

INTRODUCTION TO

FUNCTIONAL PROGRAMMING

OUTLINE

- ▶ Organization
- ▶ Introduction
- ▶ Syntax basics
- ▶ Simplest types

MEET YOUR INSTRUCTOR

- ▶ Ekaterina Verbitskaia
- ▶ Researcher @ JetBrains Research PLAN,
Programming Languages and Program Analysis
Lab
- ▶ Interested in:
 - ▶ Functional and Relational Programming
 - ▶ Partial Evaluation, Metacomputations
- ▶ @kajigor



ORGANIZATION

- ▶ 2 classes per week
- ▶ Written exam (50% of your mark)
- ▶ Work during the term (50% of your mark)
 - ▶ Homework assignments (40/100 points)
 - ▶ Project at the end of the course (40/100 points)
 - ▶ Active participation during classes (40/100 points)

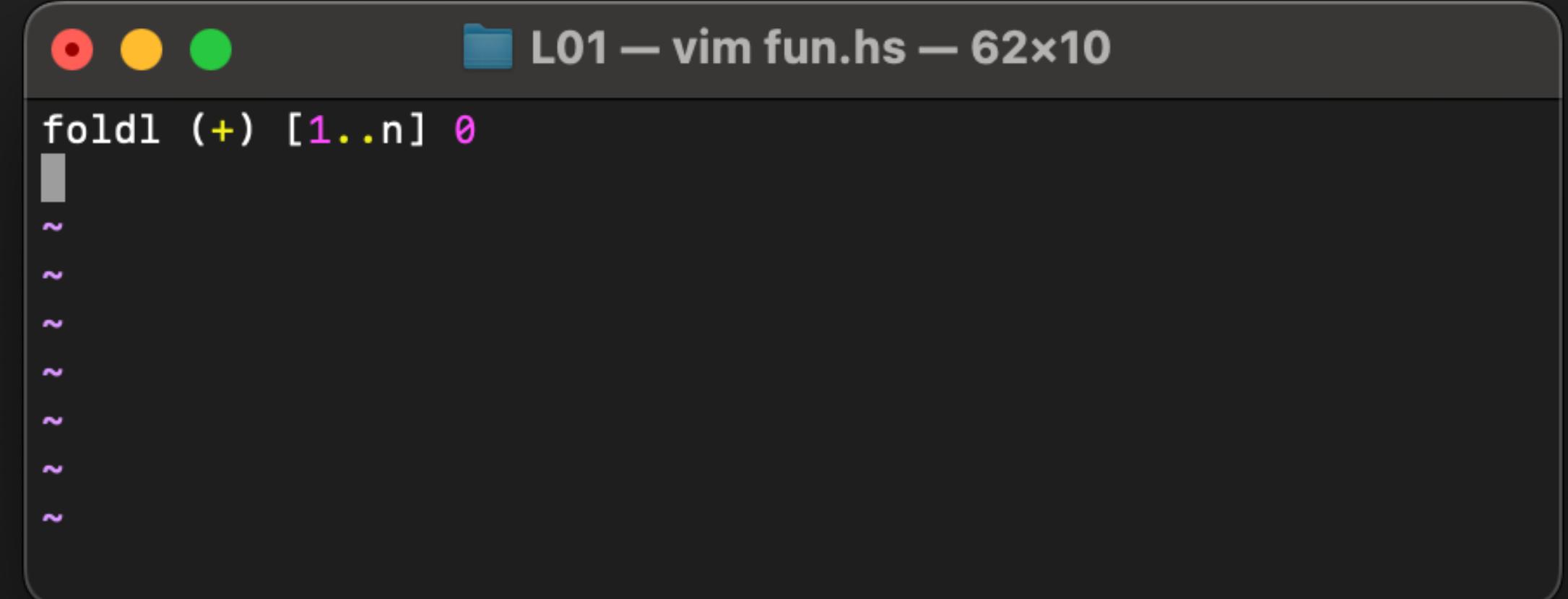


OUTLINE

- ▶ Organization
- ▶ Introduction
- ▶ Syntax basics
- ▶ Simplest types

FUNCTIONAL PROGRAMMING

- ▶ Functional programming computes by means of evaluation of functions

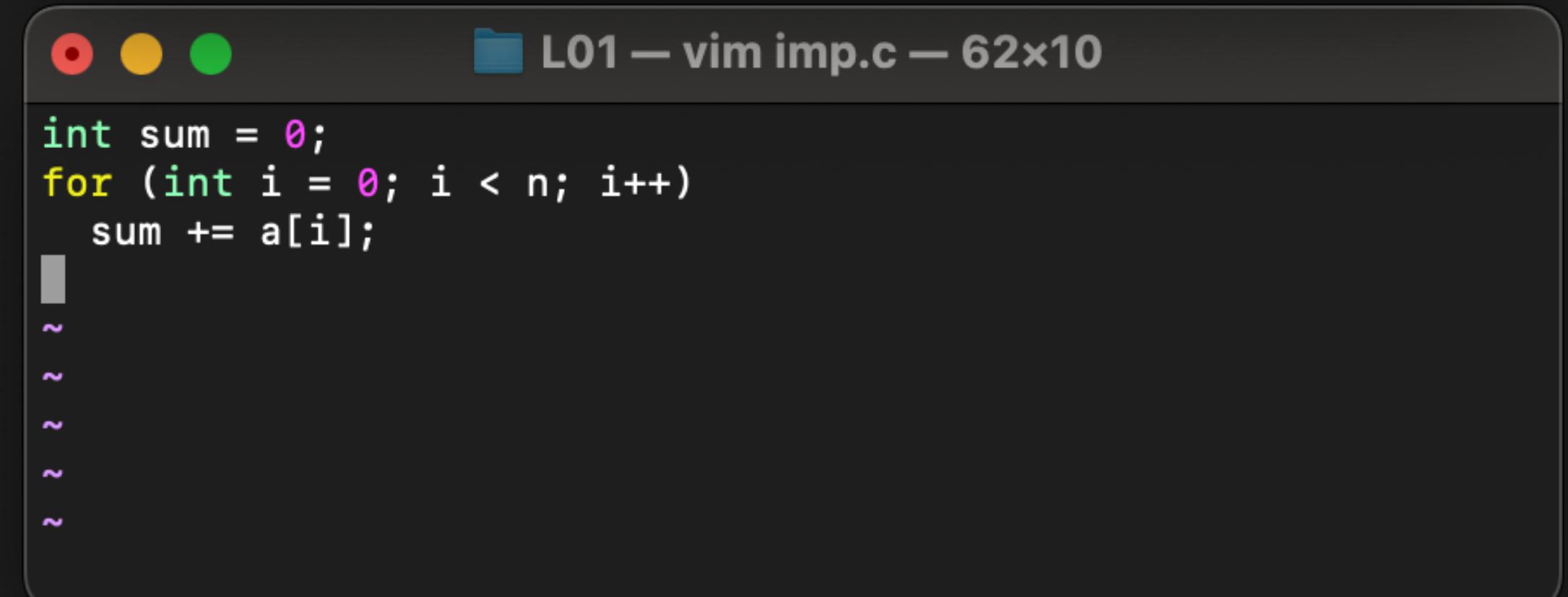


L01 – vim fun.hs – 62x10

```
foldl (+) [1..n] 0
```

The screenshot shows a terminal window with a dark background. At the top, there are three colored circles (red, yellow, green) and the text "L01 – vim fun.hs – 62x10". Below this is a code editor window containing Haskell code. The code defines a function "foldl" with the type "(+) [1..n] 0". The editor has a vertical scroll bar on the left and a status bar at the bottom.

- ▶ Imperative programming uses statements to change the program's state



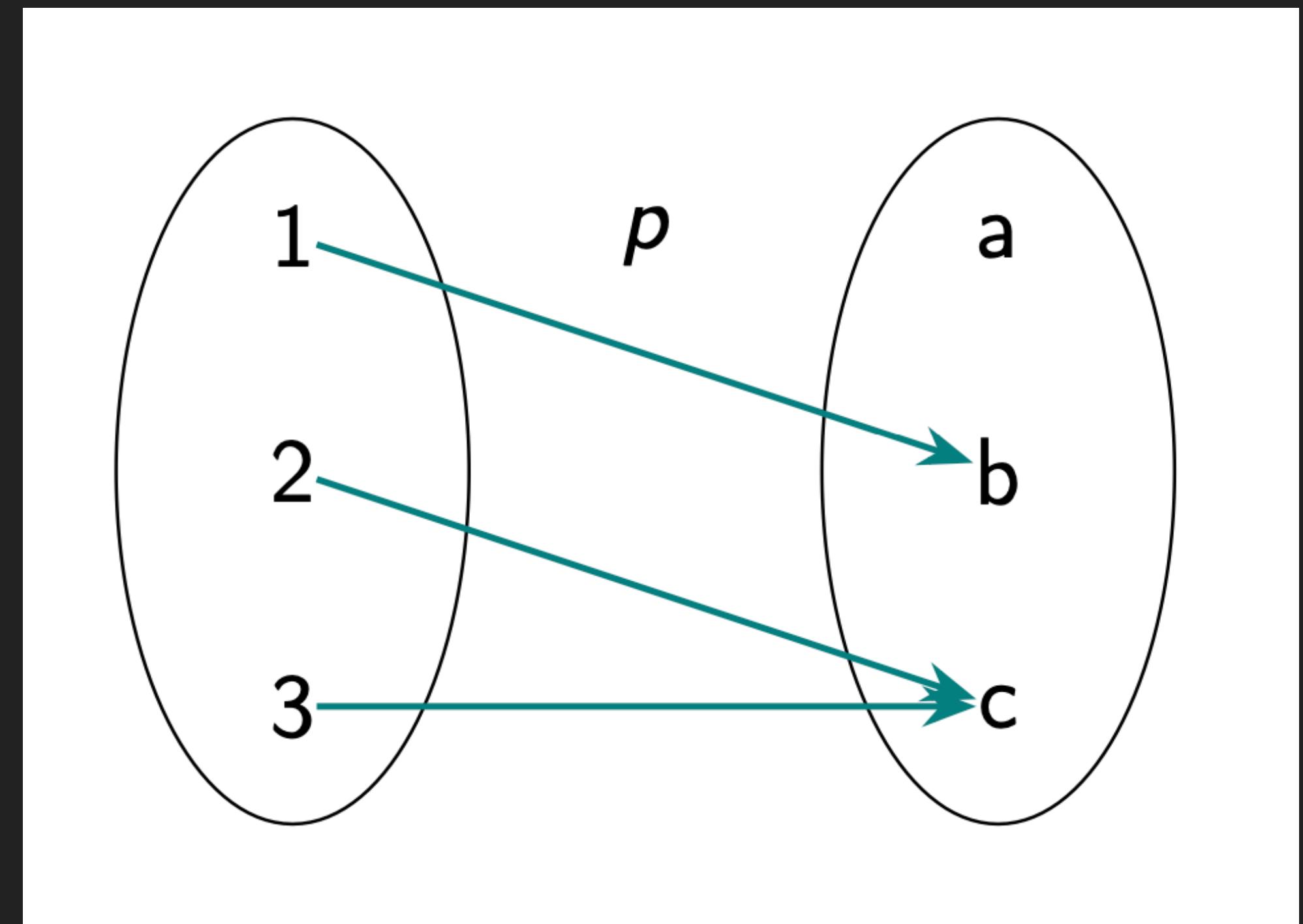
L01 – vim imp.c – 62x10

```
int sum = 0;
for (int i = 0; i < n; i++)
    sum += a[i];
```

The screenshot shows a terminal window with a dark background. At the top, there are three colored circles (red, yellow, green) and the text "L01 – vim imp.c – 62x10". Below this is a code editor window containing C code. The code initializes a variable "sum" to 0, then uses a "for" loop to iterate from 0 to "n-1", adding the value of "a[i]" to "sum" in each iteration. The editor has a vertical scroll bar on the left and a status bar at the bottom.

FUNCTION

- ▶ Fundamental building block of a program
- ▶ Pure
 - ▶ No side-effects such as IO or mutable state
 - ▶ **Always the same result for the same input**
- ▶ First-class: functions can be passed as arguments and returned as results
- ▶ Composition to create new functions



Haskell

- ▶ Pure functional programming language
- ▶ Lazy
- ▶ Strong static type system
- ▶ Type inference



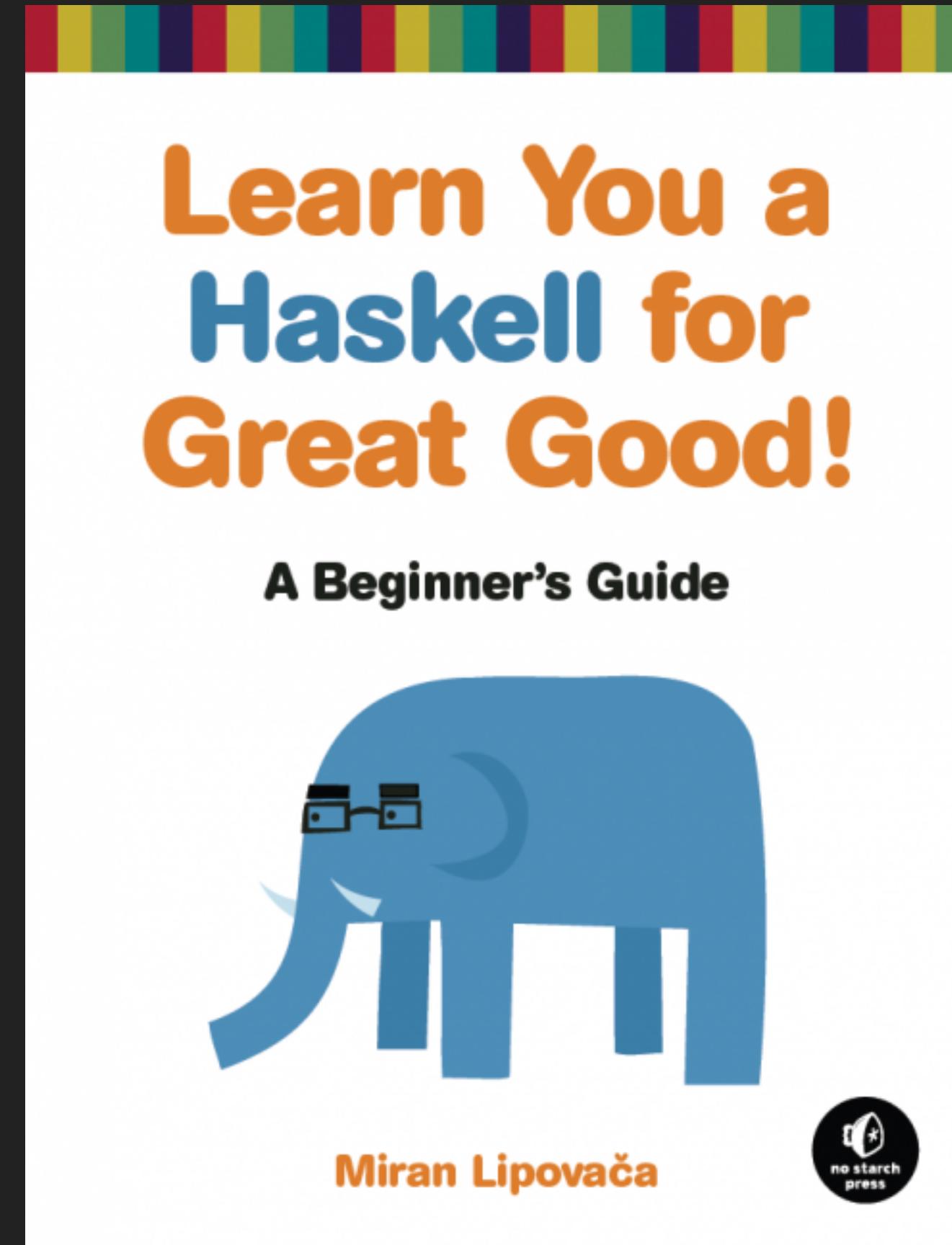
WHAT YOU NEED TO START

- ▶ Install with [GHcup](#):
 - ▶ [GHC](#) 9.6.6 – compiler
 - ▶ [Stack](#) 3.1.1 – build system
 - ▶ [HLS](#) 2.9.0.1 – language server
- ▶ [VSCode](#)
 - ▶ Haskell [extension](#)



RECOMMENDED RESOURCES

- ▶ Books:
 - ▶ [Learn You a Haskell for Great Good!](#) (free)
 - ▶ [Real World Haskell](#) (free)
 - ▶ [Haskell in Depth](#)
- ▶ [Cheat sheet](#)
- ▶ [Hoogle](#)

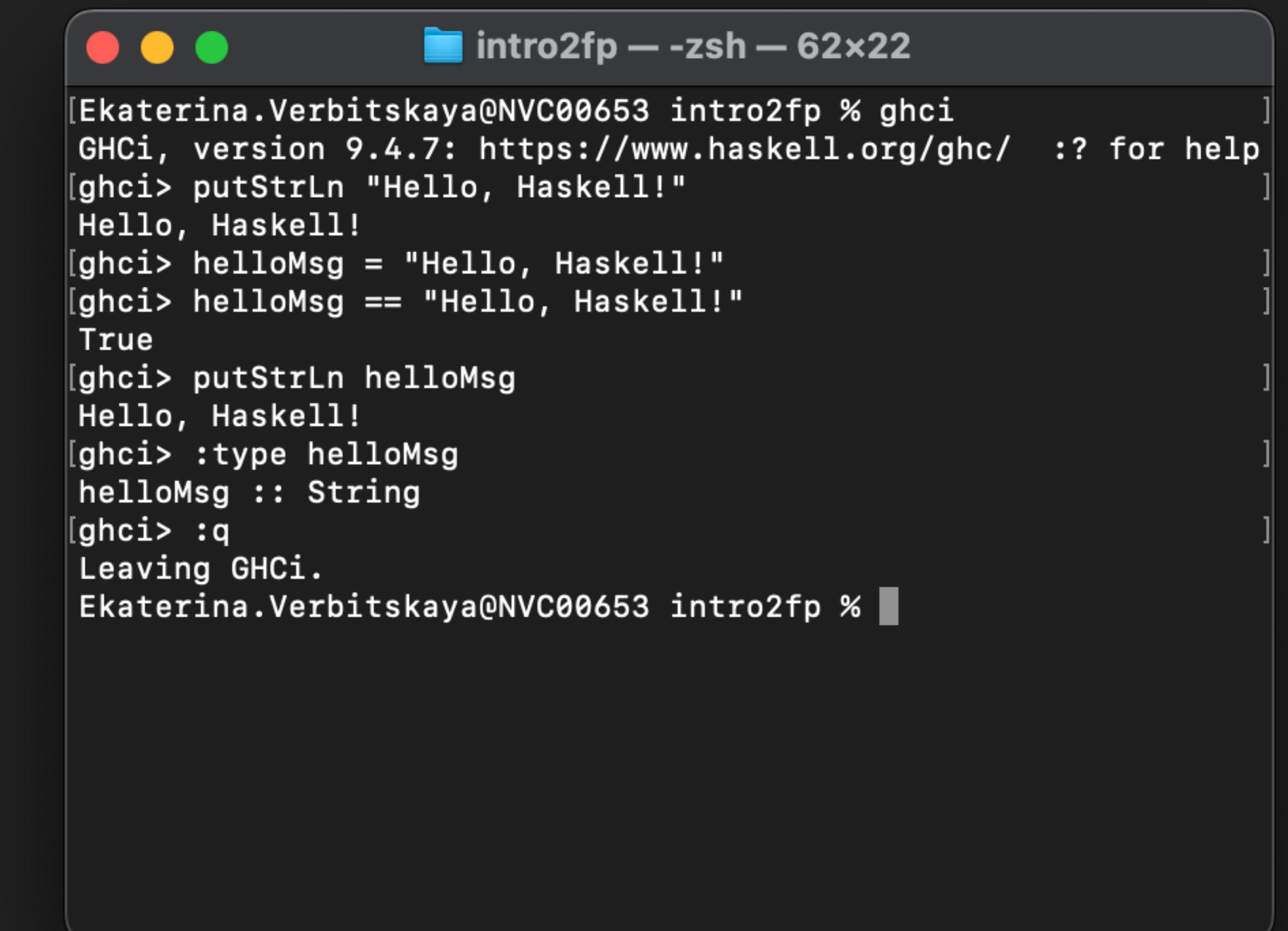


OUTLINE

- ▶ Organization
- ▶ Introduction
- ▶ Syntax basics
- ▶ Simplest types

HELLO HASKELL

- ▶ Run `ghci` and type after the prompt `ghci>`
- ▶ `putStrLn` to output a string
- ▶ `=` for variable binding
- ▶ `==` for equality check
- ▶ `:type` to check the type of the expression
- ▶ `:q` to quite `ghci`



```
[Ekaterina.Verbitskaya@NVC00653 intro2fp % ghci
GHCi, version 9.4.7: https://www.haskell.org/ghc/  ?: for help
[ghci> putStrLn "Hello, Haskell!"
Hello, Haskell!
[ghci> helloMsg = "Hello, Haskell!"
[ghci> helloMsg == "Hello, Haskell!"
True
[ghci> putStrLn helloMsg
Hello, Haskell!
[ghci> :type helloMsg
helloMsg :: String
[ghci> :q
Leaving GHCi.
Ekaterina.Verbitskaya@NVC00653 intro2fp % ]
```

FUNCTION APPLICATION

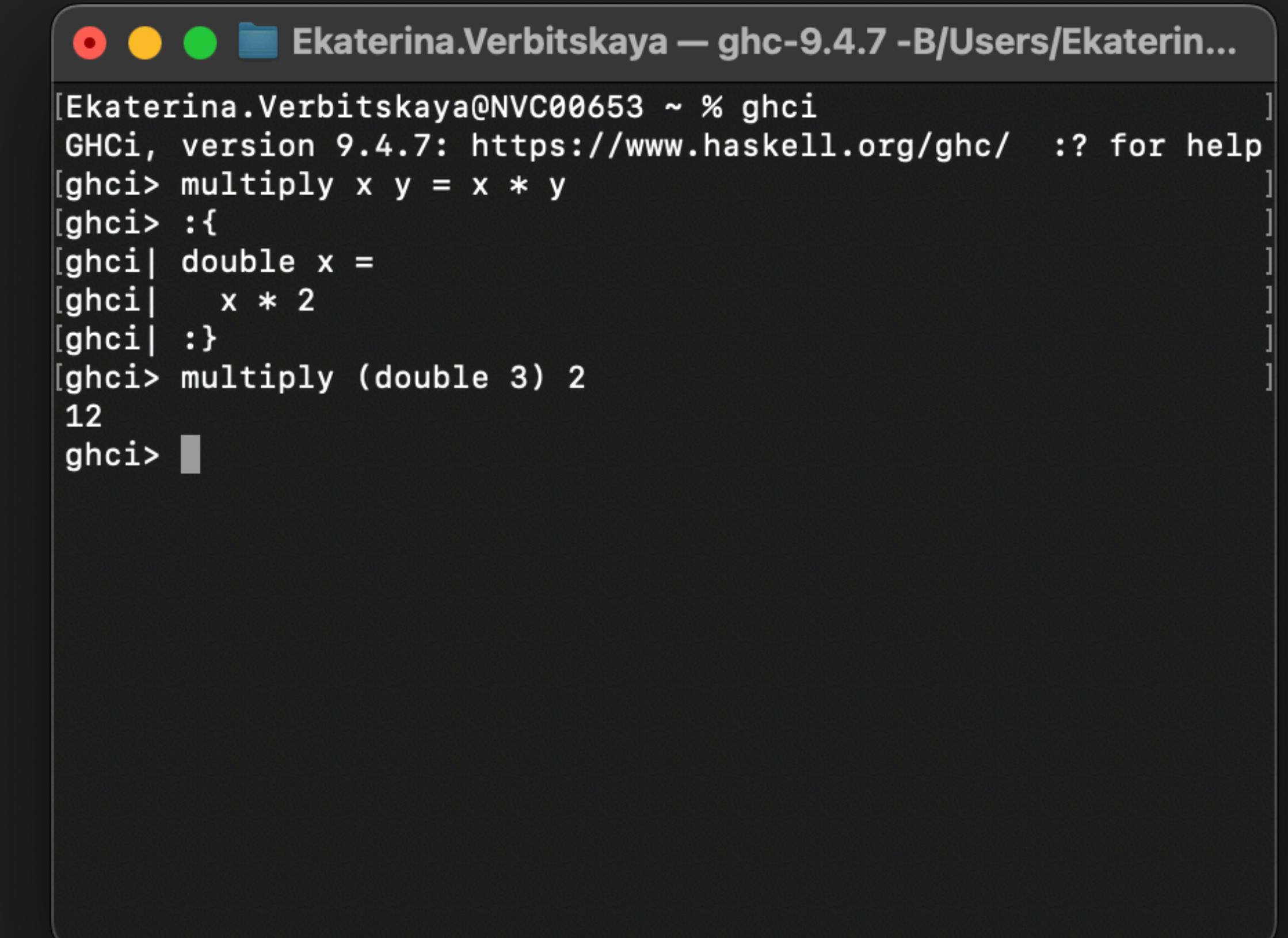
- ▶ No parentheses
 - ▶ `putStrLn helloMsg`
 - ▶ `take 5 helloMsg`
- ▶ Function application has the highest precedence:
 - ▶ `take 2 + 3 helloMsg`
 - ▶ `(take 2) + (3 helloMsg)`
- ▶ Parentheses should instead be used:
 - ▶ `take (2 + 3) helloMsg`

```
Ekaterina.Verbitskaya@NVC00653 intro2fp % ghci
GHCi, version 9.4.7: https://www.haskell.org/ghc/  ?: for help
[ghci> helloMsg = "Hello, Haskell!"
[ghci> putStrLn helloMsg
Hello, Haskell!
[ghci> take 5 helloMsg
"Hello"
[ghci> take 2 + 3 helloMsg

<interactive>:4:1: error:
  • No instance for (Show ([a0] -> [a0]))
    arising from a use of 'print'
    (maybe you haven't applied a function to enough arguments)
  • In a stmt of an interactive GHCi command: print it
[ghci> take (2+3) helloMsg
"Hello"
ghci> ]
```

FUNCTION DEFINITION

- ▶ `name arg1 ... argN = body`
- ▶ No parentheses around args or body
- ▶ Body must be on the same line or indented
- ▶ `name arg1 ... argN =
body line 1
...
body line M`



A screenshot of a terminal window titled "Ekaterina.Verbitskaya — ghc-9.4.7 -B/Users/Ekaterin...". The terminal shows the following Haskell code being run:

```
[Ekaterina.Verbitskaya@NVC00653 ~ % ghci
GHCi, version 9.4.7: https://www.haskell.org/ghc/  :? for help
[ghci> multiply x y = x * y
[ghci> :{
[ghci| double x =
[ghci|   x * 2
[ghci| :}
[ghci> multiply (double 3) 2
12
ghci> ]
```

INFIX OPERATORS

- ▶ Binary operators are intended to be used between operands
- ▶ Can have different fixity, associativity and precedence: **infixl 6 +** means that **(+)** is
 - ▶ an infix operator
 - ▶ left associative
 - ▶ has precedence of 6 (can be from 0 to 9, higher precedence is applied first)

```

Ekaterina.Verbitskaya — ghc-9.4.7 -B/Users/Ekaterin...
GHCi, version 9.4.7: https://www.haskell.org/ghc/  :? for help
[ghci> :info (+)
type Num :: * -> Constraint
class Num a where
  (+) :: a -> a -> a
  ...
  -- Defined in 'GHC.Num'
infixl 6 +
[ghci> :info (==)
type Eq :: * -> Constraint
class Eq a where
  (==) :: a -> a -> Bool
  ...
  -- Defined in 'GHC.Classes'
infix 4 ==
[ghci> :info (^)
(^) :: (Num a, Integral b) => a -> b -> a      -- Defined in
'GHC.Real'
infixr 8 ^
[ghci> 1 + 2 + 3 == 4^5^6
False
[ghci> (((1 + 2) + 3) == (4^(5^6)))

```

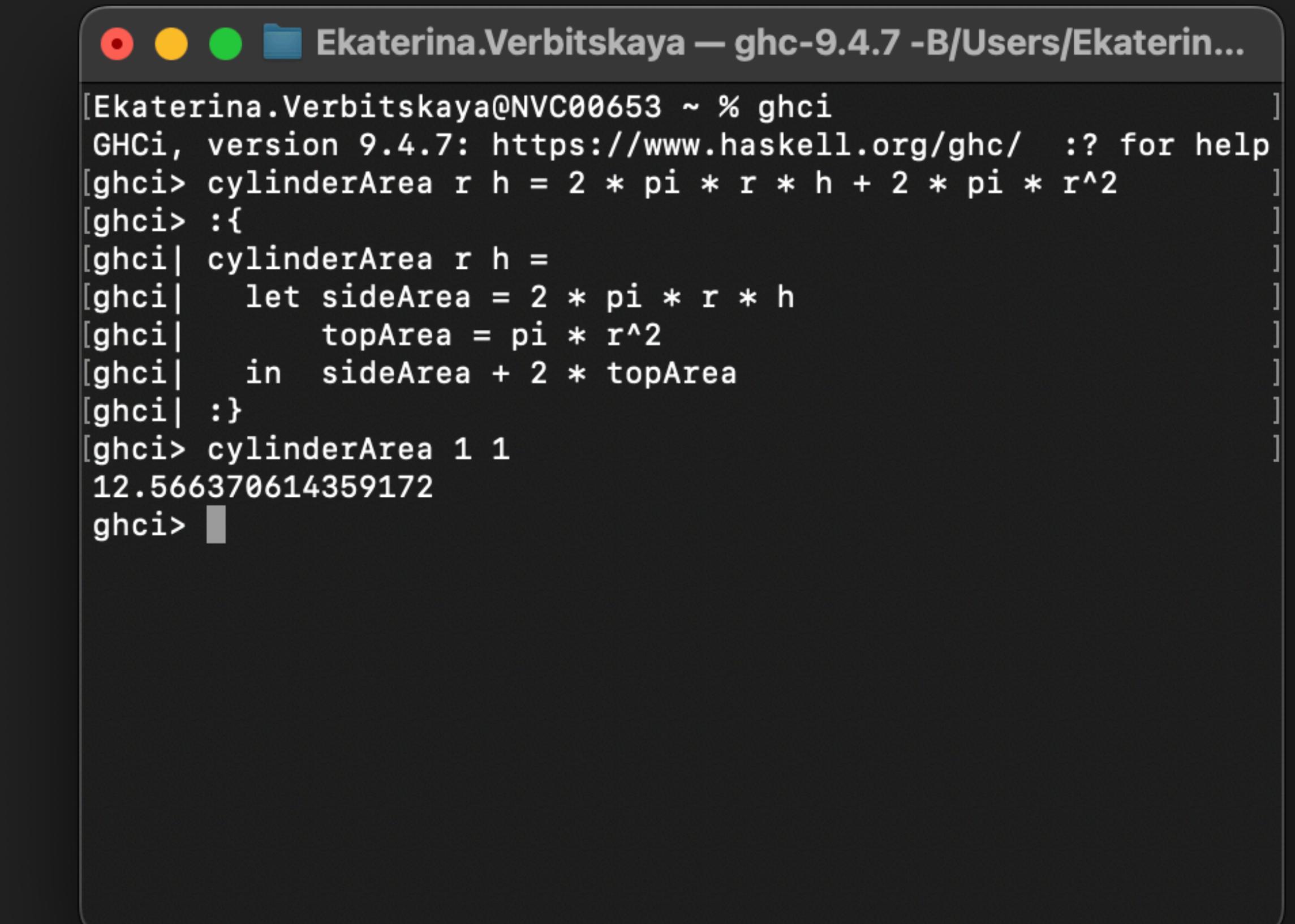
INFIX AND PREFIX NOTATION

- ▶ Infix operators can be used in prefix form
- ▶ Function names can be used in infix form
- ▶ When used in infix form, may be supplemented with fixity specification
- ▶ By default, no specification

```
Ekaterina.Verbitskaya — ghc-9.4.7 -B/Users/Ekaterin...
GHCi, version 9.4.7: https://www.haskell.org/ghc/  :? for help
[ghci> 2 * 3
6
[ghci> (*) 2 3
6
[ghci> div 6 3
2
[ghci> 6 `div` 3
2
[ghci> :info `div`
type Integral :: * -> Constraint
class (Real a, Enum a) => Integral a where
...
    div :: a -> a -> a
...
    -- Defined in 'GHC.Real'
infixl 7 `div`
[ghci> plus x y = x + y
[ghci> :info plus
plus :: Num a => a -> a -> a    -- Defined at <interactive>:6:
1
ghci> ]
```

VARIABLE BINDING

- ▶ To make your code more readable, introduce names for subexpressions
- ▶ **let** varName₁ = value₁
...
varName_N = value_N
in body

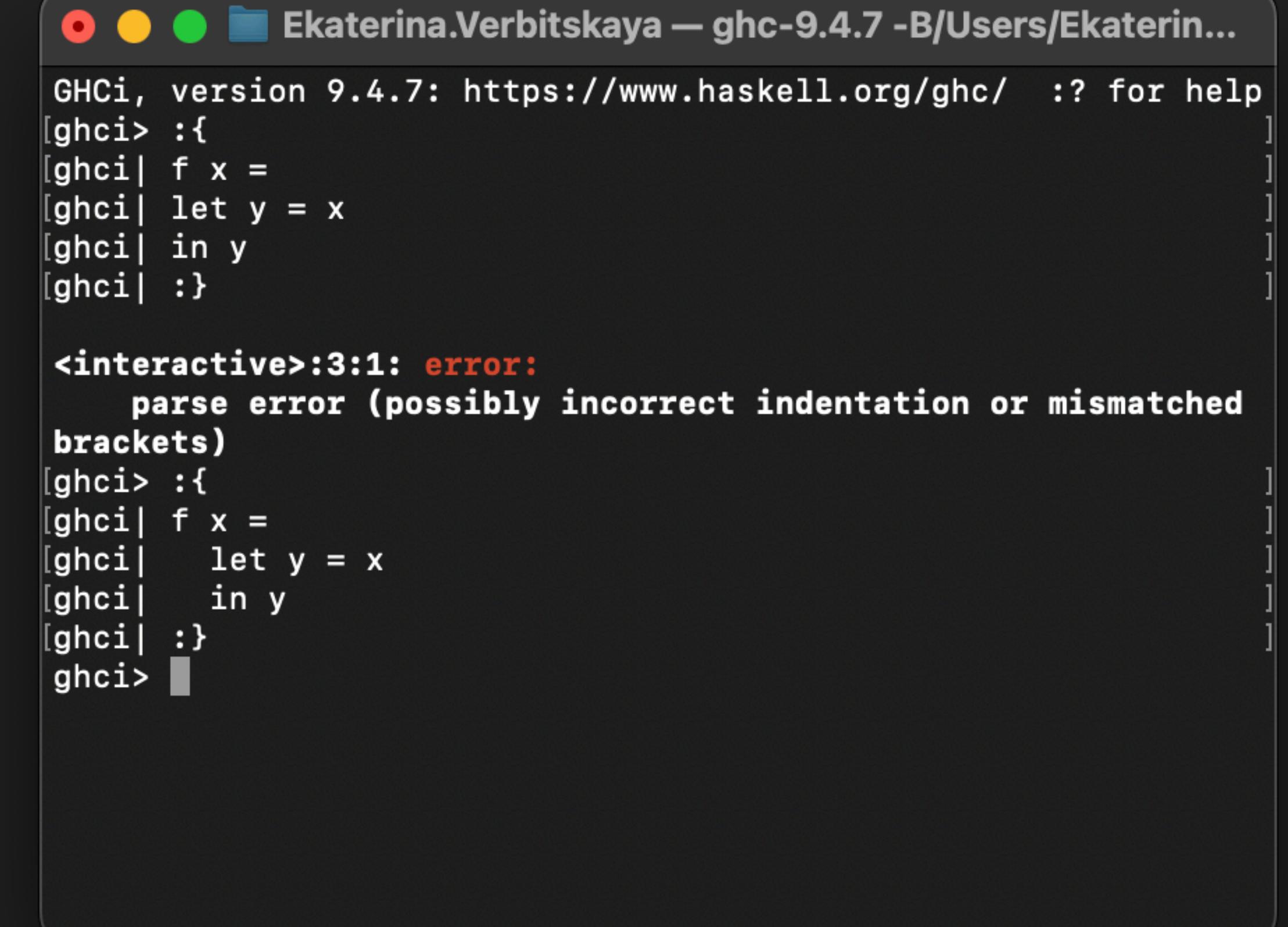


The screenshot shows a terminal window titled "Ekaterina.Verbitskaya — ghc-9.4.7 -B/Users/Ekaterin...". The terminal displays the following Haskell code:

```
[Ekaterina.Verbitskaya@NVC00653 ~ % ghci
GHCi, version 9.4.7: https://www.haskell.org/ghc/ :? for help
[ghci> cylinderArea r h = 2 * pi * r * h + 2 * pi * r^2
[ghci> :{
[ghci| cylinderArea r h =
[ghci|   let sideArea = 2 * pi * r * h
[ghci|     topArea = pi * r^2
[ghci|   in sideArea + 2 * topArea
[ghci| :}
[ghci> cylinderArea 1 1
12.566370614359172
ghci> ]
```

INDENTATION IS IMPORTANT

- ▶ Code which is part of some expression should be indented further in than the beginning of that expression



The screenshot shows a terminal window titled "Ekaterina.Verbitskaya — ghc-9.4.7 -B/Users/Ekaterin...". It displays a GHCi session. The user has typed:

```
[ghci> :{  
[ghci| f x =  
[ghci| let y = x  
[ghci| in y  
[ghci| :}  
  
<interactive>:3:1: error:  
    parse error (possibly incorrect indentation or mismatched  
    brackets)  
[ghci> :{  
[ghci| f x =  
[ghci|     let y = x  
[ghci|     in y  
[ghci| :}  
ghci> █
```

The terminal shows a parse error at line 3, column 1, indicating a problem with indentation or brackets. The user has then re-entered the code with correct indentation, and the session continues normally.

MORE INDENTATION EXAMPLES

- ▶ Align subsequent let-bindings

```
Ekaterina.Verbitskaya — ghc-9.4.7 -B/Users/Ekaterin...
GHCi, version 9.4.7: https://www.haskell.org/ghc/  :? for help
[ghci| :{
[ghci| f x = let y = x * 2
[ghci|           z = y + 13
[ghci|           in  z
[ghci| :}
[ghci| :{
[ghci| f x = let y = x * 2
[ghci|     z = y + 13
[ghci|     in z
[ghci| :}

<interactive>:8:3: error: parse error on input 'z'
[ghci| :{
[ghci| f x = let y = x * 2
[ghci|           z = y + 13
[ghci|           in z
[ghci| :}

<interactive>:10:15: error: parse error on input '='
ghci> █
```

EXPLICIT ; AND {} IN PLACE OF INDENTATION

- ▶ When you want to write a one-liner
- ▶ When you generate a Haskell program from another program

```
Ekaterina.Verbitskaya — ghc-9.4.7 -B/Users/Ekaterin...
GHCi, version 9.4.7: https://www.haskell.org/ghc/  :? for help
[ghci> :{
[ghci| f x = let { y = x * 2;
[ghci|   z = y + 13; } in z
[ghci| :}
[ghci> f x = let { y = x * 2; z = y + 13; } in z
ghci> █
```

EXPLICIT ; AND {} IN PLACE OF INDENTATION

- ▶ When you want to write a one-liner
- ▶ When you generate a Haskell program from another program
- ▶ When you miss C and Java too much

```
Ekaterina.Verbitskaya — ghc-9.4.7 -B/Users/Ekaterin...
GHCi, version 9.4.7: https://www.haskell.org/ghc/  :? for help
[ghci> :{
[ghci| f x = let { y = x * 2;
[ghci|   z = y + 13; } in z
[ghci| :}
[ghci> f x = let { y = x * 2; z = y + 13; } in z
ghci> █
```

LET-BINDING IS NOT ASSIGNMENT

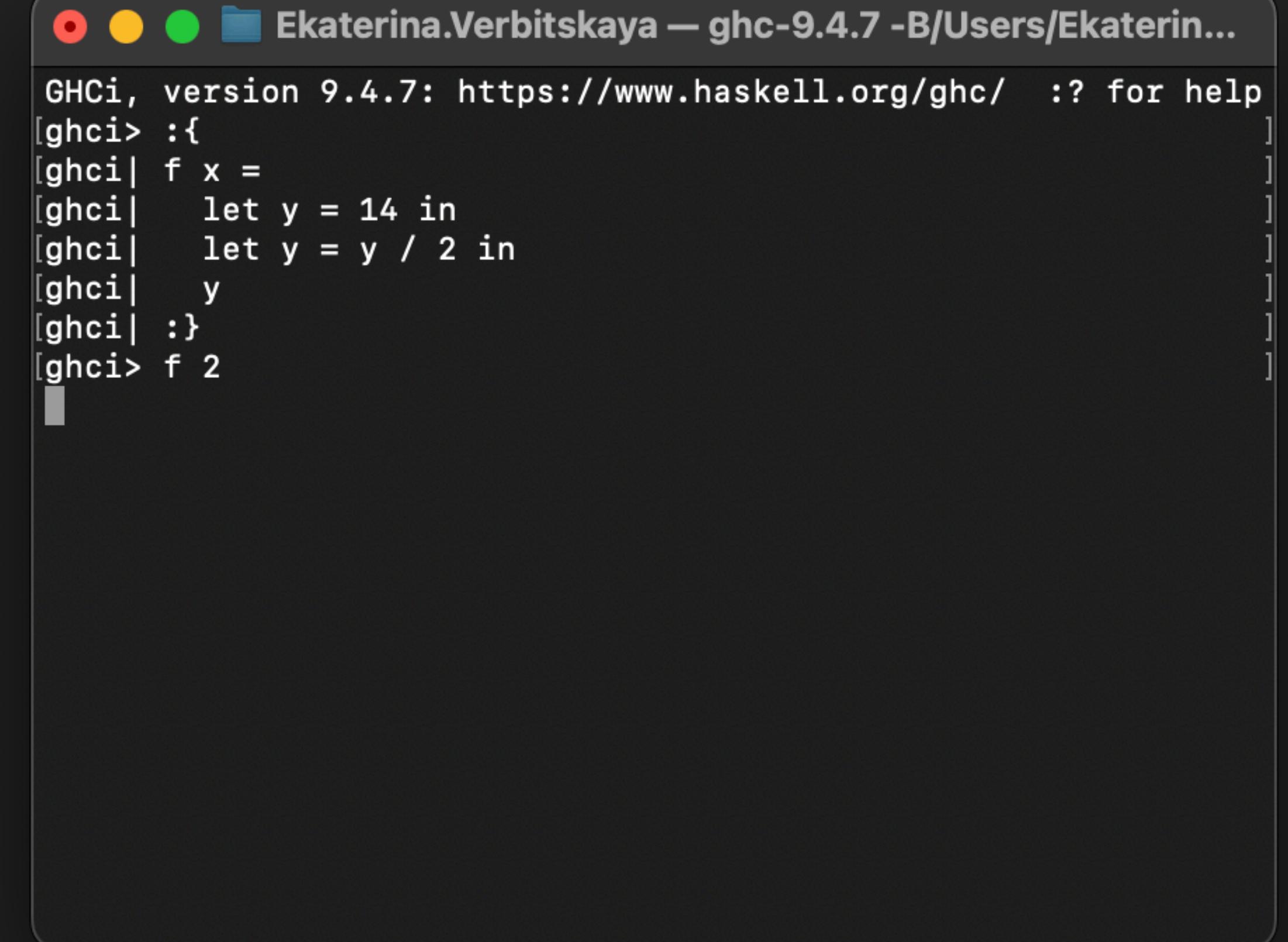
- ▶ Let-bindings only associate a variable name with the value
 - ▶ Variable name is only available in the body
 - ▶ Let-bindings can **shadow** each other
 - ▶ Multiple let-bindings of the same variable in the same block are prohibited

```
[Ekaterina.Verbitskaya@NVC00653 ~ % ghci
GHCi, version 9.4.7: https://www.haskell.org/ghc/  ?: for help
[ghci> :{
[ghci| f x =
[ghci|   let y = x * 2 in
[ghci|   let y = x / 2 in
[ghci|   y
[ghci| :}
[ghci> f 42
21.0
[ghci> :{
[ghci| g x =
[ghci|   let y = x * 2
[ghci|   y = x / 2
[ghci|   in  y
[ghci| :}

<interactive>:10:7: error:
  Conflicting definitions for 'y'
    Bound at: <interactive>:10:7
              <interactive>:11:7
```

SHADOWING IN LET-BINDINGS

- ▶ What will happen if we execute the code on the screenshot?

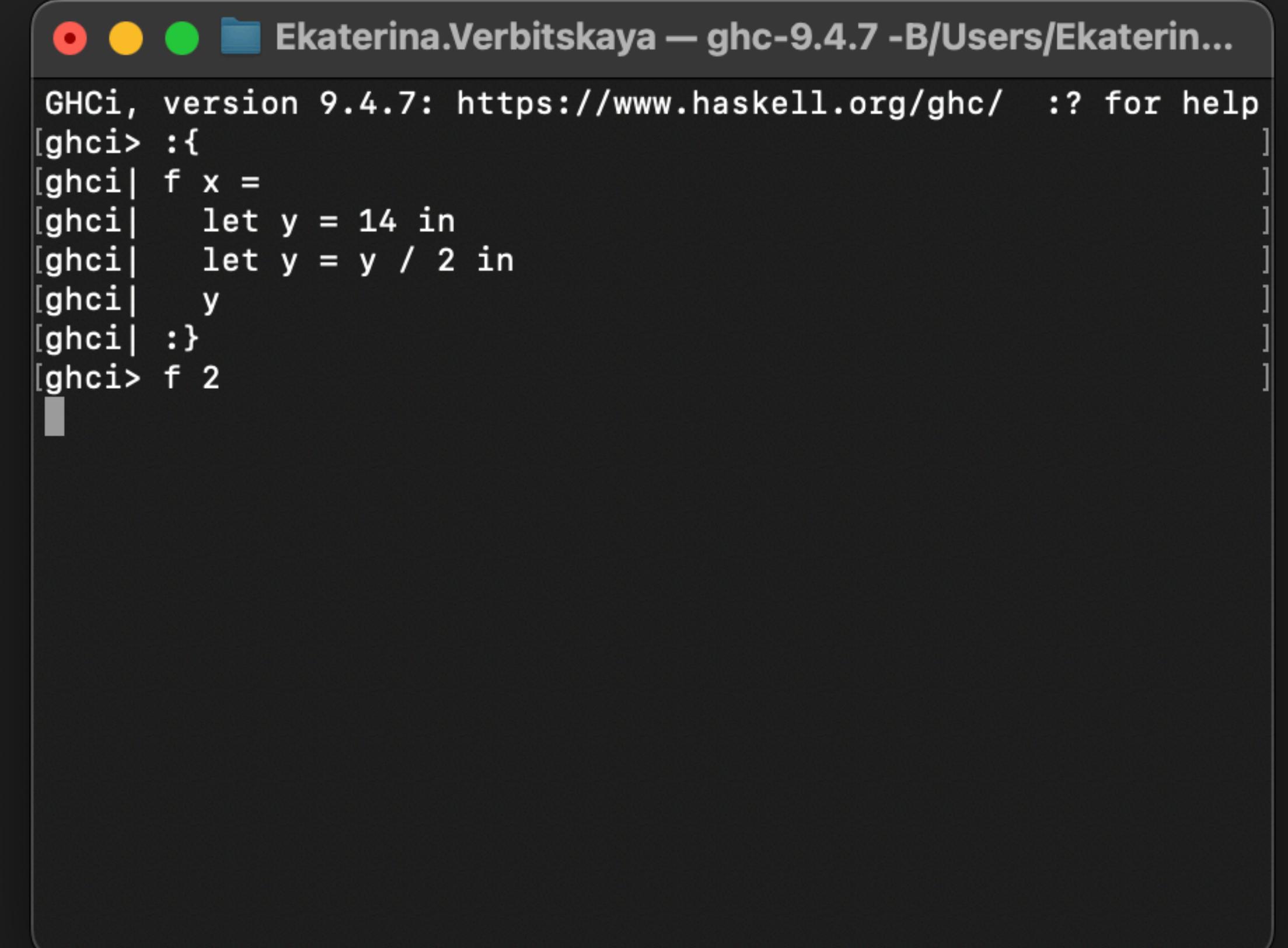


A screenshot of a terminal window titled "Ekaterina.Verbitskaya — ghc-9.4.7 -B/Users/Ekaterin...". The window shows a GHCi session with the following code and output:

```
GHCI, version 9.4.7: https://www.haskell.org/ghc/ :? for help
[ghci> :{
[ghci| f x =
[ghci|   let y = 14 in
[ghci|   let y = y / 2 in
[ghci|   y
[ghci| :}
[ghci> f 2
```

SHADOWING IN LET-BINDINGS

- ▶ What will happen if we execute the code on the screenshot?
- ▶ It diverges!
- ▶ The second let shadows the first binding
- ▶ In the body of `let y = y / 2`, `y` refers to itself!



A screenshot of the GHCi (Glasgow Haskell Compiler) interface. The title bar says "Ekaterina.Verbitskaya — ghc-9.4.7 -B/Users/Ekaterin...". The GHCi prompt shows the following code execution:

```
GHCI, version 9.4.7: https://www.haskell.org/ghc/  ?: for help
[ghci> :{
[ghci| f x =
[ghci|   let y = 14 in
[ghci|     let y = y / 2 in
[ghci|       y
[ghci|   }
[ghci> f 2
```

The code defines a function `f` that takes an argument `x`. Inside the function, there are two nested `let` bindings. The first `let` binds `y` to the value 14. The second `let` binds `y` to the result of dividing the previous `y` by 2. The final expression in the function body is `y`. When the function is called with argument 2, it enters an infinite loop because each time it tries to calculate `y / 2`, it uses the same `y` value from the inner `let` binding, which is never updated.

EXERCISES

- ▶ Implement functions:

- ▶ **triple** that triples the input

- ▶ **hypotenuse** that computes the length of a hypotenuse

- ▶ Use the function **sqrt** from Prelude

- ▶ **coneArea** which computes the area of a cone

- ▶ Use the constant **pi** from Prelude

```
Ekaterina.Verbitskaya — ghc-9.4.7 -B/Users/Ekaterin...
GHCi, version 9.4.7: https://www.haskell.org/ghc/  :? for help
[ghci> triple x = undefined
[ghci>
[ghci> hypotenuse a b = undefined
[ghci>
[ghci> coneArea r h = undefined
[ghci>
ghci> ]
```

OUTLINE

- ▶ Organization
- ▶ Introduction
- ▶ Syntax basics
- ▶ Simplest types

BOOLEAN

- ▶ Two values: True, False
- ▶ Usual operations over booleans

```
Ekaterina.Verbitskaya — ghc-9.4.7 -B/Users/Ekaterin...
GHCi, version 9.4.7: https://www.haskell.org/ghc/  :? for help
[ghci> True
True
[ghci> False
False
[ghci> True && False
False
[ghci> not False
True
[ghci> ((not True) && False) || True
True
[ghci> not True && False || True
True
[ghci> not (True && False || True)
False
ghci> █
```

IF-EXPRESSION

- ▶ Condition must be a Boolean value
- ▶ There must be both branches
- ▶ Both branches have to have the same type

```
Ekaterina.Verbitskaya — ghc-9.4.7 -B/Users/Ekaterin...
GHCi, version 9.4.7: https://www.haskell.org/ghc/  :? for help
[ghci> abs n = if n > 0 then n else (-n)
[ghci> abs 1
1
[ghci> abs (-1)
1
[ghci> if True then False
<interactive>:4:19: error:
    parse error (possibly incorrect indentation or mismatched
    brackets)
[ghci> if "True" then "False" else "True"
<interactive>:5:4: error:
  • Couldn't match type '[Char]' with 'Bool'
    Expected: Bool
    Actual: String
  • In the expression: "True"
    In the expression: if "True" then "False" else "True"
    In an equation for 'it': it = if "True" then "False" els
e "True"
ghci> █
```

CHARACTERS

- ▶ Single quotes
- ▶ Unicode
- ▶ You can use decimal, hexadecimal or octal notation
- ▶ Characters aren't numbers!
- ▶ But they can be ordered and you can get the next/previous character
- ▶ '\\' is \; '\"' is '

```
Ekaterina.Verbitskaya — ghc-9.4.7 -B/Users/Ekaterin...
GHCi, version 9.4.7: https://www.haskell.org/ghc/  :? for help
[ghci> 'Q'
'Q'
[ghci> 'Я'
'\1071'
[ghci> 'Я' == '\1071'
True
[ghci> 'A' == '\x41'
True
[ghci> 'A' < 'Z'
True
[ghci> 'A' + 1

<interactive>:6:5: error:
  • No instance for (Num Char) arising from a use of '+'
  • In the expression: 'A' + 1
    In an equation for 'it': it = 'A' + 1
[ghci> succ 'A'
'B'
ghci> ]
```

STRINGS

- ▶ Strings are lists of characters
- ▶ Strings are in double quotes
- ▶ A single-character string is not the same as the character

```
● ○ ● intro2fp — ghc-9.4.7 -B/Users/Ekaterina.Verbitskaya/... ]  
[ghci> :info String  
type String :: *  
type String = [Char]  
    -- Defined in 'GHC.Base'  
[ghci> ['H', 'e', 'l', 'l', 'o'] == "Hello"  
True  
[ghci> 'Hello'  
]  
  
<interactive>:3:1: error:  
• Syntax error on 'Hello'  
  Perhaps you intended to use TemplateHaskell or TemplateHaskellQuotes  
  • In the Template Haskell quotation 'Hello'  
[ghci> 'H' == "H"  
]  
  
<interactive>:4:8: error:  
• Couldn't match type '[Char]' with 'Char'  
  Expected: Char  
  Actual: String  
• In the second argument of '(==)', namely '"H"'  
  In the expression: 'H' == "H"  
  In an equation for 'it': it = 'H' == "H"
```

TUPLES

- ▶ Combine several values into one
 - ▶ From 2 to 62 elements (in GHC)
- ▶ Heterogeneous: elements are allowed to have different types
 - ▶ ("Hello", False, 42)
- ▶ Special functions for pairs
 - ▶ fst, snd
 - ▶ Swap

```
Ekaterina.Verbitskaya@NVC00653 intro2fp % ghci
GHCi, version 9.4.7: https://www.haskell.org/ghc/  ?: for help
[ghci> (13, 42, 777)
(13,42,777)
[ghci> ("Hello", False, 42)
("Hello",False,42)
[ghci> fst ('A', 1)
'A'
[ghci> snd ('A', 1)
1
[ghci> swap ('A', 1)

<interactive>:5:1: error:
  Variable not in scope: swap :: (Char, b0) -> t
[ghci> :module Data.Tuple
[ghci> swap ('A', 1)
(1,'A')
ghci> █
```

LISTS

- ▶ Homogeneous: all elements have the same type
- ▶ List literal: `['H', 'e', 'l', 'l', 'o']`
- ▶ Empty list: `[]`
- ▶ List constructor:
 - ▶ `'H' : ['e', 'l', 'l', 'o']`
 - ▶ `'H' : 'e' : 'l' : 'l' : 'o' : []`
 - ▶ `'H' : "ello"`
- ▶ List concatenation: `"He" ++ "llo"`

```

intro2fp — ghc-9.4.7 -B/Users/Ekaterina.Verbitskaya...
GHCI, version 9.4.7: https://www.haskell.org/ghc/  ?: for help
[ghci> ['H', 'e', 'l', 'l', 'o']
"Hello"
[ghci> []
[]
[ghci> 'H' : ['e', 'l', 'l', 'o']
"Hello"
[ghci> 'H' : 'e' : 'l' : 'l' : 'o' : []
"Hello"
[ghci> 'H' : "ello"
"Hello"
[ghci> "He" ++ "llo"
"Hello"
[ghci> "Hell" ++ 'o'
]

<interactive>:7:11: error:
  • Couldn't match expected type '[Char]' with actual type 'Char'
  • In the second argument of '(++)', namely "'o'"
    In the expression: "Hell" ++ 'o'
    In an equation for 'it': it = "Hell" ++ 'o'
ghci>

```

BABY'S FIRST TYPE ERRORS

- ▶ Static type system
 - ▶ The type of every expression is known and compile time
 - ▶ The type cannot change in run time.
- ▶ Type inference
 - ▶ Haskell can (almost often) infer types

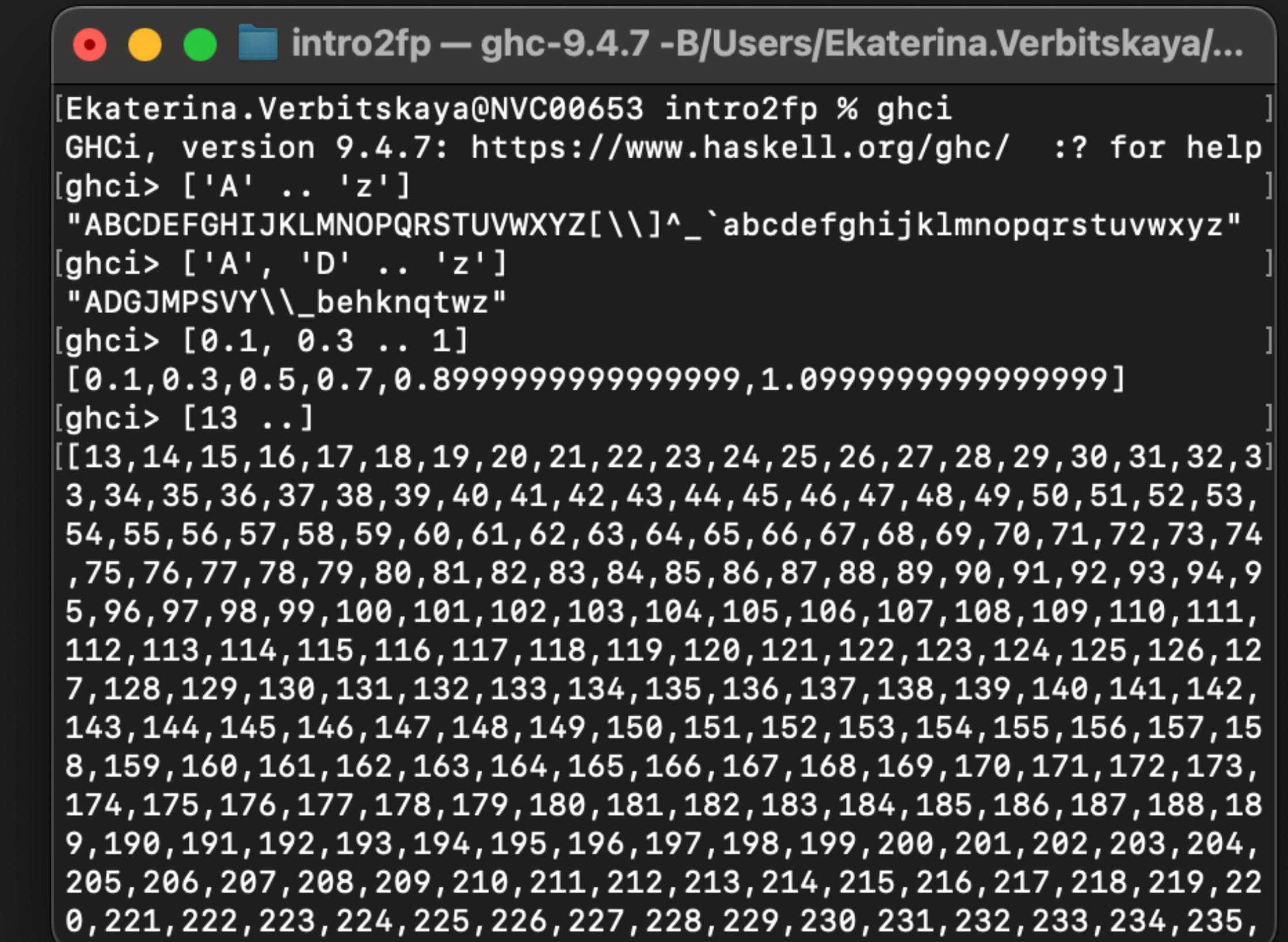
```
Ekaterina.Verbitskaya — ghc-9.4.7 -B/Users/Ekaterin...
GHCi, version 9.4.7: https://www.haskell.org/ghc/  :? for help
[ghci> "Hell" ++ 'o'

<interactive>:1:11: error:
  • Couldn't match expected type '[Char]' with actual type 'Char'
    • In the second argument of '(++)', namely "'o'"
      In the expression: "Hell" ++ 'o'
      In an equation for 'it': it = "Hell" ++ 'o'
[ghci> :t (++)
(++) :: [a] -> [a] -> [a]
[ghci> ['H', 'e', 'l', 'l'] ++ 'o'

<interactive>:3:25: error:
  • Couldn't match expected type '[Char]' with actual type 'Char'
    • In the second argument of '(++)', namely "'o"
      In the expression: ['H', 'e', 'l', 'l'] ++ 'o'
      In an equation for 'it': it = ['H', 'e', 'l', 'l', ....] ++ 'o'
[ghci> ['H', 'e', 'l', 'l'] ++ ['o']
"Hello"
```

RANGES

- ▶ Range from 'A' to 'z'
 - ▶ `['A' .. 'z']`
- ▶ Range from 'A' to 'z', every third character
 - ▶ `['A', 'D' .. 'z']`
- ▶ Don't use ranges with floating point numbers
 - ▶ `[0.1, 0.3 .. 1]`
- ▶ Range with no upper limit – infinite list
 - ▶ `[13 ..]`



The screenshot shows a terminal window titled "intro2fp — ghc-9.4.7 -B/Users/Ekaterina.Verbitskaya...". The terminal output is as follows:

```
[Ekaterina.Verbitskaya@NVC00653 intro2fp % ghci
GHCi, version 9.4.7: https://www.haskell.org/ghc/  ?: for help
[ghci> ['A' .. 'z']
"ABCDEFIGHIJKLMNOPQRSTUVWXYZ[\\"][_`abcdefghijklmnopqrstuvwxyz"
[ghci> ['A', 'D' .. 'z']
"ADGJMPSVY\\_behknqtzw"
[ghci> [0.1, 0.3 .. 1]
[0.1,0.3,0.5,0.7,0.8999999999999999,1.0999999999999999]
[ghci> [13 .. ]
[[13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,3
3,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,
54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74
,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,9
5,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,
112,113,114,115,116,117,118,119,120,121,122,123,124,125,126,12
7,128,129,130,131,132,133,134,135,136,137,138,139,140,141,142,
143,144,145,146,147,148,149,150,151,152,153,154,155,156,157,15
8,159,160,161,162,163,164,165,166,167,168,169,170,171,172,173,
174,175,176,177,178,179,180,181,182,183,184,185,186,187,188,18
9,190,191,192,193,194,195,196,197,198,199,200,201,202,203,204,
205,206,207,208,209,210,211,212,213,214,215,216,217,218,219,22
0,221,222,223,224,225,226,227,228,229,230,231,232,233,234,235,
```

EXERCISES

- ▶ Implement functions:
 - ▶ **even** that checks whether an `Int` is even
 - ▶ **odd** that checks whether an `Int` is odd
 - ▶ **evens** that returns an infinite list of even numbers
 - ▶ **f** that returns the first character of a string, along with its predecessor and successor
 - ▶ use **head**

```
Ekaterina.Verbitskaya — ghc-9.4.7 -B/Users/Ekaterin...
GHCi, version 9.4.7: https://www.haskell.org/ghc/  :? for help
[ghci> even n = undefined
[ghci>
[ghci> odd n = undefined
[ghci>
[ghci> evens = undefined
[ghci>
[ghci> f str = undefined
[ghci>
ghci> ]
```