> :{
ghci| describeAge yearOfBirth currYear
ghci|   | age < 0 = "Incorrect age"
ghci|   | age == 0 = "Newborn"
ghci|   | age < 13 = "Child"
ghci|   | age < 20 = "Teenager"
ghci|   | age < 65 = "Adult"
ghci|   | otherwise = "Senior"
ghci|   where age = currYear - yearOfBirth
ghci| :}
ghci> describeAge 1930 2024
"Senior"
ghci> describeAge 1990 2024
"Adult"
ghci>
```

CODE IN FILES

- Create fail Main.hs
- Write code in it
  - Compile with ghc -O Main.hs
  - Load in ghci: ghci Main.hs
- Function main is the entry point

L01 — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcu...

```
[Ekaterina.Verbitskaya@NVC00653 L01 % vim Main.hs                ]
[Ekaterina.Verbitskaya@NVC00653 L01 % cat Main.hs                ]
main = putStrLn "Hello, File!"

[Ekaterina.Verbitskaya@NVC00653 L01 % ghc -O Main.hs             ]
[1 of 2] Compiling Main             ( Main.hs, Main.o )
[2 of 2] Linking Main
[Ekaterina.Verbitskaya@NVC00653 L01 % ./Main                     ]
Hello, File!
[Ekaterina.Verbitskaya@NVC00653 L01 % ghci Main.hs               ]
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> main                                                      ]
Hello, File!
ghci>
```

EXERCISES

- Implement functions:
  - calculate that gets a triple (x, op, y) and computes the result
    - x and y are integer numbers
    - op – character '+', '-', '*', or '^'
  - shapeArea that gets a triple (shape, a, b) and computes the area of the shape
    - shape – string "square", "cone", or "cylinder" – case insensitive
    - for cones and cylinders a – radius, b – height

L01 — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcu...

```
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
ghci> calculate (x, op, y) = undefined
ghci>
ghci> shapeArea (shape, a, b) = undefined
ghci>
ghci>
```

OUTLINE

- Pattern matching
- Lists
- List comprehensions
- Functions over Lists
- Recursion

LIST IS HETEROGENEOUS

- Every element of the list must have the same type

2024 — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcu...

```
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
ghci> hello = ['H', 'e', 'l', 'l', 'o']
ghci> wat = ["W", 'A', True]

<interactive>:2:13: error: [GHC-83865]
    • Couldn't match type 'Char' with '[Char]'
      Expected: String
        Actual: Char
    • In the expression: 'A'
      In the expression: ["W", 'A', True]
      In an equation for 'wat': wat = ["W", 'A', True]

<interactive>:2:18: error: [GHC-83865]
    • Couldn't match type 'Bool' with '[Char]'
      Expected: String
        Actual: Bool
    • In the expression: True
      In the expression: ["W", 'A', True]
      In an equation for 'wat': wat = ["W", 'A', True]
ghci>
```

LIST

- Linked list, no fuss
- Every list is either
  - Empty []
  - Has a head and a tail (h : t)
- head and tail are functions from Prelude, and they are partial

2024 — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcu...

```
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
ghci> head [1,2,3]
1
ghci> tail [1,2,3]
[2,3]
ghci> head []
*** Exception: Prelude.head: empty list
CallStack (from HasCallStack):
  error, called at libraries/base/GHC/List.hs:1644:3 in base:G
HC.List
  errorEmptyList, called at libraries/base/GHC/List.hs:87:11 i
n base:GHC.List
  badHead, called at libraries/base/GHC/List.hs:83:28 in base:
GHC.List
  head, called at <interactive>:3:1 in interactive:Ghci3
ghci> tail []
*** Exception: Prelude.tail: empty list
CallStack (from HasCallStack):
  error, called at libraries/base/GHC/List.hs:1644:3 in base:G
HC.List
  errorEmptyList, called at libraries/base/GHC/List.hs:130:28
in base:GHC.List
  tail, called at <interactive>:4:1 in interactive:Ghci3
ghci>
```

PATTERN MATCHING ON LISTS

- You can match on
  - An empty list
  - A non-empty list
  - List with known number of elements

2024 — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcu...

```
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
ghci> :{
ghci| f [] = 0
ghci| f [x] = x
ghci| f (x : []) = x
ghci| f (h : t) = h + f t
ghci| f (x : y : z : xs) = x + y + z + f xs
ghci| :}
```

PATTERN MATCHING ON LISTS

- You can match on
  - An empty list
  - A non-empty list
  - List with known number of elements

2024 — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcu...

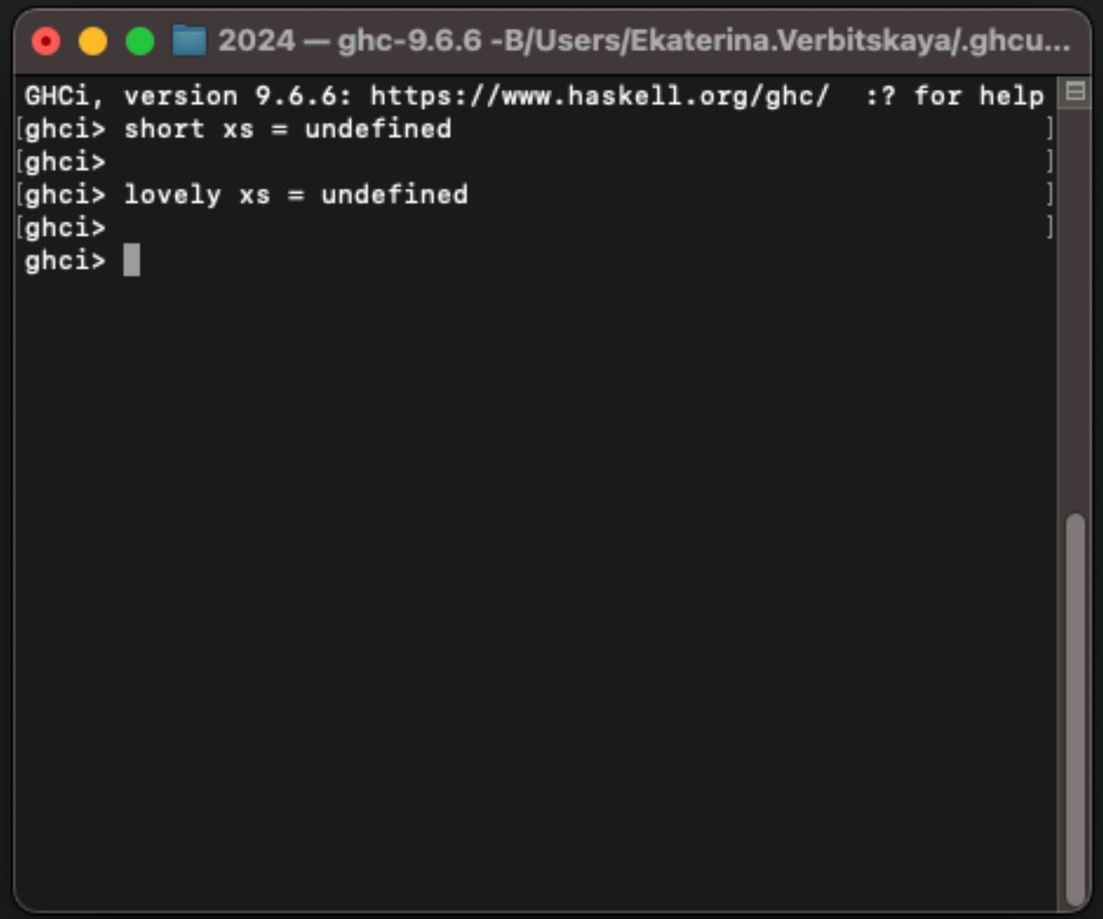
```
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
ghci> :{
ghci| f [] = 0
ghci| f [x] = x
ghci| f (x : []) = x
ghci| f (h : t) = h + f t
ghci| f (x : y : z : xs) = x + y + z + f xs
ghci| :}

<interactive>:4:1: warning: [GHC-53633] [-Woverlapping-patterns]
    Pattern match is redundant
    In an equation for 'f': f (x : []) = ...

<interactive>:6:1: warning: [GHC-53633] [-Woverlapping-patterns]
    Pattern match is redundant
    In an equation for 'f': f (x : y : z : xs) = ...
ghci>
```

EXERCISES

- Implement a predicate short which checks that the input list has length less than 3
- Implement a predicate lovely that checks that a list is either short (length <3) or has the number 14 as its third element



OUTLINE

- Pattern matching
- Lists
- List comprehensions
- Functions over Lists
- Recursion

A WAY TO FILTER A LIST

- Similar to set notation in maths
  - $\{x \mid x \in \mathbb{N}, x \equiv 0 \, (mod \, 2)\}$
- x <- [0..] selects the initial values for x
- x `mod` 2 == 0 is a predicate which filters elements

2024 — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcu...

```
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
ghci> evens = [ x | x <- [0..], x `mod` 2 == 0 ]
ghci> take 10 evens
[0,2,4,6,8,10,12,14,16,18]
ghci>
```

POST-PROCESS ELEMENTS

- You can run any kind of expression to the left of | to create values of the lists
- You can use more than one list to the right of |
- You can use let .. ~~in~~ to bind a variable name

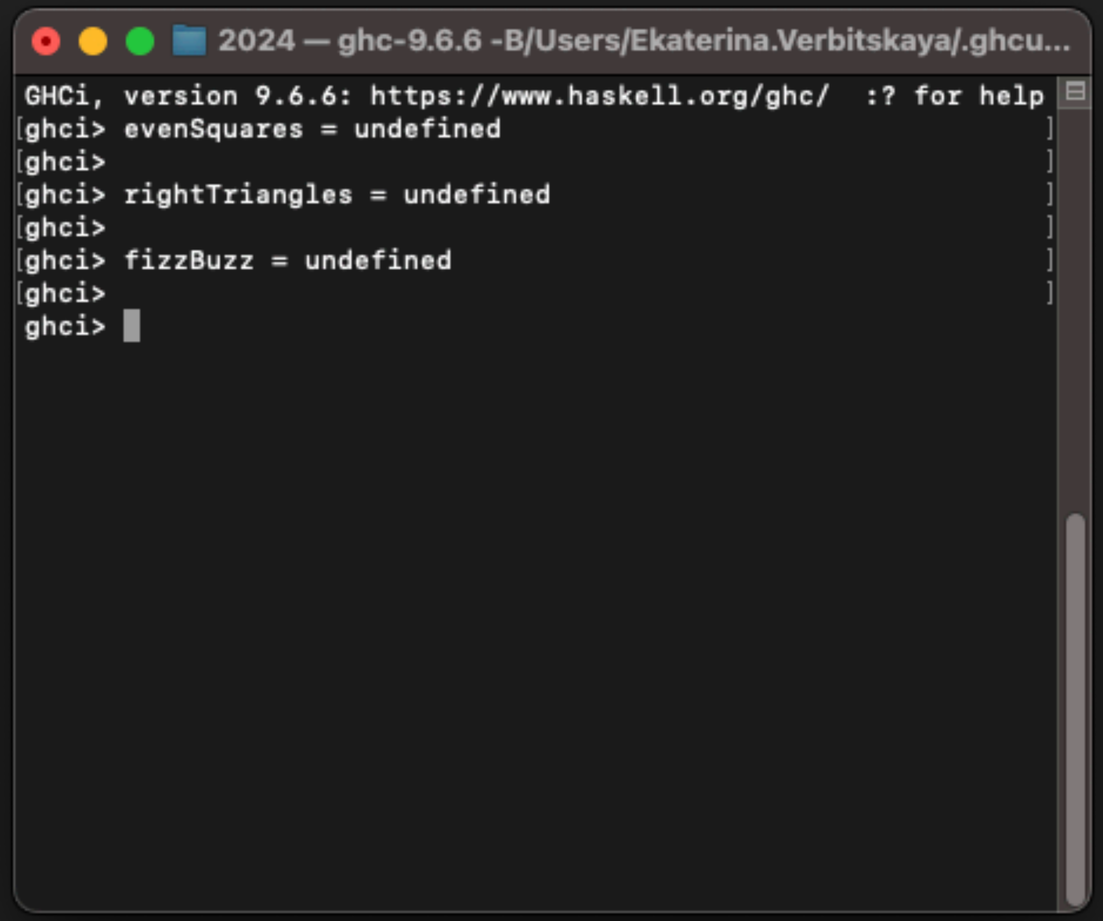
2024 — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcu...

```
GHCi, version 9.6.6: https://www.haskell.org/ghc/  :? for help
ghci> evens = [ x | x <- [0..], x `mod` 2 == 0 ]
ghci> take 10 evens
[0,2,4,6,8,10,12,14,16,18]
ghci>
ghci> evens = [ x * 2 | x <- [0..] ]
ghci> take 10 evens
[0,2,4,6,8,10,12,14,16,18]
ghci>
ghci> pairs = [ (x, y) | x <- [0 .. 2], y <- ['a' .. 'c'] ]
ghci> pairs
[(0,'a'),(0,'b'),(0,'c'),(1,'a'),(1,'b'),(1,'c'),(2,'a'),(2,'b
'),(2,'c')]
ghci>
ghci> squares = [(x, square) | x <- [0..], let square = x * x]

ghci> take 10 squares
[(0,0),(1,1),(2,4),(3,9),(4,16),(5,25),(6,36),(7,49),(8,64),(9
,81)]
ghci>
```

EXERCISES

- Create a list of even squares
- Create a list of possible integer lengths of sides of a right triangle
  - make sure there are no repetitions
  - make sure that lengths are ordered
- Create an infinite list of string values for a Fizz-Buzz challenge
  - [“1”, “2”, “Fizz”, “4”, “Buzz”, …, “14”, “FizzBuzz”,…]
  - use show to turn a number into a string



OUTLINE

- Pattern matching
- Lists
- List comprehensions
- 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

2024 — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcu...

```
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

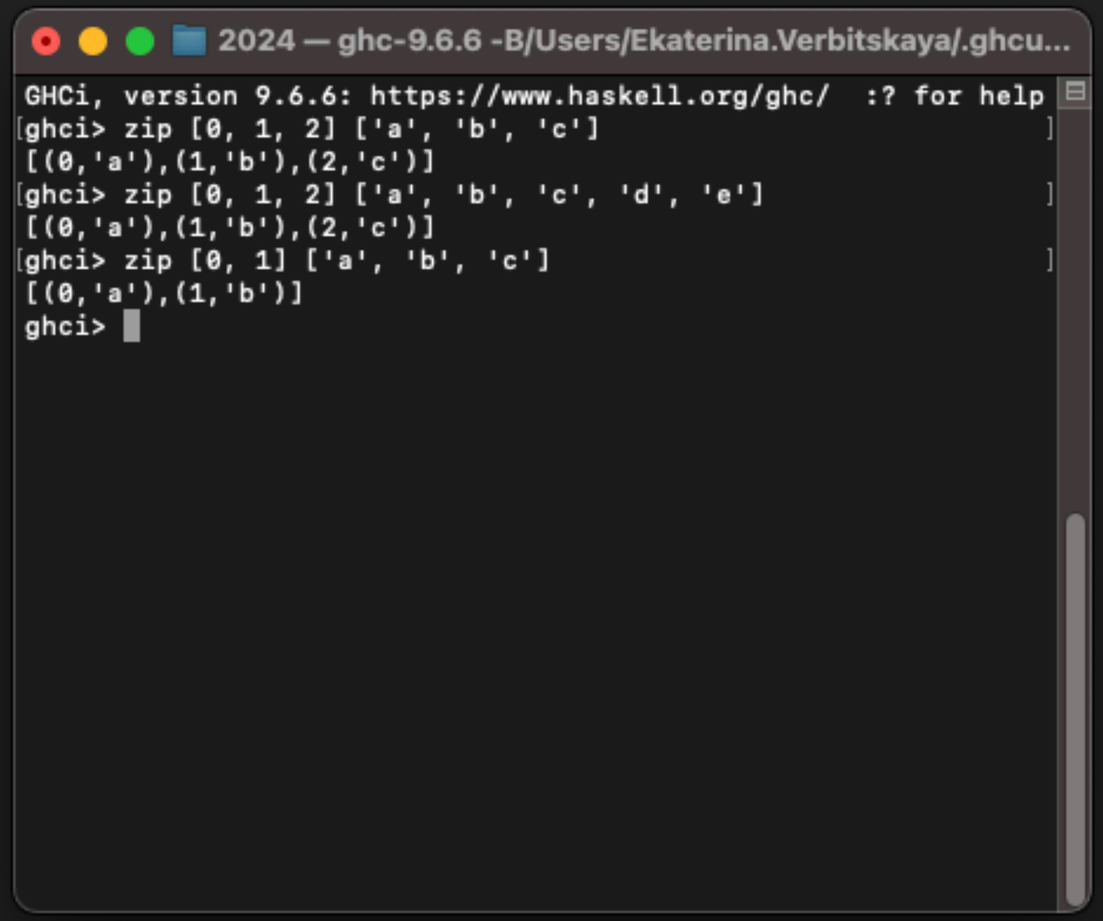
- A type contains one or more type variables
- Any value can be substituted for a type variable
- Simultaneously substitute all of the same variables
- Different variables can have different types substituted for them

2024 — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcu...

```
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



EXERCISES

- Implement evenSquares using zip

2024 — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcu...

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

OUTLINE

- Pattern matching
- Lists
- List comprehensions
- Functions over Lists
- Recursion

RECURSION

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

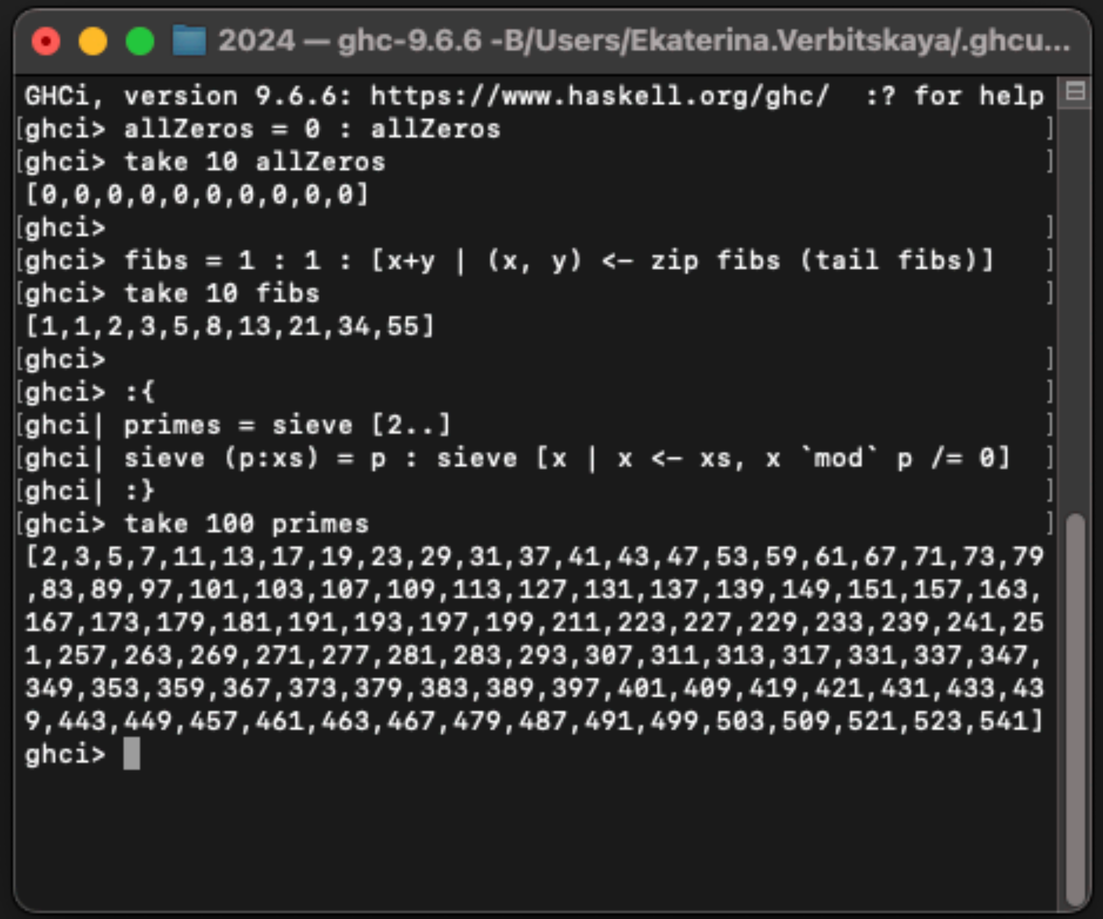
- If you want your recursion to terminate, don't forget about the base case

2024 — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcu...

```
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| take _ [] = []
ghci| take n _ | n <= 0 = []
ghci| take n (h:t) = h : take (n-1) t
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, ..]`



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

L02 — ghc-9.6.6 -B/Users/Ekaterina.Verbitskaya/.ghcup...

```
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> :t prodList
prodList :: [Int] -> Int
ghci> prodList [1,2,3,4,5]
120
ghci> :t maxList
maxList :: [Int] -> Int
ghci> maxList [3,1,2,5,4]
5
ghci> :t rev
rev :: [a] -> [a]
ghci> rev [1,2,3,4,5]
[]
ghci>
```