CS4450/7450
Chapter 2: Starting Out

Principles of Programming Languages

Dr. William Harrison

University of Missouri

August 31, 2016

GHCi is basically a fancy calculator

```
$ ghci
GHCi, version 7.10.3: http://www.haskell.org/
    ghc/  :? for help
Prelude> 4 + 2
6
Prelude> not (True && True)
False
Prelude> max 5 4
5
```

Type errors are your friends

```
Prelude> 99 + "Hey"
<interactive>:5:4:
    No instance for (Num [Char]) arising from
         a use of `+'
    In the expression: 99 + "Hey"
    In an equation for `it': it = 99 + "Hey"
Prelude>
```

GHCi Commands

Some Pragmatics

- `:l` or `:load` — load a file or module
- `:t:` or `:type` — give the type of an expression
- `:i` or `:info` — produce information about a definition
- `:q` or `:quit` — quit, derp.

GHCi Commands

Some Pragmatics

- `:l` or `:load` — load a file or module
- `:t:` or `:type` — give the type of an expression
- `:i` or `:info` — produce information about a definition
- `:q` or `:quit` — quit, derp.

```
Prelude> :t not
not :: Bool -> Bool
Prelude> :i not
not :: Bool -> Bool
        -- Defined in ‘GHC.Classes’
Prelude>
```

“Baby’s First Program”

Entered in a file `Chap2.hs`:

```haskell
module Chap2 where

doubleMe x = x + x
```

“Baby’s First Program”, cont’d

```
$ ghci
GHCi, version 7.10.3: http://www.haskell.org/
    ghc/  :? for help
Prelude> :l Chap2.hs
[1 of 1] Compiling Chap2            ( Chap2.
    hs, interpreted )
Ok, modules loaded: Chap2.
*Chap2> doubleMe 9
18
*Chap2> doubleMe 3.14
6.28
*Chap2> :t doubleMe
```

“Baby’s First Program”, cont’d

```
$ ghci
GHCi, version 7.10.3: http://www.haskell.org/
    ghc/  :? for help
Prelude> :l Chap2.hs
[1 of 1] Compiling Chap2            ( Chap2.
    hs, interpreted )
Ok, modules loaded: Chap2.
*Chap2> doubleMe 9
18
*Chap2> doubleMe 3.14
6.28
*Chap2> :t doubleMe
```

```
doubleMe :: Num a => a -> a
*Chap2>
```

Lists, an Introduction to

```
Prelude> let lostNumbers = [4,8,15,16,23,42]
Prelude> lostNumbers
[4,8,15,16,23,42]

Prelude> 99 : lostNumbers
[99,4,8,15,16,23,42]

Prelude> [1,2,3,4] ++ [9,10,11,12]
[1,2,3,4,9,10,11,12]

Prelude> "hello" ++ " " ++ "world"
"hello world"

Prelude> ['w','0'] ++ ['0','t']
"w00t"
```

Some Facts about Lists

- `[]`, `[[]]` and `[[],[],[]]` are all different things. What are their types? Can check that with GHCi.

Some Facts about Lists

- `[]`, `[[]]` and `[[],[],[]]` are all different things. What are their types? Can check that with GHCi.
- Lists are *uniform* in Haskell. E.g., `[1,2,3]` is legal and `[1,2,'c']` is not.

Some Facts about Lists

- `[]`, `[[]]` and `[[],[],[]]` are all different things. What are their types? Can check that with GHCi.
- Lists are *uniform* in Haskell. E.g., `[1,2,3]` is legal and `[1,2,'c']` is not.
- The `data` declaration for lists in Haskell is:

```
data [a] = [] | a : [a]
```

Basic Function on Lists

`head` takes a list and returns its head. The head of a list is its first element (if it exists).

```
ghci> head [5,4,3,2,1]
5
```

- What is the type of `head`?
- How do we write `head` in Haskell?

Basic Function on Lists

`tail` takes a list and returns its tail. In other words, it chops off a list's head.

```
ghci> tail [5,4,3,2,1]
[4,3,2,1]
```

- What is the type of `tail`?
- How do we write `tail` in Haskell?

Basic Function on Lists

If you want to get an element out of a list by index, use `!!`. The indices start at 0.

```
ghci> "Steve Buscemi" !! 6
'B'
ghci> [9.4,33.2,96.2,11.2,23.25] !! 1
33.2
```

- What is the type of `!!`?
- How do we write `!!` in Haskell?