

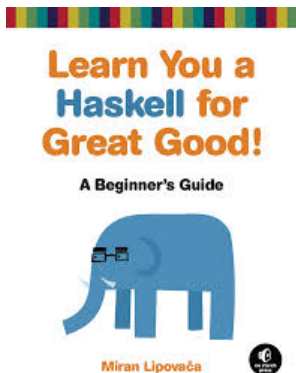
Learning Haskell

- 1 Using `ghci`. Interactivity, directives.
- 2 Expressions for each of the basic data types: `Integer`, `Float`, `Bool`, `Char`, `()` — unit.
- 3 Tuples. Lists: arithmetic sequences, list comprehensions.
- 4 Bindings. The `let` expression.
- 5 Simple functions. Anonymous functions, functions as data, special declaration syntax, patterns, cases, guards.
- 6 Using `ghc`. Writing, compiling, and running a main program, `interact`.

Erik Meijer, OSCON '09 [14min]  [↗](#)
Simon Peyton Jones, POP 2003, “Retrospective” [ppt]

Learning Haskell

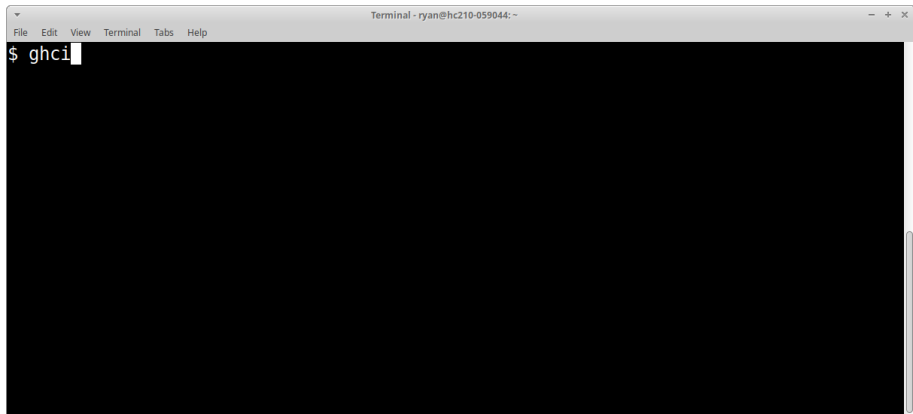
- All things Haskell: haskell.org ↗
- Tutorial: *Learn You a Haskell for Great Good!* ↗ by Miran Lipovača
- Searching for functions: [HOOGLE](http://Hoogle.org) ↗



Learning Haskell

- Higher-order functions
- Data structures
- Type inference

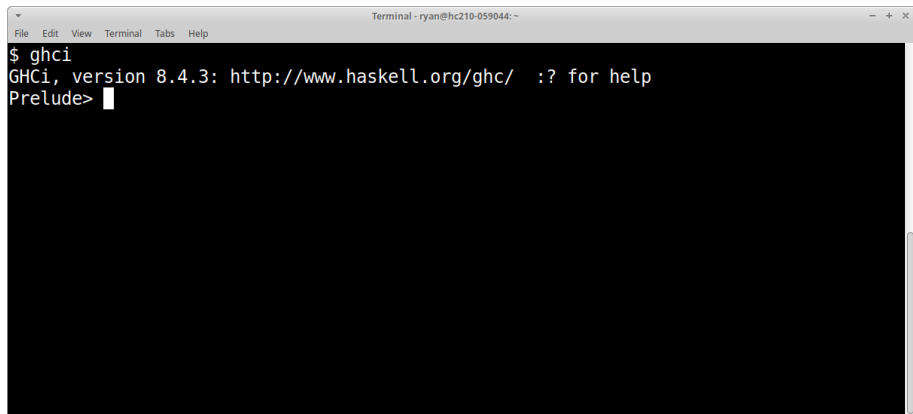
Using GHCi



A terminal window titled "Terminal - ryan@hc210-059044: ~" with a menu bar containing "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal content shows a prompt "\$" followed by the command "ghci" and a cursor. The rest of the terminal area is black.

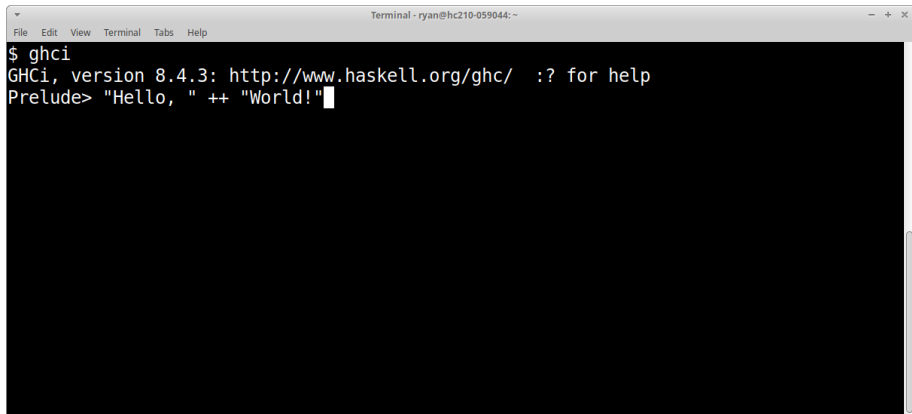
```
$ ghci
```

Using GHCi

A terminal window titled "Terminal - ryan@hc210-059044: ~" with a menu bar containing "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal content shows the command "\$ ghci" being executed, followed by the output "GHCi, version 8.4.3: http://www.haskell.org/ghc/ :? for help" and the prompt "Prelude>" with a cursor.

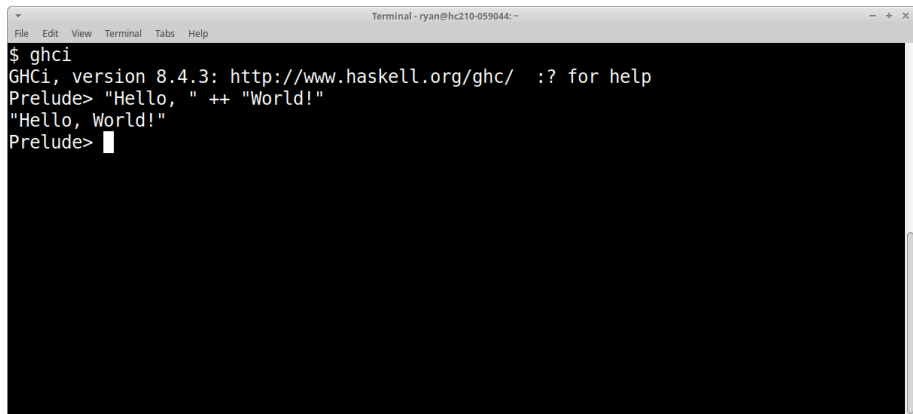
```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/ :? for help
Prelude> █
```

Using GHCi

A terminal window titled "Terminal - ryan@hc210-059044: ~" with a menu bar containing "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal content shows the command "\$ ghci" being executed, followed by the output "GHCi, version 8.4.3: http://www.haskell.org/ghc/ :? for help". The prompt "Prelude>" is shown, followed by the command "Hello, " ++ "World!" and a cursor. The terminal background is black with white text.

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/ :? for help
Prelude> "Hello, " ++ "World!"
```

Using GHCi

A terminal window titled "Terminal - ryan@hc210-059044: ~" with a menu bar containing "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal content is as follows:

```
$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Prelude> "Hello, " ++ "World!"
"Hello, World!"
Prelude> █
```

GHCI directives

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

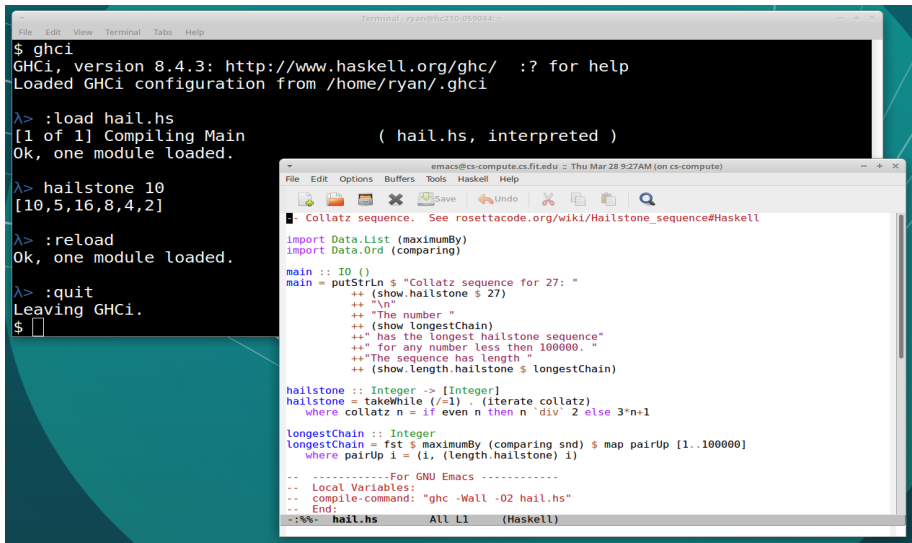
λ> :help
Commands available from the prompt:

<statement>          evaluate/run <statement>
:                    repeat last command
:{\n ..lines.. \n:}\n  multiline command
:add [*]<module> ...  add module(s) to the current target set
:browse[!] [[*]<mod>] display the names defined by module <mod>
                    (!: more details; *: all top-level names)
:cd <dir>            change directory to <dir>
:cmd <expr>         run the commands returned by <expr>::IO String
:complete <dom> [<rng>] <s> list completions for partial input string
:ctags[!] [<file>]  create tags file <file> for Vi (default: "tags")
```


GHCi directives

<code><statement></code>	evaluate/run <code><statement></code>
<code>:</code>	repeat last command
<code>:{\n ..lines.. \n:}\n</code>	multiline command
<code>:add [*]<module> ...</code>	add module(s) to the current target set
<code>:browse[!] [[*]<mod>]</code>	display the names defined by module <code><mod></code> (!: more details; *: all top-level names)
<code>:cd <dir></code>	change directory to <code><dir></code>
<code>:complete <dom> [<rng>] <s></code>	list completions for partial input string
<code>:help, :?</code>	display this list of commands
<code>:info[<name> ...]</code>	display information about the given names
<code>:kind <type></code>	show the kind of <code><type></code>
<code>:load [*]<module> ...</code>	load module(s) and their dependents
<code>:main [<arguments> ...]</code>	run the main function with the given arguments
<code>:module [+/-] [*]<mod> ...</code>	set the context for expression evaluation
<code>:quit</code>	exit GHCi
<code>:reload</code>	reload the current module set
<code>:type <expr></code>	show the type of <code><expr></code>
<code>:type +d <expr></code>	show the type of <code><expr></code> , defaulting type variables

Interactive development



```
Terminal: ryan@hc210-059044:~$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> :load hail.hs
[1 of 1] Compiling Main                ( hail.hs, interpreted )
Ok, one module loaded.

λ> hailstone 10
[10,5,16,8,4,2]

λ> :reload
Ok, one module loaded.

λ> :quit
Leaving GHCi.
$ █
```

```
emacs@cs-compute.cs.fit.edu ~ Thu Mar 28 9:27AM (on cs-compute)
File Edit Options Buffers Tools Haskell Help
Save Undo
Collatz sequence. See rosettacode.org/wiki/Hailstone_sequence#Haskell
import Data.List (maximumBy)
import Data.Ord (comparing)

main :: IO ()
main = putStrLn $ "Collatz sequence for 27: "
      ++ (show.hailstone $ 27)
      ++ "\n"
      ++ "The number "
      ++ (show longestChain)
      ++ " has the longest hailstone sequence"
      ++ " for any number less then 100000. "
      ++ "The sequence has length "
      ++ (show.length.hailstone $ longestChain)

hailstone :: Integer -> [Integer]
hailstone = takeWhile (/=1) . iterate collatz
  where collatz n = if even n then n `div` 2 else 3*n+1

longestChain :: Integer
longestChain = fst $ maximumBy (comparing snd) $ map pairUp [1..100000]
  where pairUp i = (i, (length.hailstone) i)

-- -----For GNU Emacs -----
-- Local Variables:
-- compile-command: "ghc -Wall -O2 hail.hs"
-- End:
-:~%- hail.hs All L1 (Haskell)
```

load — edit — reload — test; and repeat

Overview to skim quickly, or
Summary for later

Data

Integers

$\langle integer \rangle ::= \langle digit \rangle^+$

Tuples

$\langle tuple \rangle ::= (\langle expr \rangle, \langle expr \rangle)$

$::= (\langle expr \rangle, \langle expr \rangle, \langle expr \rangle)$

$::= (\langle expr \rangle, \langle expr \rangle, \langle expr \rangle, \langle expr \rangle)$

Lists

$\langle list \rangle ::= []$

$::= \langle expr \rangle : \langle list \rangle$

Functions

$\langle function \rangle ::= \backslash \langle name \rangle \rightarrow \langle expr \rangle$

Integers

$\langle integer \rangle ::= \langle digit \rangle^+$

Some Integers

0

42

3

7

5429723947

567

The Type of Integers

Integer

Integer

Integer

Integer

Integer

Integer

Tuples

$\langle tuple \rangle ::= (\langle expr \rangle, \langle expr \rangle)$

$::= (\langle expr \rangle, \langle expr \rangle, \langle expr \rangle)$

$::= (\langle expr \rangle, \langle expr \rangle, \langle expr \rangle, \langle expr \rangle)$

(5,3+4)

('z',1,1,1)

(True,2,False,3)

('a',True)

(4*7,('a',True),'p')

(2+3,2*3,2-3)

(Integer, Integer)

(Char, Integer, Integer, Integer)

(Bool, Integer, Bool, Integer)

(Char, Bool)

(Integer, (Char, Bool), Char)

(Integer, Integer, Integer)

Lists

$\langle list \rangle ::= []$

$::= \langle expr \rangle : \langle list \rangle$

1:1:[]

'a': 'b': 'c': []

():():[]

3:5:(3+6):7:[]

False:False:True:[]

True:False:True:[]

[Integer]
[Char] = String

[()]

[Integer]

[Bool]

[Bool]

Lists Have Special Syntax

`[1,1]`
`['a','b','c'] = "abc"`

`[(),()]`

`[3,5,3+6,7]`

`[False,False,True]`

`[True,False,True]`

Functions

$$\langle \textit{function} \rangle ::= \backslash \langle \textit{name} \rangle \rightarrow \langle \textit{expr} \rangle$$

lambda

formal parameter

body

```
\xs ->2:xs
|i ->max i 0
\s ->s++", "++s
|i ->i+2
\x ->sin (x+pi)
\c ->ord c
```


`[Integer] -> [Integer]`

`Integer -> Integer`

`String -> String`

`Integer -> Integer`

`Double -> Double`

`Char -> Integer`

Data, all data exists apart from names. Data

```
meaningOfLive = 42
```

```
myVerySpecialPair = ('a', 0)
```

```
myShortList = [True, False, True]
```

```
add2 = \ i -> i+2
```

```
cons2 = \ xs -> 2:xs
```

```
double = \ s -> s ++ ", " ++ s
```

Function Declarations Have Special Syntax

```
cons2 xs = 2:xs
```

```
cutOff i = max i 0
```

```
doubleWord s = s ++ ", " ++ s
```

```
add2 i = i+2
```

```
f x = sin (x+pi)
```

```
toOrd c = ord c
```

Complex Canonical Values

```
(2,('a',5), "abc", True)  :: (Integer, (Char,Integer))
([1],('a',5), "abc", True, "xyz")  :: ([Integer], (Char,Integer))
(\i->i+2 , \x->sind(x+pi) )  :: (Integer->Integer, Integer->Integer)
```

```
[(1,True,"abc"), (2,True,"mno"), (3,True,"xyz")] :: [(Integer, Boolean, String)]
```

```
\ i -> (i,i)  :: Integer -> (Integer,Integer)
```

```
\ p -> (fst p + snd p, fst p * snd p, fst p - snd p)
```

End of the quick overview
(proceed to basics), or
End of summary
(proceed to writing a program with GHC)

The Basics

- ① Primitive: Integer, floating-point, character, boolean, unit
- ② Structures: Tuple, lists, functions
- ③ Text: list of character

Integer

143

succ 34

3+4

5*7

9-4

negate 8

*---- watch out: 3 * -8 NO!*

3 * (-8)

Integer

```
min 17 34  
max 17 34
```

```
div 24 7  
24 'div' 7  
24 'rem' 7  
mod 36 5  
----> 1
```

```
quot 36 5  
----> 7
```

```
div 36 5  
----> 7
```

```
quot 36 (-5)  
----> -7
```

```
div 36 (-5)  
----> -8
```


Integer

```
quotRem 36 (-5)  
----> (-7, 1)
```

```
divMod 36 -5  
----> (-8, -4)
```

Floating-Point

```
--- pi, exp, sqrt, log, (**), sin, tan, cos,  
--- truncate, ceiling, round, truncate, floor  
3.2 + 43.1  
5.2 * 43.2345236  
9.9 - 2.3  
2345.2345 / 34.34  
34.4 ^ 34  
4254 ** 4.345  
sqrt (4.59)  
sin (1.7172)
```

Char

'a'

```
--- :browse Data.Char
ord  :: Char -> Int
chr  :: Int  -> Char
digitToInt  :: Char -> Int
intToDigit  :: Int  -> Char
toLower     :: Char -> Char
toUpper     :: Char -> Char
isAlphaNum  :: Char -> Bool
isDigit     :: Char -> Bool
isAlpha     :: Char -> Bool
isLower     :: Char -> Bool
isUpper     :: Char -> Bool
isSpace     :: Char -> Bool
```

Bool

```
False
```

```
True
```

```
False && True
```

```
False || True
```

```
not True
```

```
-- otherwise is defined to be the value True
```

```
-- for the purposes of making guards more readable
```

```
otherwise
```

```
if True then 4 else 5
```

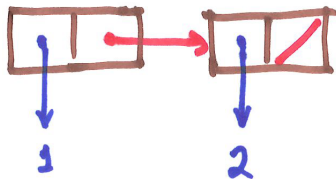
```
-- important predicates
```

```
-- (==), (/=), (<), (>=), (>), (<=), compare
```

Tuples

```
(2 'a')  
(4.34, 'a', 456)  
(True, (), 1)  
(2, "ab", 2*2, 5.0)  
( 2 ) -- not a tuple  
((())) -- unit  
  
fst (2, 'b')  
snd (1, 'a')  
  
(2, ('a', 3, 4), "abcd")  
((2, 3, 4), (True, 3.3))
```

What is a List?



How do you make a List in Haskell?

Two constructors “nil” and “cons”

```
[]
```

```
1 : []
```

```
1 : (2 : [])
```

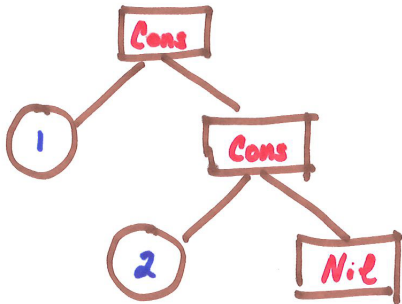
```
1 : (2 : (3 : []))
```

```
1 : (2 : (3 : 4 : []))
```

How do you make a List in Haskell?

“Cons” is right associative; and, anyway, lists have special syntax.

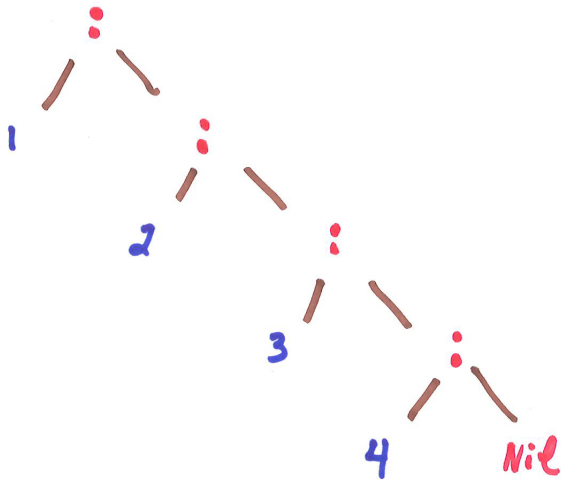
<code>[]</code>	<code>[]</code>	<code>[]</code>
<code>1 : []</code>	<code>1 : []</code>	<code>[1]</code>
<code>1 : (2 : [])</code>	<code>1 : 2 : []</code>	<code>[1, 2]</code>
<code>1 : (2 : (3 : []))</code>	<code>1 : 2 : 3 : []</code>	<code>[1, 2, 3]</code>
<code>1 : (2 : (3 : 4 : []))</code>	<code>1 : 2 : 3 : 4 : []</code>	<code>[1, 2, 3, 4]</code>



Cons 1 (cons 2 Nil)

1 : 2 : Nil

[1, 2]



[1, 2, 3, 4]

What can you do with a list?

```
head [1,2,3,4]
```

```
tail [1,2,3,4]
```

```
null [1,2,3,4]
```

Lists are polymorphic but homogeneous.

```
[1,2,3,4]
```

```
['a', 'b', 'c']
```

```
[(1, 'a'), (2, 'b'), (3, 'c')]
```

```
[ [], [1], [1,2,3,4], [2,3] ]
```

There are no mutable arrays in Haskell, so lists are the “go to” data structure for collections.

Every list function one can think up has been pre-defined in Haskell. See [lists](#) at Wiki Haskell.

```
length
(++)    -- append
elem    -- member
(!!)    -- get element
concat  -- flatten
last    init
splitAt
take    drop
sort    nub
reverse
sum     product
minimum maximum
and     or
all     any
```

List and Arithmetic Sequences

```
[-1,-1 .. 0]
[3,2 .. 8]
[0,2 .. 1]
[1,1 .. 1]
[3,3 .. (-4)]
[2,1 .. (-4)]
[3,3 .. (-12)]
[-1,-1 ..      8]
[-6,-6 .. 12]
[1,2..(-12)]
[-6,-5 .. (-4)]
[1,0 .. 5]
[-6,-6 .. 8]
[-1,-2 .. (-4)]
[1,2 .. 5]
```

List Comprehension

Video; 31 minutes

Presenter: E Meijer

based on Hutton's book, chapter 5

Channel 9 lectures at YouTube

List Comprehension

Defining sets by their properties is sometimes known as set comprehension and found often in mathematical texts. For example,

$$\{x \in \mathbb{R} \mid x > 0\}$$

In mathematical notation we write

$$\{x^2 \mid x \in \{1, 2, \dots, 5\}\}$$

to mean the set $\{1, 4, 9, 16, 25\}$.

In Haskell a similar syntax is used for lists. The special square brackets syntax is expanded to include generators and filters.

Haskell List Comprehension

A list comprehension has the form:

$$[\text{expr} \mid \text{qual}_1, \dots, \text{qual}_n]$$

where $1 \geq n$

There are three types qualifiers

- generators of the form $\text{pat} \leftarrow \text{expr}$, where p is a pattern (see Section 3.17) of type t and e is an expression of type $[t]$
- local bindings that provide new definitions for use in the generated expression expr or subsequent boolean guards and generators
- boolean guards, which are arbitrary expressions of type `Bool`.

See the section on List Comprehensions [↗](#) in the Haskell 2010 Language Report [↗](#).

Also, there are intriguing language extensions:

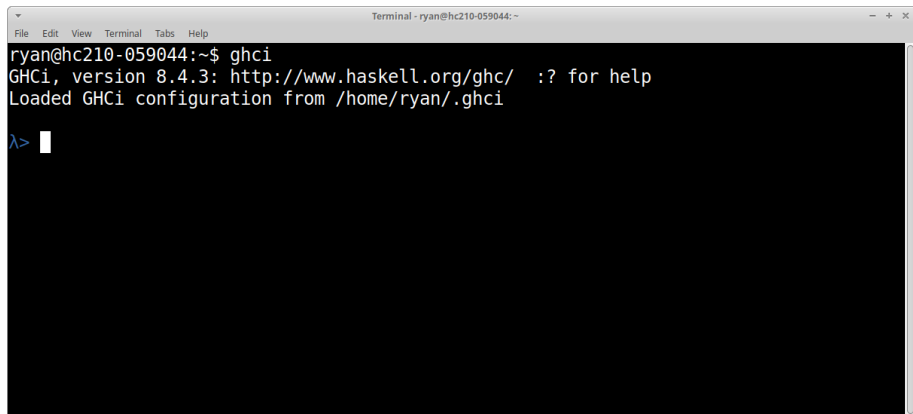
- `ParallelListComp`. Syntax: $[\text{expr} \mid \text{qualifier}, \dots \mid \text{qualifier}, \dots]$
- `TransformListComp`. Three new keywords: `group`, `by`, and `using`.

See a paper [↗](#) by Simon Peyton Jones.

Upcoming script

```
[ n | n <- [1..5] ]  
[ (n+2,5*n) | n <- [1..3] ]    -- list of pairs  
:mod + Data.Char  
[ toUpper c | c <- "one fish" ]
```

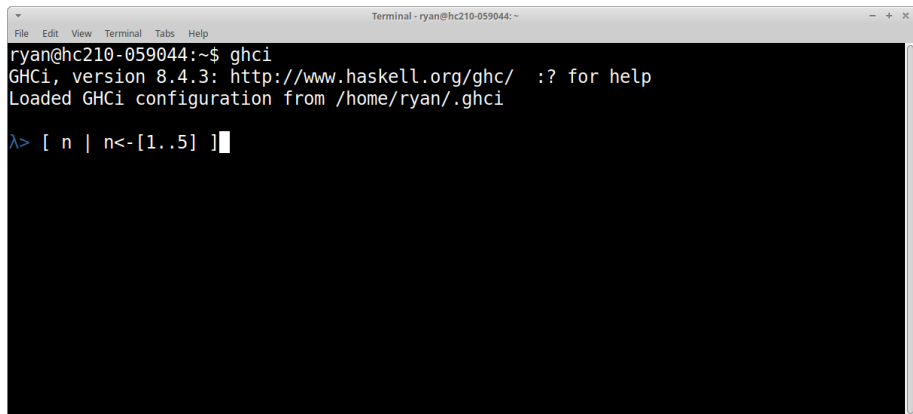
List Comprehension

A terminal window titled "Terminal - ryan@hc210-059044: ~" with a menu bar containing "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal output shows the command "ghci" being executed, followed by the GHCi version (8.4.3), a URL for help, and the path to the configuration file. The prompt changes from "\$" to "λ>".

```
ryan@hc210-059044:~$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> 
```

List Comprehension

A terminal window titled "Terminal - ryan@hc210-059044: ~" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows the following text:

```
ryan@hc210-059044:~$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> [ n | n<-[1..5] ]
```

List Comprehension

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> [ n | n<-[1..5] ]  
[1,2,3,4,5]  
  
λ> █
```

List Comprehension

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> [ n | n<-[1..5] ]  
[1,2,3,4,5]  
  
λ> [ (n+2,5*n) | n<-[1..3] ]
```

List Comprehension

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> [ n | n<-[1..5] ]  
[1,2,3,4,5]  
  
λ> [ (n+2,5*n) | n<-[1..3] ]  
[(3,5),(4,10),(5,15)]  
  
λ> █
```

List Comprehension

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> [ n | n<-[1..5] ]  
[1,2,3,4,5]  
  
λ> [ (n+2,5*n) | n<-[1..3] ]  
[(3,5),(4,10),(5,15)]  
  
λ> [ (n+2,5*n) | n<-[1..3] ]
```


List Comprehension

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> [ n | n<-[1..5] ]  
[1,2,3,4,5]  
  
λ> [ (n+2,5*n) | n<-[1..3] ]  
[(3,5),(4,10),(5,15)]  
  
λ> [ (n+2,5*n) | n<-[1..3] ]  
[(3,5),(4,10),(5,15)]  
  
λ> █
```

List Comprehension

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> [ n | n<-[1..5] ]  
[1,2,3,4,5]  
  
λ> [ (n+2,5*n) | n<-[1..3] ]  
[(3,5),(4,10),(5,15)]  
  
λ> [ (n+2,5*n) | n<-[1..3] ]  
[(3,5),(4,10),(5,15)]  
  
λ> [ (n+2,5*n) | n<-[1..3] ]
```

List Comprehension

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> [ n | n<-[1..5] ]  
[1,2,3,4,5]  
  
λ> [ (n+2,5*n) | n<-[1..3] ]  
[(3,5),(4,10),(5,15)]  
  
λ> [ (n+2,5*n) | n<-[1..3] ]  
[(3,5),(4,10),(5,15)]  
  
λ> [ (n+2,5*n) | n<-[1..3] ]  
[(3,5),(4,10),(5,15)]  
  
λ> █
```

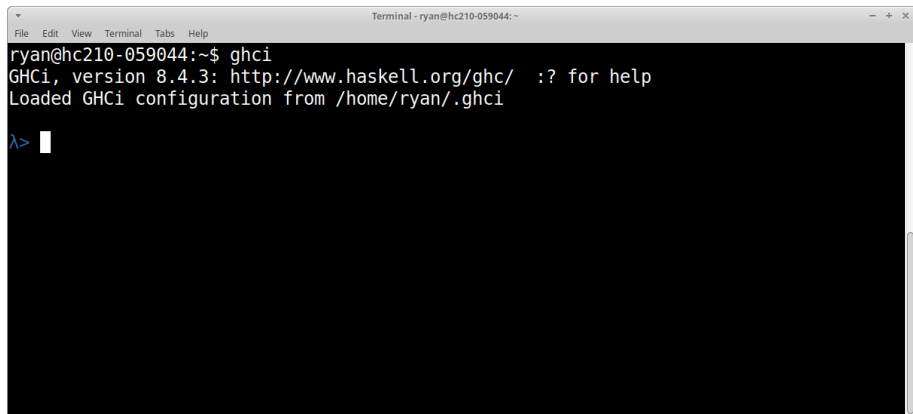
List Comprehension

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> [ n | n<-[1..5] ]  
[1,2,3,4,5]  
  
λ> [ (n+2,5*n) | n<-[1..3] ]  
[(3,5),(4,10),(5,15)]  
  
λ> [ (n+2,5*n) | n<-[1..3] ]  
[(3,5),(4,10),(5,15)]  
  
λ> [ (n+2,5*n) | n<-[1..3] ]  
[(3,5),(4,10),(5,15)]  
  
λ> :mod + Data.Char  
  
λ> [ toUpper c | c <- "one fish" ]
```

List Comprehension

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
[1,2,3,4,5]  
  
λ> [ (n+2,5*n) | n<-[1..3] ]  
[(3,5),(4,10),(5,15)]  
  
λ> [ (n+2,5*n) | n<-[1..3] ]  
[(3,5),(4,10),(5,15)]  
  
λ> [ (n+2,5*n) | n<-[1..3] ]  
[(3,5),(4,10),(5,15)]  
  
λ> :mod + Data.Char  
  
λ> [ toUpper c | c <- "one fish" ]  
"ONE FISH"  
  
λ> |
```

List Comprehension (Multiple Generators)

A terminal window titled "Terminal - ryan@hc210-059044: ~" with a menu bar containing "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal output shows the command "ghci" being executed, followed by the GHCi version (8.4.3), a URL for help, and the path to the configuration file. The prompt changes from a shell prompt to a lambda prompt.

```
ryan@hc210-059044:~$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> █
```

List Comprehension (Multiple Generators)

A terminal window titled "Terminal - ryan@hc210-059044: ~" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows the following text:

```
ryan@hc210-059044:~$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> [ (i,j) | i<-[1..3], j<-[1..2] ]
```

List Comprehension (Multiple Generators)

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> [ (i,j) | i<-[1..3], j<-[1..2] ]  
[(1,1),(1,2),(2,1),(2,2),(3,1),(3,2)]  
  
λ> █
```


List Comprehension (Multiple Generators)

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> [ (i,j) | i<-[1..3], j<-[1..2] ]  
[(1,1),(1,2),(2,1),(2,2),(3,1),(3,2)]  
  
λ> [ (i,c) | i<-[1..2], c<-"abc" ]
```

List Comprehension (Multiple Generators)

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> [ (i,j) | i<-[1..3], j<-[1..2] ]  
[(1,1),(1,2),(2,1),(2,2),(3,1),(3,2)]  
  
λ> [ (i,c) | i<-[1..2], c<- "abc" ]  
[(1,'a'),(1,'b'),(1,'c'),(2,'a'),(2,'b'),(2,'c')]  
  
λ> █
```

List Comprehension (Multiple Generators)

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> [ (i,j) | i<-[1..3], j<-[1..2] ]  
[(1,1),(1,2),(2,1),(2,2),(3,1),(3,2)]  
  
λ> [ (i,c) | i<-[1..2], c<- "abc" ]  
[(1,'a'),(1,'b'),(1,'c'),(2,'a'),(2,'b'),(2,'c')]  
  
λ> [ (i,c) | c<- "abc", j<-[1..2] ]
```

List Comprehension (Multiple Generators)

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> [ (i,j) | i<-[1..3], j<-[1..2] ]  
[(1,1),(1,2),(2,1),(2,2),(3,1),(3,2)]  
  
λ> [ (i,c) | i<-[1..2], c<- "abc" ]  
[(1,'a'),(1,'b'),(1,'c'),(2,'a'),(2,'b'),(2,'c')]  
  
λ> [ (i,c) | c<- "abc", j<-[1..2] ]  
  
<interactive>:3:4: error: Variable not in scope: i  
  
λ> [ (i,c) | c<- "abc", i<-[1..2] ]
```

List Comprehension (Multiple Generators)

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> [ (i,j) | i<-[1..3], j<-[1..2] ]
[(1,1),(1,2),(2,1),(2,2),(3,1),(3,2)]

λ> [ (i,c) | i<-[1..2], c<-["abc"] ]
[(1,'a'),(1,'b'),(1,'c'),(2,'a'),(2,'b'),(2,'c')]

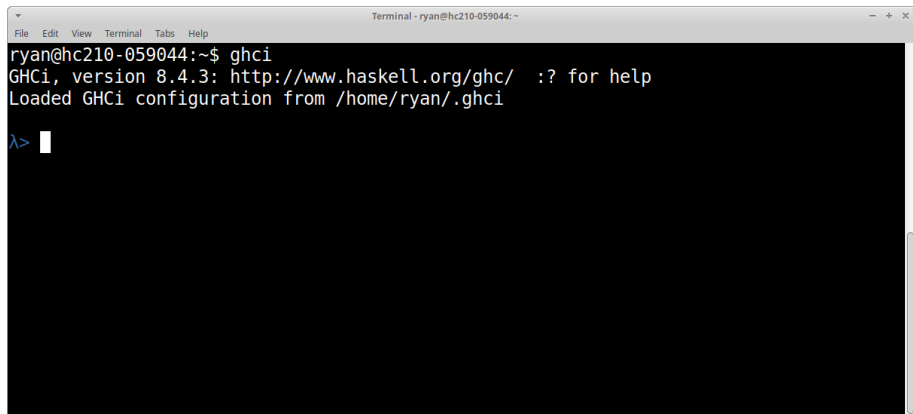
λ> [ (i,c) | c<-["abc"], j<-[1..2] ]

<interactive>:3:4: error: Variable not in scope: i

λ> [ (i,c) | c<-["abc"], i<-[1..2] ]
[(1,'a'),(2,'a'),(1,'b'),(2,'b'),(1,'c'),(2,'c')]

λ> █
```

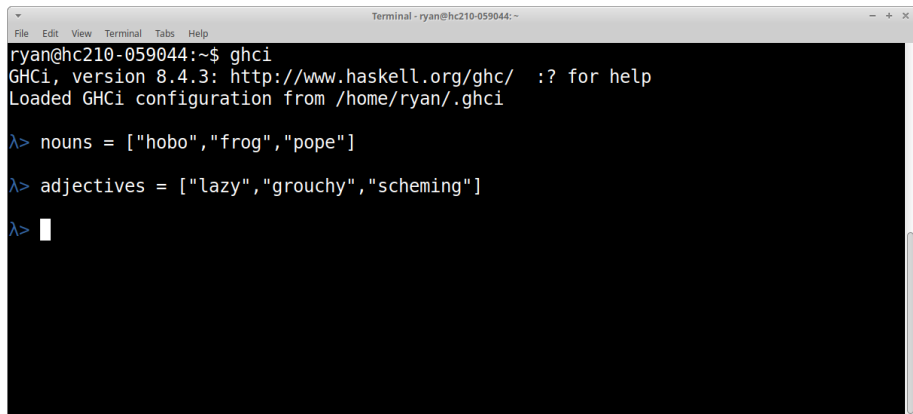
List Comprehension (Multiple Generators)

A terminal window titled "Terminal - ryan@hc210-059044: ~" with a menu bar containing "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal output shows the command "ghci" being executed, followed by the GHCi version (8.4.3), a URL for help, and the path to the configuration file. The prompt changes from a shell prompt to a lambda prompt.

```
ryan@hc210-059044:~$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> █
```

List Comprehension (Multiple Generators)



```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> nouns = ["hobo","frog","pope"]  
λ> adjectives = ["lazy","grouchy","scheming"]  
λ> █
```

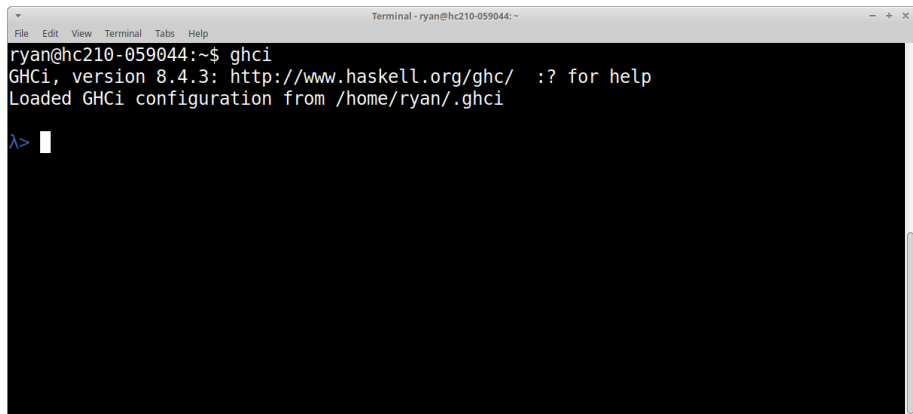
List Comprehension (Multiple Generators)

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> nouns = ["hobo","frog","pope"]  
  
λ> adjectives = ["lazy","grouchy","scheming"]  
  
λ> [adjective++" "++noun | adjective<-adjectives, noun<-nouns]
```


List Comprehension (Multiple Generators)

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> nouns = ["hobo","frog","pope"]  
  
λ> adjectives = ["lazy","grouchy","scheming"]  
  
λ> [adjective++" ++noun | adjective<-adjectives, noun<-nouns]  
["lazy hobo","lazy frog","lazy pope","grouchy hobo","grouchy frog","grouchy pope","sc  
heming hobo","scheming frog","scheming pope"]  
  
λ> █
```

List Comprehension (Infinite Lists, Nested)

A terminal window titled "Terminal - ryan@hc210-059044: ~" with a menu bar containing "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal text shows the execution of the "ghci" command, the GHCi version (8.4.3), the GHCi website URL, and the loaded configuration file path. The prompt is now "λ>" with a cursor.

```
ryan@hc210-059044:~$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> █
```

List Comprehension (Infinite Lists, Nested)



```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> take 8 [ (i,j) | i<-[1,2], j<-[i..] ]
```

List Comprehension (Infinite Lists, Nested)

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> take 8 [ (i,j) | i<-[1,2], j<-[i..] ]  
[(1,1),(1,2),(1,3),(1,4),(1,5),(1,6),(1,7),(1,8)]  
  
λ> █
```

List Comprehension (Infinite Lists, Nested)

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> take 8 [ (i,j) | i<-[1,2], j<-[i..] ]  
[(1,1),(1,2),(1,3),(1,4),(1,5),(1,6),(1,7),(1,8)]  
  
λ> take 8 [ (i,j) | j<-[3..], i<-[1,2] ]
```

List Comprehension (Infinite Lists, Nested)

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> take 8 [ (i,j) | i<-[1,2], j<-[i..] ]  
[(1,1),(1,2),(1,3),(1,4),(1,5),(1,6),(1,7),(1,8)]  
  
λ> take 8 [ (i,j) | j<-[3..], i<-[1,2] ]  
[(1,3),(2,3),(1,4),(2,4),(1,5),(2,5),(1,6),(2,6)]  
  
λ> █
```

List Comprehension (Infinite Lists, Nested)

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> take 8 [ (i,j) | i<-[1,2], j<-[i..] ]  
[(1,1),(1,2),(1,3),(1,4),(1,5),(1,6),(1,7),(1,8)]  
  
λ> take 8 [ (i,j) | j<-[3..], i<-[1,2] ]  
[(1,3),(2,3),(1,4),(2,4),(1,5),(2,5),(1,6),(2,6)]  
  
λ> take 4 [ [ (i,j) | i <- [1,2] ] | j <- [1..] ]
```

List Comprehension (Infinite Lists, Nested)

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> take 8 [ (i,j) | i<-[1,2], j<-[i..] ]  
[(1,1),(1,2),(1,3),(1,4),(1,5),(1,6),(1,7),(1,8)]  
  
λ> take 8 [ (i,j) | j<-[3..], i<-[1,2] ]  
[(1,3),(2,3),(1,4),(2,4),(1,5),(2,5),(1,6),(2,6)]  
  
λ> take 4 [ [ (i,j) | i <- [1,2] ] | j <- [1..] ]  
[[ (1,1), (2,1) ], [ (1,2), (2,2) ], [ (1,3), (2,3) ], [ (1,4), (2,4) ]]  
  
λ> █
```

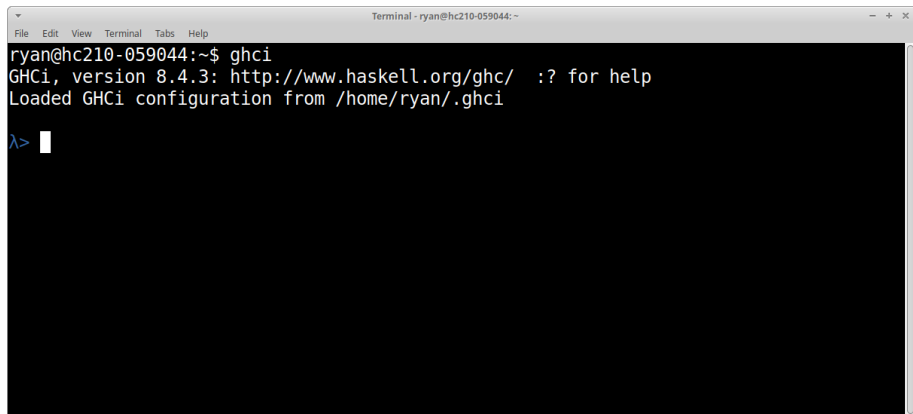

List Comprehension (Infinite Lists, Nested)

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> take 8 [ (i,j) | i<-[1,2], j<-[1..] ]  
[(1,1),(1,2),(1,3),(1,4),(1,5),(1,6),(1,7),(1,8)]  
  
λ> take 8 [ (i,j) | j<-[3..], i<-[1,2] ]  
[(1,3),(2,3),(1,4),(2,4),(1,5),(2,5),(1,6),(2,6)]  
  
λ> take 4 [ [ (i,j) | i <- [1,2] ] | j <- [1..] ]  
[[ (1,1), (2,1) ], [ (1,2), (2,2) ], [ (1,3), (2,3) ], [ (1,4), (2,4) ]]  
  
λ> take 8 $ concat [ [ (i,j) | i <- [1,2] ] | j <- [1..] ]
```

List Comprehension (Infinite Lists, Nested)

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> take 8 [ (i,j) | i<-[1,2], j<-[i..] ]  
[(1,1),(1,2),(1,3),(1,4),(1,5),(1,6),(1,7),(1,8)]  
  
λ> take 8 [ (i,j) | j<-[3..], i<-[1,2] ]  
[(1,3),(2,3),(1,4),(2,4),(1,5),(2,5),(1,6),(2,6)]  
  
λ> take 4 [ [ (i,j) | i <- [1,2] ] | j <- [1..] ]  
[[ (1,1), (2,1) ], [ (1,2), (2,2) ], [ (1,3), (2,3) ], [ (1,4), (2,4) ]]  
  
λ> take 8 $ concat [ [ (i,j) | i <- [1,2] ] | j <- [1..] ]  
[(1,1),(2,1),(1,2),(2,2),(1,3),(2,3),(1,4),(2,4)]  
  
λ> █
```

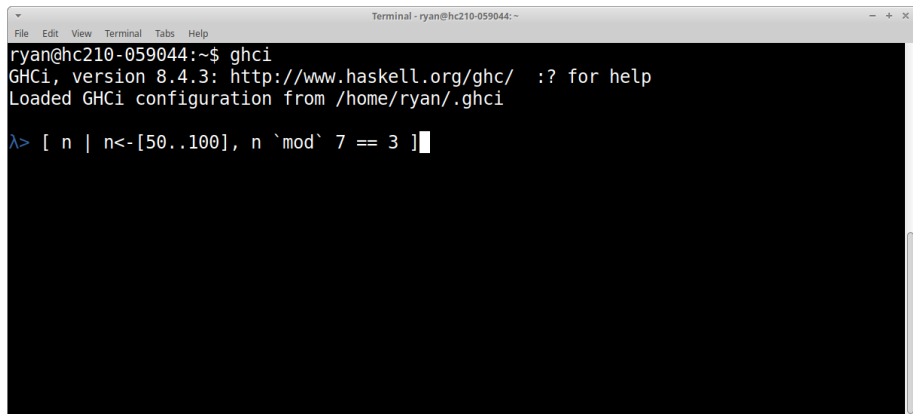
List Comprehension (Guards aka Filters)

A terminal window titled "Terminal - ryan@hc210-059044: ~" with a menu bar containing "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal output shows the command "ghci" being executed, followed by the GHCi version (8.4.3), a URL for help, and the path to the configuration file. The prompt changes from a shell prompt to a lambda prompt.

```
ryan@hc210-059044:~$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> █
```

List Comprehension (Guards aka Filters)



```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> [ n | n<-[50..100], n `mod` 7 == 3 ]
```

List Comprehension (Guards aka Filters)

A terminal window titled "Terminal - ryan@hc210-059044: ~" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The prompt is "ryan@hc210-059044:~\$ ghci". The output shows "GHCi, version 8.4.3: http://www.haskell.org/ghc/ :? for help" and "Loaded GHCi configuration from /home/ryan/.ghci". A list comprehension is entered: "λ> [n | n<-[50..100], n `mod` 7 == 3]". The result is "[52,59,66,73,80,87,94]". The prompt "λ>" is followed by a cursor.

```
ryan@hc210-059044:~$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/ :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> [ n | n<-[50..100], n `mod` 7 == 3 ]
[52,59,66,73,80,87,94]

λ> █
```

List Comprehension (Guards aka Filters)

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> [ n | n<-[50..100], n `mod` 7 == 3 ]  
[52,59,66,73,80,87,94]  
  
λ> [ n*n | n<-[1..22], even n ]
```

List Comprehension (Guards aka Filters)

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> [ n | n<-[50..100], n `mod` 7 == 3 ]  
[52,59,66,73,80,87,94]  
  
λ> [ n*n | n<-[1..22], even n ]  
[4,16,36,64,100,144,196,256,324,400,484]  
  
λ> █
```

List Comprehension (Guards aka Filters)

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> [ n | n<-[50..100], n `mod` 7 == 3 ]  
[52,59,66,73,80,87,94]  
  
λ> [ n*n | n<-[1..22], even n ]  
[4,16,36,64,100,144,196,256,324,400,484]  
  
λ> [ n*n | n<-[2,4..22] ]
```


List Comprehension (Guards aka Filters)

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> [ n | n<-[50..100], n `mod` 7 == 3 ]  
[52,59,66,73,80,87,94]  
  
λ> [ n*n | n<-[1..22], even n ]  
[4,16,36,64,100,144,196,256,324,400,484]  
  
λ> [ n*n | n<-[2,4..22] ]  
[4,16,36,64,100,144,196,256,324,400,484]  
  
λ> █
```

List Comprehension (Guards aka Filters)

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> [ n | n<-[50..100], n `mod` 7 == 3 ]  
[52,59,66,73,80,87,94]  
  
λ> [ n*n | n<-[1..22], even n ]  
[4,16,36,64,100,144,196,256,324,400,484]  
  
λ> [ n*n | n<-[2,4..22] ]  
[4,16,36,64,100,144,196,256,324,400,484]  
  
λ> [ (n,n*n) | n<-[ -3..3], n<n*n ]
```

List Comprehension (Guards aka Filters)

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> [ n | n<-[50..100], n `mod` 7 == 3 ]  
[52,59,66,73,80,87,94]  
  
λ> [ n*n | n<-[1..22], even n ]  
[4,16,36,64,100,144,196,256,324,400,484]  
  
λ> [ n*n | n<-[2,4..22] ]  
[4,16,36,64,100,144,196,256,324,400,484]  
  
λ> [ (n,n*n) | n<-[ -3..3], n<n*n ]  
[(-3,9),(-2,4),(-1,1),(2,4),(3,9)]  
  
λ> █
```

List Comprehension (Guards aka Filters)

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> [ n | n<-[50..100], n `mod` 7 == 3 ]  
[52,59,66,73,80,87,94]  
  
λ> [ n*n | n<-[1..22], even n ]  
[4,16,36,64,100,144,196,256,324,400,484]  
  
λ> [ n*n | n<-[2,4..22] ]  
[4,16,36,64,100,144,196,256,324,400,484]  
  
λ> [ (n,n*n) | n<-[ -3..3], n<n*n ]  
[(-3,9),(-2,4),(-1,1),(2,4),(3,9)]  
  
λ> [ n | n <- [10..20], n /= 13, n /= 15, n /= 19 ]
```

List Comprehension (Guards aka Filters)

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
  
λ> [ n | n<-[50..100], n `mod` 7 == 3 ]  
[52,59,66,73,80,87,94]  
  
λ> [ n*n | n<-[1..22], even n ]  
[4,16,36,64,100,144,196,256,324,400,484]  
  
λ> [ n*n | n<-[2,4..22] ]  
[4,16,36,64,100,144,196,256,324,400,484]  
  
λ> [ (n,n*n) | n<-[-3..3], n<n*n ]  
[(-3,9),(-2,4),(-1,1),(2,4),(3,9)]  
  
λ> [ n | n <- [10..20], n /= 13, n /= 15, n /= 19 ]  
[10,11,12,14,16,17,18,20]  
  
λ> █
```

List Comprehension (Guards aka Filters)

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help

λ> [ n | n<-[50..100], n `mod` 7 == 3 ]
[52,59,66,73,80,87,94]

λ> [ n*n | n<-[1..22], even n ]
[4,16,36,64,100,144,196,256,324,400,484]

λ> [ n*n | n<-[2,4..22] ]
[4,16,36,64,100,144,196,256,324,400,484]

λ> [ (n,n*n) | n<-[-3..3], n<n*n ]
[(-3,9),(-2,4),(-1,1),(2,4),(3,9)]

λ> [ n | n <- [10..20], n /= 13, n /= 15, n /= 19 ]
[10,11,12,14,16,17,18,20]

λ> [ c | c <- "one fish", c `elem` "aeiou" ]
```

List Comprehension (Guards aka Filters)

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help

λ> [ n*n | n<-[1..22], even n ]
[4,16,36,64,100,144,196,256,324,400,484]

λ> [ n*n | n<-[2,4..22] ]
[4,16,36,64,100,144,196,256,324,400,484]

λ> [ (n,n*n) | n<-[-3..3], n<n*n ]
[(-3,9),(-2,4),(-1,1),(2,4),(3,9)]

λ> [ n | n <- [10..20], n /= 13, n /= 15, n /= 19 ]
[10,11,12,14,16,17,18,20]

λ> [ c | c <- "one fish", c `elem` "aeiou" ]
"oei"

λ> █
```

List Comprehension (Guards aka Filters)

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help

λ> [ n*n | n<-[1..22], even n ]
[4,16,36,64,100,144,196,256,324,400,484]

λ> [ n*n | n<-[2,4..22] ]
[4,16,36,64,100,144,196,256,324,400,484]

λ> [ (n,n*n) | n<-[-3..3], n<n*n ]
[(-3,9),(-2,4),(-1,1),(2,4),(3,9)]

λ> [ n | n <- [10..20], n /= 13, n /= 15, n /= 19 ]
[10,11,12,14,16,17,18,20]

λ> [ c | c <- "one fish", c `elem` "aeiou" ]
"oei"

λ> [if x<10 then "BOOM!" else "BANG!" | x<-[7..13], odd x]
```


List Comprehension (Guards aka Filters)

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help

λ> [ n*n | n<-[2,4..22] ]
[4,16,36,64,100,144,196,256,324,400,484]

λ> [ (n,n*n) | n<-[-3..3], n<n*n ]
[(-3,9),(-2,4),(-1,1),(2,4),(3,9)]

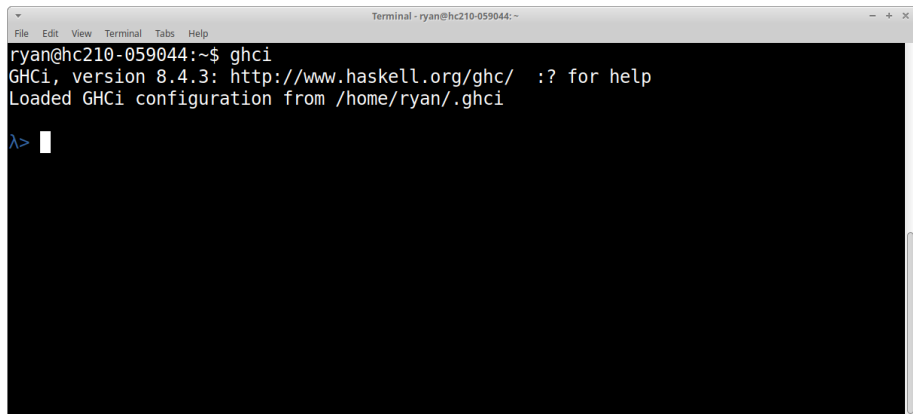
λ> [ n | n <- [10..20], n /= 13, n /= 15, n /= 19 ]
[10,11,12,14,16,17,18,20]

λ> [ c | c <- "one fish", c `elem` "aeiou" ]
"oei"

λ> [if x<10 then "BOOM!" else "BANG!" | x<-[7..13], odd x]
["BOOM!","BOOM!","BANG!","BANG!"]

λ> 
```

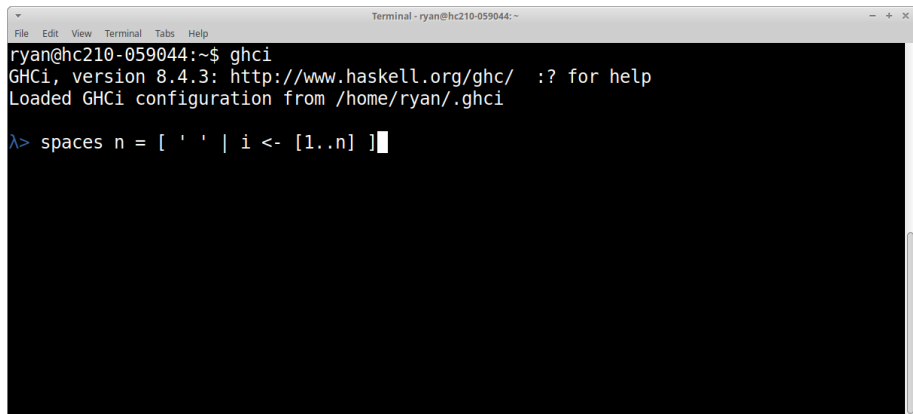
List Comprehension and Functions

A terminal window titled "Terminal - ryan@hc210-059044: ~" with a menu bar containing "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal output shows the command "ghci" being executed, followed by the GHCi version (8.4.3), a URL for help, and the path to the configuration file. The prompt changes from a shell prompt to a lambda prompt.

```
ryan@hc210-059044:~$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> █
```

List Comprehension and Functions



```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> spaces n = [ ' ' | i <- [1..n] ]
```

List Comprehension and Functions

A terminal window titled "Terminal - ryan@hc210-059044: ~" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows the following text:

```
ryan@hc210-059044:~$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> spaces n = [ ' ' | i <- [1..n] ]

λ> space 3
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> spaces n = [ ' ' | i <- [1..n] ]  
  
λ> space 3  
  
<interactive>:2:1: error:  
    • Variable not in scope: space :: Integer -> t  
    • Perhaps you meant 'spaces' (line 1)  
  
λ> spaces 3
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> spaces n = [ ' ' | i <- [1..n] ]  
  
λ> space 3  
  
<interactive>:2:1: error:  
    • Variable not in scope: space :: Integer -> t  
    • Perhaps you meant 'spaces' (line 1)  
  
λ> spaces 3  
"  "  
  
λ> █
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
ryan@hc210-059044:~$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> spaces n = [ ' ' | i <- [1..n] ]

λ> space 3
<interactive>:2:1: error:
  • Variable not in scope: space :: Integer -> t
  • Perhaps you meant 'spaces' (line 1)

λ> spaces 3
"  "

λ> spaces 8
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
Loaded GHCi configuration from /home/ryan/.ghci
λ> spaces n = [ ' ' | i <- [1..n] ]
λ> space 3
<interactive>:2:1: error:
    • Variable not in scope: space :: Integer -> t
    • Perhaps you meant 'spaces' (line 1)
λ> spaces 3
"  "
λ> spaces 8
"      "
λ>
```


List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> spaces n = [ ' ' | i <- [1..n] ]  
  
λ> space 3  
  
<interactive>:2:1: error:  
    • Variable not in scope: space :: Integer -> t  
    • Perhaps you meant 'spaces' (line 1)  
  
λ> spaces 3  
"  "  
  
λ> spaces 8  
"      "  
  
λ> doublePos xs = [2*x | x<-xs, x>0]
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
λ> spaces n = [ ' ' | i <- [1..n] ]
λ> space 3
<interactive>:2:1: error:
  • Variable not in scope: space :: Integer -> t
  • Perhaps you meant 'spaces' (line 1)
λ> spaces 3
"  "
λ> spaces 8
"      "
λ> doublePos xs = [2*x | x<-xs, x>0]
λ>
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
λ> spaces n = [ ' ' | i <- [1..n] ]
λ> space 3
<interactive>:2:1: error:
  • Variable not in scope: space :: Integer -> t
  • Perhaps you meant 'spaces' (line 1)
λ> spaces 3
"  "
λ> spaces 8
"      "
λ> doublePos xs = [2*x | x<-xs, x>0]
λ> doublePos [-1,2,1,2,3]
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
  
<interactive>:2:1: error:  
• Variable not in scope: space :: Integer -> t  
• Perhaps you meant 'spaces' (line 1)  
  
λ> spaces 3  
"  "  
  
λ> spaces 8  
"      "  
  
λ> doublePos xs = [2*x | x<-xs, x>0]  
  
λ> doublePos [-1,2,1,2,3]  
[4,2,4,6]  
  
λ> 
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help

<interactive>:2:1: error:
  • Variable not in scope: space :: Integer -> t
  • Perhaps you meant 'spaces' (line 1)

λ> spaces 3
"  "

λ> spaces 8
"      "

λ> doublePos xs = [2*x | x<-xs, x>0]

λ> doublePos [-1,2,1,2,3]
[4,2,4,6]

λ> doublePos [1,4..13]
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
• Perhaps you meant 'spaces' (line 1)  
  
λ> spaces 3  
"  "  
  
λ> spaces 8  
"  "  
  
λ> doublePos xs = [2*x | x<-xs, x>0]  
  
λ> doublePos [-1,2,1,2,3]  
[4,2,4,6]  
  
λ> doublePos [1,4..13]  
[2,8,14,20,26]  
  
λ> 
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
• Perhaps you meant 'spaces' (line 1)  
  
λ> spaces 3  
" "  
  
λ> spaces 8  
" "  
  
λ> doublePos xs = [2*x | x<-xs, x>0]  
  
λ> doublePos [-1,2,1,2,3]  
[4,2,4,6]  
  
λ> doublePos [1,4..13]  
[2,8,14,20,26]  
  
λ> doublePos [1,0..-78]
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
λ> spaces 8  
"      "  
λ> doublePos xs = [2*x | x<-xs, x>0]  
λ> doublePos [-1,2,1,2,3]  
[4,2,4,6]  
λ> doublePos [1,4..13]  
[2,8,14,20,26]  
λ> doublePos [1,0..-78]  
<interactive>:8:15: error:  
    Variable not in scope: (..) :: Integer -> Integer -> a  
λ> |
```


List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
λ> spaces 8
"      "

λ> doublePos xs = [2*x | x<-xs, x>0]

λ> doublePos [-1,2,1,2,3]
[4,2,4,6]

λ> doublePos [1,4..13]
[2,8,14,20,26]

λ> doublePos [1,0..-78]

<interactive>:8:15: error:
  Variable not in scope: (..-) :: Integer -> Integer -> a

λ> doublePos [1,0.. (-78)]
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
λ> doublePos xs = [2*x | x<-xs, x>0]

λ> doublePos [-1,2,1,2,3]
[4,2,4,6]

λ> doublePos [1,4..13]
[2,8,14,20,26]

λ> doublePos [1,0..-78]

<interactive>:8:15: error:
  Variable not in scope: (..) :: Integer -> Integer -> a

λ> doublePos [1,0.. (-78)]
[2]

λ> 
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: -
File Edit View Terminal Tabs Help
λ> doublePos xs = [2*x | x<-xs, x>0]

λ> doublePos [-1,2,1,2,3]
[4,2,4,6]

λ> doublePos [1,4..13]
[2,8,14,20,26]

λ> doublePos [1,0..-78]

<interactive>:8:15: error:
  Variable not in scope: (..) :: Integer -> Integer -> a

λ> doublePos [1,0.. (-78)]
[2]

λ> factors n = [x | x<-[1..n], n `mod` x == 0]
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
λ> doublePos [-1,2,1,2,3]
[4,2,4,6]

λ> doublePos [1,4..13]
[2,8,14,20,26]

λ> doublePos [1,0..-78]
<interactive>:8:15: error:
  Variable not in scope: (..) :: Integer -> Integer -> a

λ> doublePos [1,0.. (-78)]
[2]

λ> factors n = [x | x<-[1..n], n `mod` x == 0]

λ> 
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
λ> doublePos [-1,2,1,2,3]
[4,2,4,6]

λ> doublePos [1,4..13]
[2,8,14,20,26]

λ> doublePos [1,0..-78]
<interactive>:8:15: error:
  Variable not in scope: (..) :: Integer -> Integer -> a

λ> doublePos [1,0.. (-78)]
[2]

λ> factors n = [x | x<-[1..n], n `mod` x == 0]

λ> factors 12
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
λ> doublePos [1,4..13]
[2,8,14,20,26]

λ> doublePos [1,0..-78]

<interactive>:8:15: error:
  Variable not in scope: (..) :: Integer -> Integer -> a

λ> doublePos [1,0.. (-78)]
[2]

λ> factors n = [x | x<-[1..n], n `mod` x == 0]

λ> factors 12
[1,2,3,4,6,12]

λ> 
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
λ> doublePos [1,0..-78]  
  
<interactive>:8:15: error:  
    Variable not in scope: (..) :: Integer -> Integer -> a  
  
λ> doublePos [1,0.. (-78)]  
[2]  
  
λ> factors n = [x | x<-[1..n], n `mod` x == 0]  
  
λ> factors 12  
[1,2,3,4,6,12]  
  
λ> factors 6  
[1,2,3,6]  
  
λ> 
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
λ> doublePos [1,0..-78]  
  
<interactive>:8:15: error:  
  Variable not in scope: (..) :: Integer -> Integer -> a  
  
λ> doublePos [1,0.. (-78)]  
[2]  
  
λ> factors n = [x | x<-[1..n], n `mod` x == 0]  
  
λ> factors 12  
[1,2,3,4,6,12]  
  
λ> factors 6  
[1,2,3,6]  
  
λ> factors 17
```


List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
Variable not in scope: (..) :: Integer -> Integer -> a  
  
λ> doublePos [1,0.. (-78)]  
[2]  
  
λ> factors n = [x | x<-[1..n], n `mod` x == 0]  
  
λ> factors 12  
[1,2,3,4,6,12]  
  
λ> factors 6  
[1,2,3,6]  
  
λ> factors 17  
[1,17]  
  
λ> 
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
Variable not in scope: (..) :: Integer -> Integer -> a  
  
λ> doublePos [1,0.. (-78)]  
[2]  
  
λ> factors n = [x | x<-[1..n], n `mod` x == 0]  
  
λ> factors 12  
[1,2,3,4,6,12]  
  
λ> factors 6  
[1,2,3,6]  
  
λ> factors 17  
[1,17]  
  
λ> factors 0
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
[2]  
λ> factors n = [x | x<-[1..n], n `mod` x == 0]  
λ> factors 12  
[1,2,3,4,6,12]  
λ> factors 6  
[1,2,3,6]  
λ> factors 17  
[1,17]  
λ> factors 0  
[]  
λ>
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
[2]  
λ> factors n = [x | x<-[1..n], n `mod` x == 0]  
λ> factors 12  
[1,2,3,4,6,12]  
λ> factors 6  
[1,2,3,6]  
λ> factors 17  
[1,17]  
λ> factors 0  
[]  
λ> prime n = factors n == [1,n]
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
λ> factors n = [x | x<-[1..n], n `mod` x == 0]  
  
λ> factors 12  
[1,2,3,4,6,12]  
  
λ> factors 6  
[1,2,3,6]  
  
λ> factors 17  
[1,17]  
  
λ> factors 0  
[]  
  
λ> prime n = factors n == [1,n]  
  
λ> 
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
λ> factors n = [x | x<-[1..n], n `mod` x == 0]  
  
λ> factors 12  
[1,2,3,4,6,12]  
  
λ> factors 6  
[1,2,3,6]  
  
λ> factors 17  
[1,17]  
  
λ> factors 0  
[]  
  
λ> prime n = factors n == [1,n]  
  
λ> prime 12
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
[1,2,3,4,6,12]  
  
λ> factors 6  
[1,2,3,6]  
  
λ> factors 17  
[1,17]  
  
λ> factors 0  
[]  
  
λ> prime n = factors n == [1,n]  
  
λ> prime 12  
False  
  
λ> 
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
[1,2,3,4,6,12]  
  
λ> factors 6  
[1,2,3,6]  
  
λ> factors 17  
[1,17]  
  
λ> factors 0  
[]  
  
λ> prime n = factors n == [1,n]  
  
λ> prime 12  
False  
  
λ> prime 53
```


List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
[1,2,3,6]  
  
λ> factors 17  
[1,17]  
  
λ> factors 0  
[]  
  
λ> prime n = factors n == [1,n]  
  
λ> prime 12  
False  
  
λ> prime 53  
True  
  
λ> 
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
[1,2,3,6]  
  
λ> factors 17  
[1,17]  
  
λ> factors 0  
[]  
  
λ> prime n = factors n == [1,n]  
  
λ> prime 12  
False  
  
λ> prime 53  
True  
  
λ> primes n = [ i | i<-[2..n], prime i]
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
λ> factors 17
[1,17]
λ> factors 0
[]
λ> prime n = factors n == [1,n]
λ> prime 12
False
λ> prime 53
True
λ> primes n = [ i | i<-[2..n], prime i ]
λ> 
```

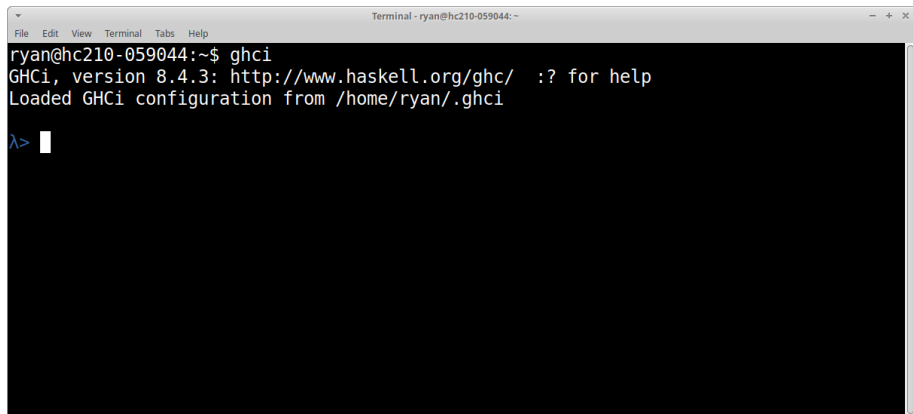
List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
λ> factors 17  
[1,17]  
  
λ> factors 0  
[]  
  
λ> prime n = factors n == [1,n]  
  
λ> prime 12  
False  
  
λ> prime 53  
True  
  
λ> primes n = [ i | i<-[2..n], prime i]  
  
λ> primes 101
```

List Comprehension and Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
λ> factors 0  
[]  
  
λ> prime n = factors n == [1,n]  
  
λ> prime 12  
False  
  
λ> prime 53  
True  
  
λ> primes n = [ i | i<-[2..n], prime i]  
  
λ> primes 101  
[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]  
  
λ> 
```

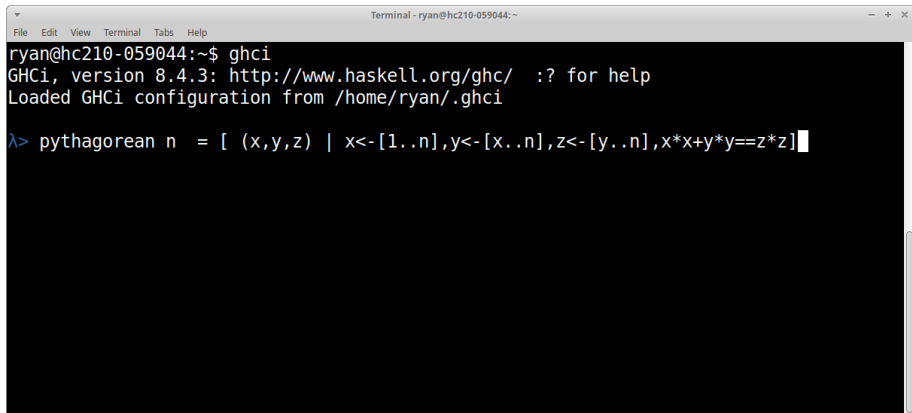
List Comprehension, Pythagorean Triples

A terminal window titled "Terminal - ryan@hc210-059044: ~" with a menu bar containing "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal output shows the command "ghci" being executed, followed by the GHCi version (8.4.3), a URL for help, and the path to the configuration file. The prompt changes from a shell prompt to a Haskell lambda prompt.

```
ryan@hc210-059044:~$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> █
```

List Comprehension, Pythagorean Triples

A terminal window titled "Terminal - ryan@hc210-059044: ~" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows the following text:

```
ryan@hc210-059044:~$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> pythagorean n = [ (x,y,z) | x<-[1..n],y<-[x..n],z<-[y..n],x*x+y*y==z*z]
```

List Comprehension, Pythagorean Triples

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> pythagorean n = [ (x,y,z) | x<-[1..n],y<-[x..n],z<-[y..n],x*x+y*y==z*z]  
λ> █
```


List Comprehension, Pythagorean Triples

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> pythagorean n = [ (x,y,z) | x<-[1..n],y<-[x..n],z<-[y..n],x*x+y*y==z*z]  
  
λ> pythagorean 12
```

List Comprehension, Pythagorean Triples

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> pythagorean n = [ (x,y,z) | x<-[1..n],y<-[x..n],z<-[y..n],x*x+y*y==z*z]  
  
λ> pythagorean 12  
[(3,4,5),(6,8,10)]  
  
λ> █
```

List Comprehension, Pythagorean Triples

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
ryan@hc210-059044:~$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> pythagorean n = [ (x,y,z) | x<-[1..n],y<-[x..n],z<-[y..n],x*x+y*y==z*z]  
  
λ> pythagorean 12  
[(3,4,5),(6,8,10)]  
  
λ> pythagorean 200
```

List Comprehension, Pythagorean Triples

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
, (24, 143, 145), (25, 60, 65), (26, 168, 170), (27, 36, 45), (27, 120, 123), (28, 45, 53), (28, 96, 100),  
(28, 195, 197), (30, 40, 50), (30, 72, 78), (32, 60, 68), (32, 126, 130), (33, 44, 55), (33, 56, 65), (33,  
180, 183), (35, 84, 91), (35, 120, 125), (36, 48, 60), (36, 77, 85), (36, 105, 111), (36, 160, 164), (39,  
52, 65), (39, 80, 89), (40, 42, 58), (40, 75, 85), (40, 96, 104), (42, 56, 70), (42, 144, 150), (44, 117, 1  
25), (45, 60, 75), (45, 108, 117), (48, 55, 73), (48, 64, 80), (48, 90, 102), (48, 140, 148), (48, 189, 19  
5), (49, 168, 175), (50, 120, 130), (51, 68, 85), (51, 140, 149), (52, 165, 173), (54, 72, 90), (55, 132,  
143), (56, 90, 106), (56, 105, 119), (56, 192, 200), (57, 76, 95), (57, 176, 185), (60, 63, 87), (60, 80,  
100), (60, 91, 109), (60, 144, 156), (60, 175, 185), (63, 84, 105), (64, 120, 136), (65, 72, 97), (65, 15  
6, 169), (66, 88, 110), (66, 112, 130), (69, 92, 115), (70, 168, 182), (72, 96, 120), (72, 135, 153), (72  
, 154, 170), (75, 100, 125), (75, 180, 195), (78, 104, 130), (78, 160, 178), (80, 84, 116), (80, 150, 170  
, (81, 108, 135), (84, 112, 140), (84, 135, 159), (85, 132, 157), (87, 116, 145), (88, 105, 137), (88, 1  
65, 187), (90, 120, 150), (93, 124, 155), (95, 168, 193), (96, 110, 146), (96, 128, 160), (99, 132, 165)  
, (99, 168, 195), (100, 105, 145), (102, 136, 170), (104, 153, 185), (105, 140, 175), (108, 144, 180), (  
111, 148, 185), (114, 152, 190), (117, 156, 195), (119, 120, 169), (120, 126, 174), (120, 160, 200), (1  
30, 144, 194)]  
  
λ> pythagorean 100
```

List Comprehension, Pythagorean Triples

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
[ (154,170), (75,100,125), (75,180,195), (78,104,130), (78,160,178), (80,84,116), (80,150,170),
(81,108,135), (84,112,140), (84,135,159), (85,132,157), (87,116,145), (88,105,137), (88,165,187),
(90,120,150), (93,124,155), (95,168,193), (96,110,146), (96,128,160), (99,132,165),
(99,168,195), (100,105,145), (102,136,170), (104,153,185), (105,140,175), (108,144,180),
(111,148,185), (114,152,190), (117,156,195), (119,120,169), (120,126,174), (120,160,200),
(130,144,194) ]

λ> pythagorean 100
[(3,4,5), (5,12,13), (6,8,10), (7,24,25), (8,15,17), (9,12,15), (9,40,41), (10,24,26), (11,60,61),
(12,16,20), (12,35,37), (13,84,85), (14,48,50), (15,20,25), (15,36,39), (16,30,34), (16,63,65),
(18,24,30), (18,80,82), (20,21,29), (20,48,52), (21,28,35), (21,72,75), (24,32,40),
(24,45,51), (24,70,74), (25,60,65), (27,36,45), (28,45,53), (28,96,100), (30,40,50), (30,72,78),
(32,60,68), (33,44,55), (33,56,65), (35,84,91), (36,48,60), (36,77,85), (39,52,65), (39,80,89),
(40,42,58), (40,75,85), (42,56,70), (45,60,75), (48,55,73), (48,64,80), (51,68,85),
(54,72,90), (57,76,95), (60,63,87), (60,80,100), (65,72,97) ]

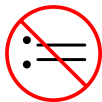
λ> 
```

Before continuing on the last and most important data value: **functions**, we consider the problem of giving data values names.

It is important psychologically to give the programmer a means of referring to data and not just creating data.

- Every object of computation/value gets a name in the same way:

```
theMeaningOfLife = 42
```



NB. Not assignment!

- Control of scope using `let` expression.

```
let <declaration> in <expression>
```

[We don't need any examples now.]

Syntax of Names

Names consist of either letters and digits, or of all symbols. All symbolic names are treated as infix operators by the parser. All non-symbolic names are treated as prefix functions by the parser.

```
:  
!!  
++  
*
```

```
div  
quotRem  
Integer
```


Syntax of Names

Enclosing a name with `()` tells the parser to drop the infix assumption.
Enclosing a name with backquotes tells the parser to assume infix assumption.

	prefix	infix
alphanumeric	<code>elem</code>	<code>'elem'</code>
symbolic	<code>(+)</code>	<code>+</code>

```
div 24 7  
24 'div' 7
```

```
elem 5 [1,2,3,4,5,6]  
5 'elem' [1,2,3,4,5,6]
```

```
(+) 2 3  
2 + 3
```

Declarations

A name is given a value by a declaration of the simple form:

```
name = value
```

In haskell this declaration is actually a specific case of a more general declartion of the form:

```
pattern "=" value
```

Patterns (section 3.17.1 of the reference 2010 manual) are more naturally discussed later in the context of functions.

There are declarations for data types and classes. These will be discussed later.

Declarations

Many languages have declarations for many kinds of things: constants, variables, procedures, and functions.

Haskell does not talk about memory locations (however, see boxed versus unboxed), hence no need for variable declarations.

Fundamentally, given a datum a name in Haskell is the same for any value—functions included.

Overview of Simple Functions

- ① syntax (lambda)
- ② Two Haskell quirks
- ③ anonymous (functions as regular data)
- ④ special syntax
- ⑤ patterns (generalization of formal parameters)
- ⑥ definition by cases
- ⑦ guards [omit]

Syntax of Functions

```
\ x -> 2 * x
```

Syntax of Functions

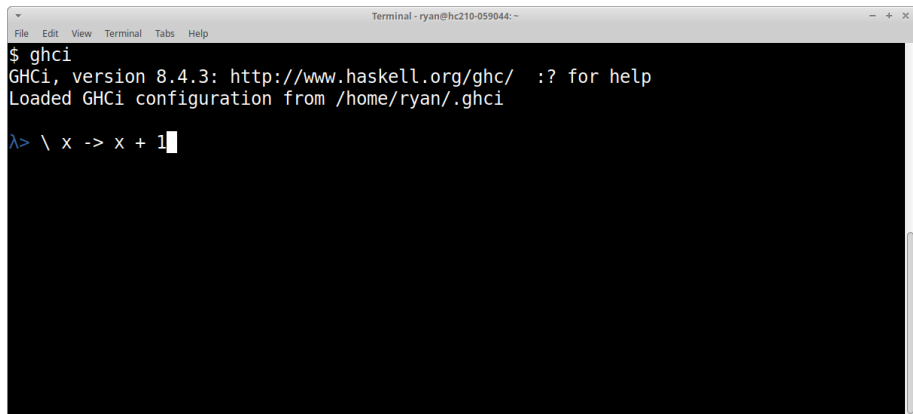
- backslash; pronounced “lambda”
- identifier: name of formal parameter (but generalized to patterns later!)
- arrow separates the body of the function from the parameter

Using GHCi with Functions

Before we can begin illustrating functions we must deal with two quirks in Haskell which totally distract and even obscure the main issues.

- ① Haskell does not print anything reasonable to represent a function by default.
- ② The types of many function contain class constraints, and there is no need to discuss classes in a simple introduction to Haskell. (*Use `:type +d directive.`*)

Using GHCi with Functions



```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> \ x -> x + 1
```


Using GHCi with Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> \ x -> x + 1

<interactive>:1:1: error:
  • No instance for (Show (Integer -> Integer))
    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

λ> █
```

Using GHCi with Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> \ x -> x + 1

<interactive>:1:1: error:
  • No instance for (Show (Integer -> Integer))
    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

λ> :mod + Text.Show.Functions

λ> █
```

Using GHCi with Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> \ x -> x + 1

<interactive>:1:1: error:
  • No instance for (Show (Integer -> Integer))
    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

λ> :mod + Text.Show.Functions

λ> \ x -> x + 1
```

Using GHCi with Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> \ x -> x + 1

<interactive>:1:1: error:
  • No instance for (Show (Integer -> Integer))
    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

λ> :mod + Text.Show.Functions

λ> \ x -> x + 1
<function>

λ> █
```

Using GHCi with Functions

Haskell magical `show` function will show/print anything, *but not functions*.

In Haskell's defense, what is the printable representation of a function? Is the Intel assembly code? The abstract syntax tree of the code? Should it just parrot back out the input source code?

One can get the Haskell interactive system to print the word `<function>` which seems like a really better idea than one of its inscrutable error messages.

Upcoming script

```
:type 'A'  
:type 3  
:type \ b -> not b  
:type \ x -> not x  
:set +t  
\ x -> x + 1
```

Using GHCi with Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> \x -> x + 1

<interactive>:1:1: error:
  • No instance for (Show (Integer -> Integer))
    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

λ> :mod + Text.Show.Functions

λ> :type 'A'
```

Using GHCi with Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> \x -> x + 1

<interactive>:1:1: error:
  • No instance for (Show (Integer -> Integer))
    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

λ> :mod + Text.Show.Functions

λ> :type 'A'
'A' :: Char

λ> 
```


Using GHCi with Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> \x -> x + 1

<interactive>:1:1: error:
  • No instance for (Show (Integer -> Integer))
    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

λ> :mod + Text.Show.Functions

λ> :type 'A'
'A' :: Char

λ> :type 3
```

Using GHCi with Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
λ> \x -> x + 1

<interactive>:1:1: error:
  • No instance for (Show (Integer -> Integer))
    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

λ> :mod + Text.Show.Functions

λ> :type 'A'
'A' :: Char

λ> :type 3
3 :: Num p => p

λ> 
```

Using GHCi with Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
λ> \x -> x + 1

<interactive>:1:1: error:
  • No instance for (Show (Integer -> Integer))
    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

λ> :mod + Text.Show.Functions

λ> :type 'A'
'A' :: Char

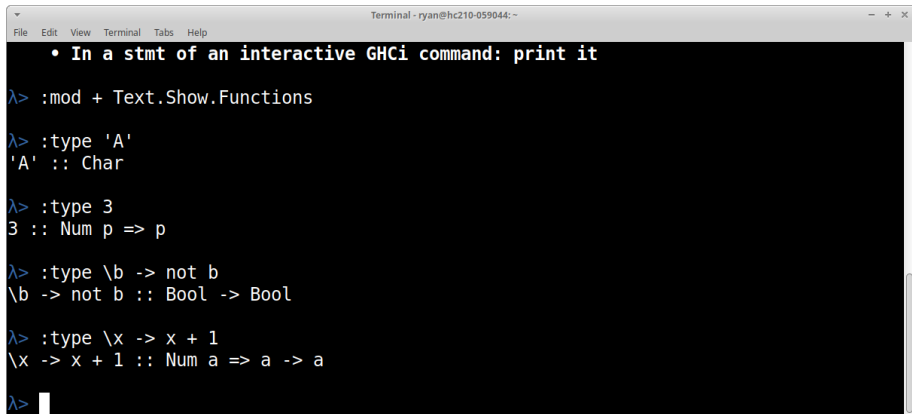
λ> :type 3
3 :: Num p => p

λ> :type \b -> not b
```

Using GHCi with Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
• No instance for (Show (Integer -> Integer))  
  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  
  
λ> :mod + Text.Show.Functions  
  
λ> :type 'A'  
'A' :: Char  
  
λ> :type 3  
3 :: Num p => p  
  
λ> :type \b -> not b  
\b -> not b :: Bool -> Bool  
  
λ> :type \x -> x + 1
```

Using GHCi with Functions



A terminal window titled "Terminal - ryan@hc210-059044: ~" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal content is as follows:

```
• In a stmt of an interactive GHCi command: print it

λ> :mod + Text.Show.Functions

λ> :type 'A'
'A' :: Char

λ> :type 3
3 :: Num p => p

λ> :type \b -> not b
\b -> not b :: Bool -> Bool

λ> :type \x -> x + 1
\x -> x + 1 :: Num a => a -> a

λ>
```

Using GHCi with Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
\ x -> x + 1 :: Num a => a -> a

λ> set + t

<interactive>:7:1: error:
  • Variable not in scope: set
  • Perhaps you meant 'seq' (imported from Prelude)

<interactive>:7:7: error: Variable not in scope: t

λ> :set + t
unknown option: ''
Some flags have not been recognized: t

λ> :set +t

λ> 
```

Using GHCi with Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
<interactive>:7:1: error:  
  • Variable not in scope: set  
  • Perhaps you meant 'seq' (imported from Prelude)  
  
<interactive>:7:7: error: Variable not in scope: t  
  
λ> :set + t  
unknown option: ''  
Some flags have not been recognized: t  
  
λ> :set +t  
  
λ> \x -> x + 1  
<function>  
it :: Num a => a -> a  
  
λ> █
```

Functions Are Data

Upcoming script

```
:mod + Text.Show.Functions Data.Char
```

```
\ x -> x + 1
```

```
\ i -> max i 0
```

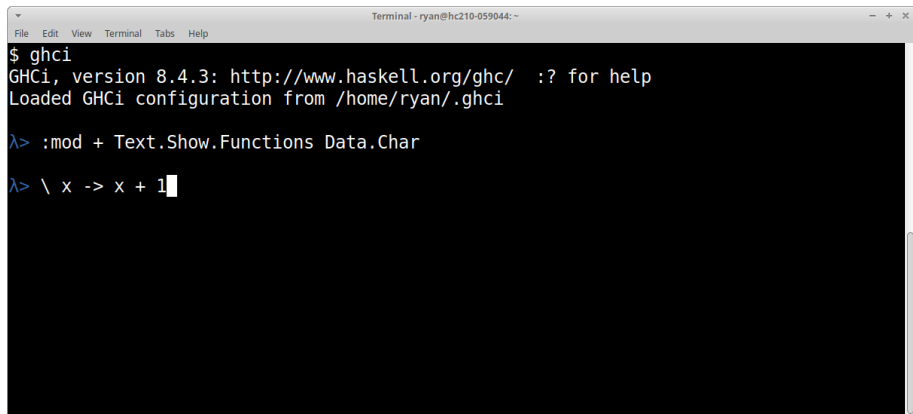
```
\ x -> sin (x+pi)
```

```
\ c -> prd c
```

```
\ s -> s ++ ", " ++ s
```

```
\ xs -> 2:xs
```


Functions Definitions

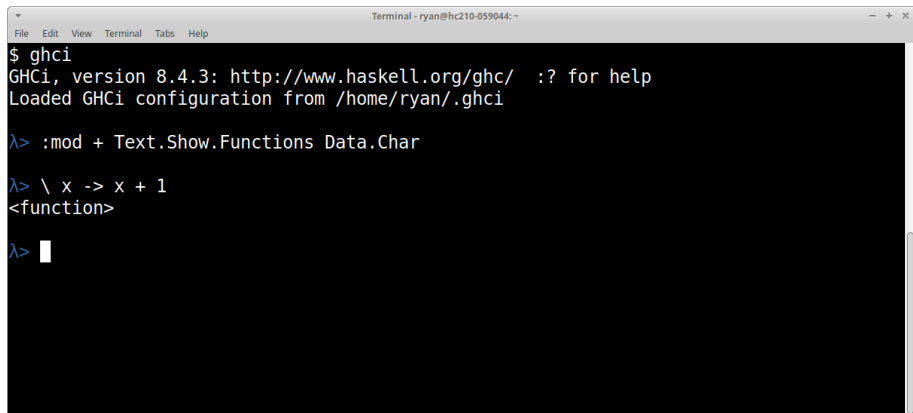
A terminal window titled "Terminal - ryan@hc210-059044: ~" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal content is as follows:

```
$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> :mod + Text.Show.Functions Data.Char

λ> \ x -> x + 1
```

Functions Definitions

A terminal window titled "Terminal - ryan@hc210-059044: ~" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal content shows a GHCi session. The user enters "\$ ghci", which outputs "GHCi, version 8.4.3: http://www.haskell.org/ghc/ :? for help" and "Loaded GHCi configuration from /home/ryan/.ghci". The user then enters "\ λ> :mod + Text.Show.Functions Data.Char", followed by "\ λ> \ x -> x + 1" which outputs "<function>". The prompt "\ λ>" is followed by a white cursor block.

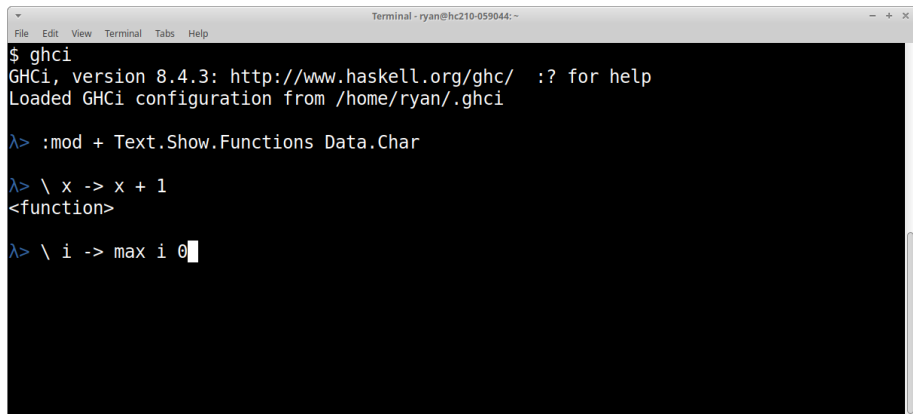
```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/ :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> :mod + Text.Show.Functions Data.Char

λ> \ x -> x + 1
<function>

λ> █
```

Functions Definitions

A terminal window titled "Terminal - ryan@hc210-059044: ~" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows a Haskell GHCi session. The user enters "\$ ghci", which outputs "GHCi, version 8.4.3: http://www.haskell.org/ghc/ :? for help" and "Loaded GHCi configuration from /home/ryan/.ghci". The user then enters "λ> :mod + Text.Show.Functions Data.Char". Next, they enter "λ> \ x -> x + 1", which outputs "<function>". Finally, they enter "λ> \ i -> max i 0" and the cursor is at the end of the line.

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/ :? for help
Loaded GHCi configuration from /home/ryan/.ghci

λ> :mod + Text.Show.Functions Data.Char

λ> \ x -> x + 1
<function>

λ> \ i -> max i 0
```

Functions Definitions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
$ ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/ryan/.ghci

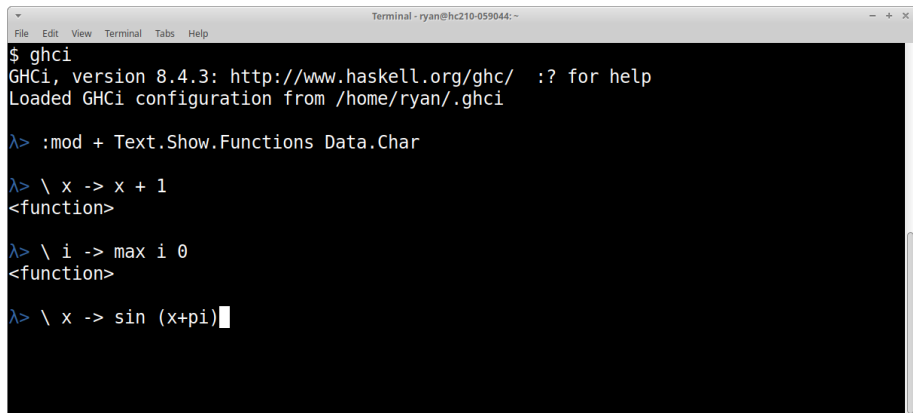
λ> :mod + Text.Show.Functions Data.Char

λ> \ x -> x + 1
<function>

λ> \ i -> max i 0
<function>

λ> █
```

Functions Definitions



```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> :mod + Text.Show.Functions Data.Char  
  
λ> \ x -> x + 1  
<function>  
  
λ> \ i -> max i 0  
<function>  
  
λ> \ x -> sin (x+pi)
```

Functions Definitions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> :mod + Text.Show.Functions Data.Char  
  
λ> \ x -> x + 1  
<function>  
  
λ> \ i -> max i 0  
<function>  
  
λ> \ x -> sin (x+pi)  
<function>  
  
λ> █
```

Functions Definitions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
$ ghci  
GHCi, version 8.4.3: http://www.haskell.org/ghc/  :? for help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> :mod + Text.Show.Functions Data.Char  
  
λ> \ x -> x + 1  
<function>  
  
λ> \ i -> max i 0  
<function>  
  
λ> \ x -> sin (x+pi)  
<function>  
  
λ> \ c -> ord c
```

Functions Definitions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> :mod + Text.Show.Functions Data.Char  
  
λ> \ x -> x + 1  
<function>  
  
λ> \ i -> max i 0  
<function>  
  
λ> \ x -> sin (x+pi)  
<function>  
  
λ> \ c -> ord c  
<function>  
  
λ> |
```


Functions Definitions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
Loaded GHCi configuration from /home/ryan/.ghci  
  
λ> :mod + Text.Show.Functions Data.Char  
  
λ> \ x -> x + 1  
<function>  
  
λ> \ i -> max i 0  
<function>  
  
λ> \ x -> sin (x+pi)  
<function>  
  
λ> \ c -> ord c  
<function>  
  
λ> \ s -> s ++ ", " ++ s
```

Functions Definitions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
  
λ> \ x -> x + 1  
<function>  
  
λ> \ i -> max i 0  
<function>  
  
λ> \ x -> sin (x+pi)  
<function>  
  
λ> \ c -> ord c  
<function>  
  
λ> \ s -> s ++ ", " ++ s  
<function>  
  
λ> |
```

Functions Definitions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
  
λ> \ x -> x + 1  
<function>  
  
λ> \ i -> max i 0  
<function>  
  
λ> \ x -> sin (x+pi)  
<function>  
  
λ> \ c -> ord c  
<function>  
  
λ> \ s -> s ++ ", " ++ s  
<function>  
  
λ> \ xs -> 2:xs
```

Functions Definitions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
  
λ> \ i -> max i 0  
<function>  
  
λ> \ x -> sin (x+pi)  
<function>  
  
λ> \ c -> ord c  
<function>  
  
λ> \ s -> s ++ ", " ++ s  
<function>  
  
λ> \ xs -> 2:xs  
<function>  
  
λ> |
```

What Can You Do With a Function?

- Integer: get the next integer
- Tuple: get the first and the second part
- List: get the rest of the list
- Function: use/apply it to an argument

Functions Application

We ask what is the type of the function, and then we apply it to some element of the domain.

Function Application

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
  
λ> \ i -> max i 0  
<function>  
  
λ> \ x -> sin (x+pi)  
<function>  
  
λ> \ c -> ord c  
<function>  
  
λ> \ s -> s ++ ", " ++ s  
<function>  
  
λ> \ xs -> 2:xs  
<function>  
  
λ> :type +d \ x -> x + 1
```

Function Application

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
  
λ> \ x -> sin (x+pi)  
<function>  
  
λ> \ c -> ord c  
<function>  
  
λ> \ s -> s ++ ", " ++ s  
<function>  
  
λ> \ xs -> 2:xs  
<function>  
  
λ> :type +d \ x -> x + 1  
\ x -> x + 1 :: Integer -> Integer  
  
λ> |
```


Function Application

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
  
λ> \ x -> sin (x+pi)  
<function>  
  
λ> \ c -> ord c  
<function>  
  
λ> \ s -> s ++ ", " ++ s  
<function>  
  
λ> \ xs -> 2:xs  
<function>  
  
λ> :type +d \ x -> x + 1  
\ x -> x + 1 :: Integer -> Integer  
  
λ> (\ x -> x + 1) 234
```

Function Application

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
  
λ> \ c -> ord c  
<function>  
  
λ> \ s -> s ++ ", " ++ s  
<function>  
  
λ> \ xs -> 2:xs  
<function>  
  
λ> :type +d \ x -> x + 1  
\ x -> x + 1 :: Integer -> Integer  
  
λ> (\ x -> x + 1) 234  
235  
  
λ> |
```

Function Application

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
  
λ> \ c -> ord c  
<function>  
  
λ> \ s -> s ++ ", " ++ s  
<function>  
  
λ> \ xs -> 2:xs  
<function>  
  
λ> :type +d \ x -> x + 1  
\ x -> x + 1 :: Integer -> Integer  
  
λ> (\ x -> x + 1) 234  
235  
  
λ> :type +d \ i -> max i 0
```

Function Application

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
  
λ> \ s -> s ++ ", " ++ s  
<function>  
  
λ> \ xs -> 2:xs  
<function>  
  
λ> :type +d \ x -> x + 1  
\ x -> x + 1 :: Integer -> Integer  
  
λ> (\ x -> x + 1) 234  
235  
  
λ> :type +d \ i -> max i 0  
\ i -> max i 0 :: Integer -> Integer  
  
λ> |
```

Function Application

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
  
λ> \ s -> s ++ ", " ++ s  
<function>  
  
λ> \ xs -> 2:xs  
<function>  
  
λ> :type +d \ x -> x + 1  
\ x -> x + 1 :: Integer -> Integer  
  
λ> (\ x -> x + 1) 234  
235  
  
λ> :type +d \ i -> max i 0  
\ i -> max i 0 :: Integer -> Integer  
  
λ> (\ i -> max i 0) -8
```

Function Application

```
Terminal - ryan@hc210-059044: - + x
File Edit View Terminal Tabs Help
\ x -> x + 1 :: Integer -> Integer

λ> (\ x -> x + 1) 234
235

λ> :type +d \ i -> max i 0
\ i -> max i 0 :: Integer -> Integer

λ> (\ i -> max i 0) -8

<interactive>:11:1: error:
  • Non type-variable argument in the constraint: Num (a -> a)
    (Use FlexibleContexts to permit this)
  • When checking the inferred type
    it :: forall a. (Ord a, Num a, Num (a -> a)) => a -> a

λ> |
```

Function Application

```
Terminal - ryan@hc210-059044: - + x
File Edit View Terminal Tabs Help
\ x -> x + 1 :: Integer -> Integer

λ> (\ x -> x + 1) 234
235

λ> :type +d \ i -> max i 0
\ i -> max i 0 :: Integer -> Integer

λ> (\ i -> max i 0) -8

<interactive>:11:1: error:
  • Non type-variable argument in the constraint: Num (a -> a)
    (Use FlexibleContexts to permit this)
  • When checking the inferred type
    it :: forall a. (Ord a, Num a, Num (a -> a)) => a -> a

λ> (\ i -> max i 0) (-8) |
```

Function Application

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
235  
  
λ> :type +d \ i -> max i 0  
\ i -> max i 0 :: Integer -> Integer  
  
λ> (\ i -> max i 0) -8  
  
<interactive>:11:1: error:  
  • Non type-variable argument in the constraint: Num (a -> a)  
    (Use FlexibleContexts to permit this)  
  • When checking the inferred type  
    it :: forall a. (Ord a, Num a, Num (a -> a)) => a -> a  
  
λ> (\ i -> max i 0) (-8)  
0  
  
λ> |
```


Function Application

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
235
λ> :type +d \ i -> max i 0
\ i -> max i 0 :: Integer -> Integer
λ> (\ i -> max i 0) -8
<interactive>:11:1: error:
  • Non type-variable argument in the constraint: Num (a -> a)
    (Use FlexibleContexts to permit this)
  • When checking the inferred type
    it :: forall a. (Ord a, Num a, Num (a -> a)) => a -> a
λ> (\ i -> max i 0) (-8)
0
λ> :type +d \ x -> sin (x+pi)
```

Function Application

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
\ i -> max i 0 :: Integer -> Integer

λ> (\ i -> max i 0) -8

<interactive>:11:1: error:
• Non type-variable argument in the constraint: Num (a -> a)
  (Use FlexibleContexts to permit this)
• When checking the inferred type
  it :: forall a. (Ord a, Num a, Num (a -> a)) => a -> a

λ> (\ i -> max i 0) (-8)
0

λ> :type +d \ x -> sin (x+pi)
\ x -> sin (x+pi) :: Double -> Double

λ> 
```

Function Application

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
\ i -> max i 0 :: Integer -> Integer

λ> (\ i -> max i 0) -8

<interactive>:11:1: error:
  • Non type-variable argument in the constraint: Num (a -> a)
    (Use FlexibleContexts to permit this)
  • When checking the inferred type
    it :: forall a. (Ord a, Num a, Num (a -> a)) => a -> a

λ> (\ i -> max i 0) (-8)
0

λ> :type +d \ x -> sin (x+pi)
\ x -> sin (x+pi) :: Double -> Double

λ> :type +d \ c -> ord c
```

Function Application

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help

<interactive>:11:1: error:
  • Non type-variable argument in the constraint: Num (a -> a)
    (Use FlexibleContexts to permit this)
  • When checking the inferred type
    it :: forall a. (Ord a, Num a, Num (a -> a)) => a -> a

λ> (\ i -> max i 0) (-8)
0

λ> :type +d \ x -> sin (x+pi)
\ x -> sin (x+pi) :: Double -> Double

λ> :type +d \ c -> ord c
\ c -> ord c :: Char -> Int

λ> 
```

Function Application

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help

<interactive>:11:1: error:
• Non type-variable argument in the constraint: Num (a -> a)
  (Use FlexibleContexts to permit this)
• When checking the inferred type
  it :: forall a. (Ord a, Num a, Num (a -> a)) => a -> a

λ> (\ i -> max i 0) (-8)
0

λ> :type +d \ x -> sin (x+pi)
\ x -> sin (x+pi) :: Double -> Double

λ> :type +d \ c -> ord c
\ c -> ord c :: Char -> Int

λ> :type +d \ s -> s ++ " , " ++ s
```

Function Application

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
(Use FlexibleContexts to permit this)  
• When checking the inferred type  
  it :: forall a. (Ord a, Num a, Num (a -> a)) => a -> a  
  
λ> (\ i -> max i 0) (-8)  
0  
  
λ> :type +d \ x -> sin (x+pi)  
\ x -> sin (x+pi) :: Double -> Double  
  
λ> :type +d \ c -> ord c  
\ c -> ord c :: Char -> Int  
  
λ> :type +d \ s -> s ++ ", " ++ s  
\ s -> s ++ ", " ++ s :: [Char] -> [Char]  
  
λ> |
```

Function Application

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
(Use FlexibleContexts to permit this)  
• When checking the inferred type  
  it :: forall a. (Ord a, Num a, Num (a -> a)) => a -> a  
  
λ> (\ i -> max i 0) (-8)  
0  
  
λ> :type +d \ x -> sin (x+pi)  
\ x -> sin (x+pi) :: Double -> Double  
  
λ> :type +d \ c -> ord c  
\ c -> ord c :: Char -> Int  
  
λ> :type +d \ s -> s ++ ", " ++ s  
\ s -> s ++ ", " ++ s :: [Char] -> [Char]  
  
λ> (\ s -> s ++ ", " ++ s) "very"
```

Function Application

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help

λ> (\ i -> max i 0) (-8)
0

λ> :type +d \ x -> sin (x+pi)
\ x -> sin (x+pi) :: Double -> Double

λ> :type +d \ c -> ord c
\ c -> ord c :: Char -> Int

λ> :type +d \ s -> s ++ ", " ++ s
\ s -> s ++ ", " ++ s :: [Char] -> [Char]

λ> (\ s -> s ++ ", " ++ s) "very"
"very, very"

λ> 
```


How do functions get names?

The same way anything gets a name!

```
add1 = \ x -> x + 1
```

```
cutOff = \ i -> max i 0
```

```
g = \ x -> sin (x+pi)
```

```
f = \ c -> ord c
```

```
double = \ s -> s ++ ", " ++ s
```

```
cons2 = \ xs -> 2:xs
```

Naming Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
  
λ> (\ i -> max i 0) (-8)  
0  
  
λ> :type +d \ x -> sin (x+pi)  
\ x -> sin (x+pi) :: Double -> Double  
  
λ> :type +d \ c -> ord c  
\ c -> ord c :: Char -> Int  
  
λ> :type +d \ s -> s ++ ", " ++ s  
\ s -> s ++ ", " ++ s :: [Char] -> [Char]  
  
λ> (\ s -> s ++ ", " ++ s) "very"  
"very, very"  
  
λ> add1 = \ x -> x + 1
```

Naming Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
0  
  
λ> :type +d \ x -> sin (x+pi)  
\ x -> sin (x+pi) :: Double -> Double  
  
λ> :type +d \ c -> ord c  
\ c -> ord c :: Char -> Int  
  
λ> :type +d \ s -> s ++ ", " ++ s  
\ s -> s ++ ", " ++ s :: [Char] -> [Char]  
  
λ> (\ s -> s ++ ", " ++ s) "very"  
"very, very"  
  
λ> add1 = \ x -> x + 1  
  
λ> 
```

Naming Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
\ x -> sin (x+pi) :: Double -> Double

λ> :type +d \ c -> ord c
\ c -> ord c :: Char -> Int

λ> :type +d \ s -> s ++ ", " ++ s
\ s -> s ++ ", " ++ s :: [Char] -> [Char]

λ> (\ s -> s ++ ", " ++ s) "very"
"very, very"

λ> add1 = \ x -> x + 1

λ> :type +d add1
add1 :: Integer -> Integer

λ> 
```

Naming Functions

```
Terminal - ryan@hc210-059044: ~
File Edit View Terminal Tabs Help
\ x -> sin (x+pi) :: Double -> Double

λ> :type +d \ c -> ord c
\ c -> ord c :: Char -> Int

λ> :type +d \ s -> s ++ ", " ++ s
\ s -> s ++ ", " ++ s :: [Char] -> [Char]

λ> (\ s -> s ++ ", " ++ s) "very"
"very, very"

λ> add1 = \ x -> x + 1

λ> :type +d add1
add1 :: Integer -> Integer

λ> add1 813
```

Naming Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
\ c -> ord c :: Char -> Int  
  
λ> :type +d \ s -> s ++ ", " ++ s  
\ s -> s ++ ", " ++ s :: [Char] -> [Char]  
  
λ> (\ s -> s ++ ", " ++ s) "very"  
"very, very"  
  
λ> add1 = \ x -> x + 1  
  
λ> :type +d add1  
add1 :: Integer -> Integer  
  
λ> add1 813  
814  
  
λ> 
```

Naming Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
λ> (\ s -> s ++ ", " ++ s) "very"  
"very, very"  
  
λ> add1 = \ x -> x + 1  
  
λ> :type +d add1  
add1 :: Integer -> Integer  
  
λ> add1 813  
814  
  
λ> cutOff = \ i -> max i 0  
  
λ> :type +d cutOff  
cutOff :: Integer -> Integer  
  
λ> |
```

Naming Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
λ> add1 = \ x -> x + 1  
  
λ> :type +d add1  
add1 :: Integer -> Integer  
  
λ> add1 813  
814  
  
λ> cutOff = \ i -> max i 0  
  
λ> :type +d cutOff  
cutOff :: Integer -> Integer  
  
λ> cutOff (-10)  
0  
  
λ> 
```


Naming Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
  
λ> cutOff (-10)  
0  
  
λ> double = \ x -> x ++ ", " ++ s  
<interactive>:23:30: error: Variable not in scope: s :: [Char]  
  
λ> double = \ s -> s ++ ", " ++ s  
  
λ> :type +d double  
double :: [Char] -> [Char]  
  
λ> double "much"  
"much, much"  
  
λ> █
```

Naming Functions

```
Terminal - ryan@hc210-059044: ~  
File Edit View Terminal Tabs Help  
λ> double = \ s -> s ++ ", " ++ s  
  
λ> :type +d double  
double :: [Char] -> [Char]  
  
λ> double "much"  
"much, much"  
  
λ> cons2 = \ xs -> 2:xs  
  
λ> :type +d cons2  
cons2 :: [Integer] -> [Integer]  
  
λ> cons2 [3,4,5]  
[2,3,4,5]  
  
λ> 
```

Function Declaration Has Special Syntax

```
add1 = \ x -> x + 1
```

```
add1 x = x + 1
```

```
cutOff = \ i -> max i 0
```

```
cutOff i = max i 0
```

```
g = \ x -> sin (x+pi)
```

```
g x = sin (x+pi)
```

```
f = \ c -> ord c
```

```
f c = ord c
```

```
double = \ s -> s ++ " , " ++ s
```

```
double s = s ++ " , " ++ s
```

```
cons2 = \ xs -> 2 : xs
```

```
cons2 xs = 2 : xs
```

Functions

```
f x = if null x then "Empty!" else "Not Empty!"  
factorial n = if n<2 then 1 else n * factorial (n-1)
```

Patterns

Think of a formal argument as a name that matches any value in the domain of the function.

Patterns as formal arguments use constructors to match against the value given to the function as the actual argument.

If the value does not match, you are out of luck.

Patterns

```
p3 (x,y,z) = z  -- definition of function p3
```

```
p3 (1,2,3)  -- evaluates to 3
```

```
tr = (1,2,3)
```

```
p3 tr  -- evaluates to 3
```

```
f (x:2:y:rest) = x+y  -- definition of f
```

```
f (1:2:3:9:[])  -- evaluates to 4
```

```
f [1,2,3,9]    -- evaluates to 4
```

```
l = [12,3,9]
```

```
f l  -- evaluates to 4
```

Wildcard Patterns

```
p3 (_,_,z) = z -- definition of function p3
```

```
f (_:2:_:_ ) = x+y -- definition of f
```

Definition by cases

```
f []           = "empty"  
f (x:[])      = "single"  
f (x:y:[])    = "small"  
f (x:y:z:[])  = "medium"  
f (_)         = "large"
```


Overview

- Everything is digital (all files are binary)
- Unix streams and pipes (function composition is important)
- What is a program?
- GHC
 - ▶ writing a program
 - ▶ compiling a program (also `-Wall`)
 - ▶ running a program (also `+RTS -RTS`)
- `interact`: boilerplate to turn a function into a working program

0010001110110010101100101011010110010000001001100011101000110
0101100101011010110010000001001100011101000110010000101110001
001010110101100100000010011000111010001100100001011000100011
0010011000111010001100100001011100010001110110010101100101011
0011001000010111000100011101100101011001010110101100100000010
0010000101110001000111011001010110010101101011001000000100110
0001000111011001010110010101101011001000000100110001110100011
1011001010110010101101011001000000100110001110100011001000010
0101011001010110101100100000010011000111010001100100001011100
0101011010110010000001001100011101000110010000101110001000111
1001000000100110001110100011001000010111000100011101100101011
0000100110001110100011001000010111000100011101100101011001010
1101000110010000101110001000111011001010110010101101011001000
1100100001011100010001110110010101100101011010110010000001001
1101100101011001010110101100100000010011000111010001100100001
0010101101011001000000100110001110100011001000010111000100011
1011001000000100110001110100011001000010111000100011101100101
0001110100011001000010111000100011101100101011001010110101100
0011001000010111000100011101100101011001010110101100100000010
0001000111011001010110010101101011001000000100110001110100011
0110010101100101011010110010000001001100011101000110010000101
0101011010110010000001001100011101000110010000101110001000111
0010000001001100011101000110010000101110001000111011001010110

Everything is Digital ↗

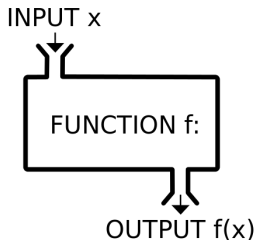
Streams and Pipes



Along the Stream by Sharon France

We know what a function is: it maps an element in the domain to an element in the range.

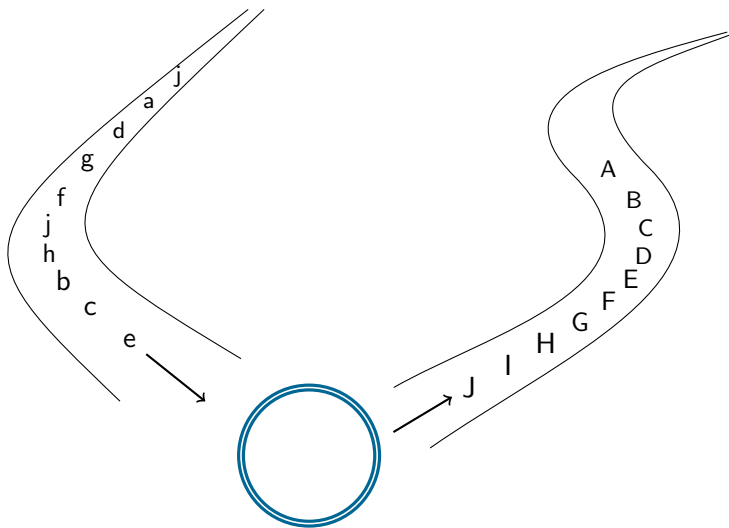
When we think of a program as a function we generally think of something like the factorial function from numbers to numbers.



But this is not a “real” program. A real program reads input and writes output.

Limiting ourselves to the important case of programs from US-ASCII text to US-ASCII text, a real program is really a *function* from a stream/file/string of text to a stream/file/string of text.

Input and Output Stream



Using the Glasgow Haskell Compiler

In Haskell, just like numerous other programming languages, one creates a source program in a text file, one invokes the compiler to create an executable file, and then one commands the computer to run the executable file on some text input and observe or collect the text output.

```
$ ghc -Wall -o main Main.hs  
$ main < input.txt > output.txt
```

Warning!

Ask the compiler to help you write a more beautiful program by turning on all **warnings**.

```
$ ghc -Wall -o main Main.hs
```

Don't turn in a program with warnings.

How do we create a Haskell program that reads the input and writes the output?

Some programming languages have elaborate IO mechanisms which one must learn to simply process the input and produce the output. In Haskell the simplest approach is to use the function `interact` to transform the conceptually pure program (a function from the input stream to the output stream) to an actual, working program on the computer.

standard input stream
(domain)

a predefined Haskell
function

interact

::

(String → String)

→ IO ()

"has type"

standard output stream
(range)

[IO monad!
ignore!]

Examples On-Line

[README](#) ↗