# Guards

CSC345: Programming Languages and Paradigms

WCU
WEST CHESTER UNIVERSITY

# Today

- Writing and Running Haskell Code
- Guards
- “Conversion” Functions
- Tests

- … where we left off…

# Defining Functions

*General Form:*

```
name :: Type
name = expression
```

- Every identifier has a type
- A type is a category of values
- Function types contain arrows

```
addOne :: Int -> Int
addOne n = n + 1
```

- Terminology:
  - Formal parameter
  - Function body
  - Function name
  - Expression
  - Argument

Naming Rules

- `camelCasing`: names for functions and other identifiers begin with a small letter
- `Type` name begin with a capital letter

# Functions w/ multiple parameters

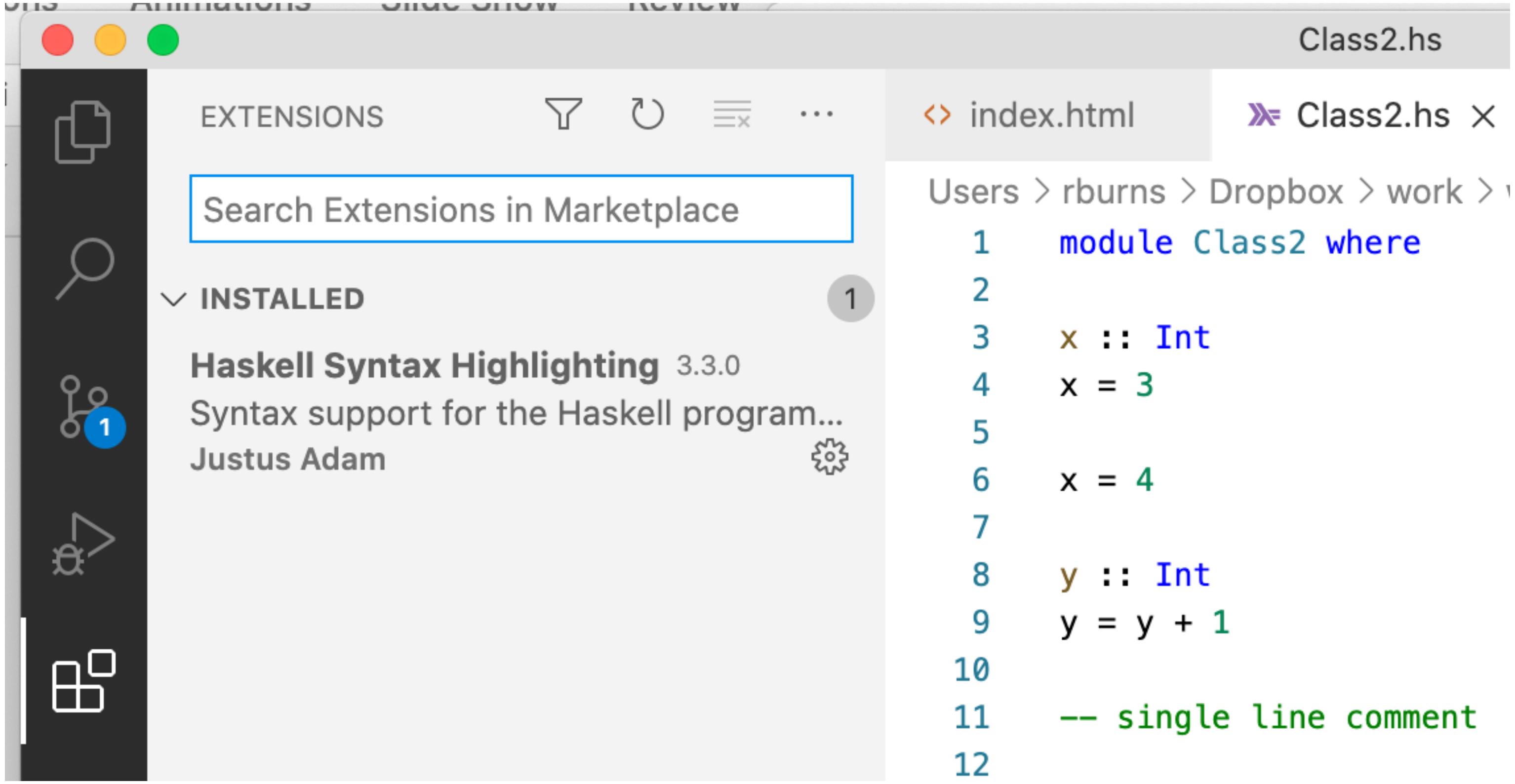
```
name :: t1 -> t2 -> ... -> tk -> t
```

```
name x1 x2 ... xk = e
```

# Example: a function to test whether three `Integers` are equal

# Example: a function to test whether four `Integers` are equal

# Example: a function that models `xor`



Users > rburns > Dropbox > work >

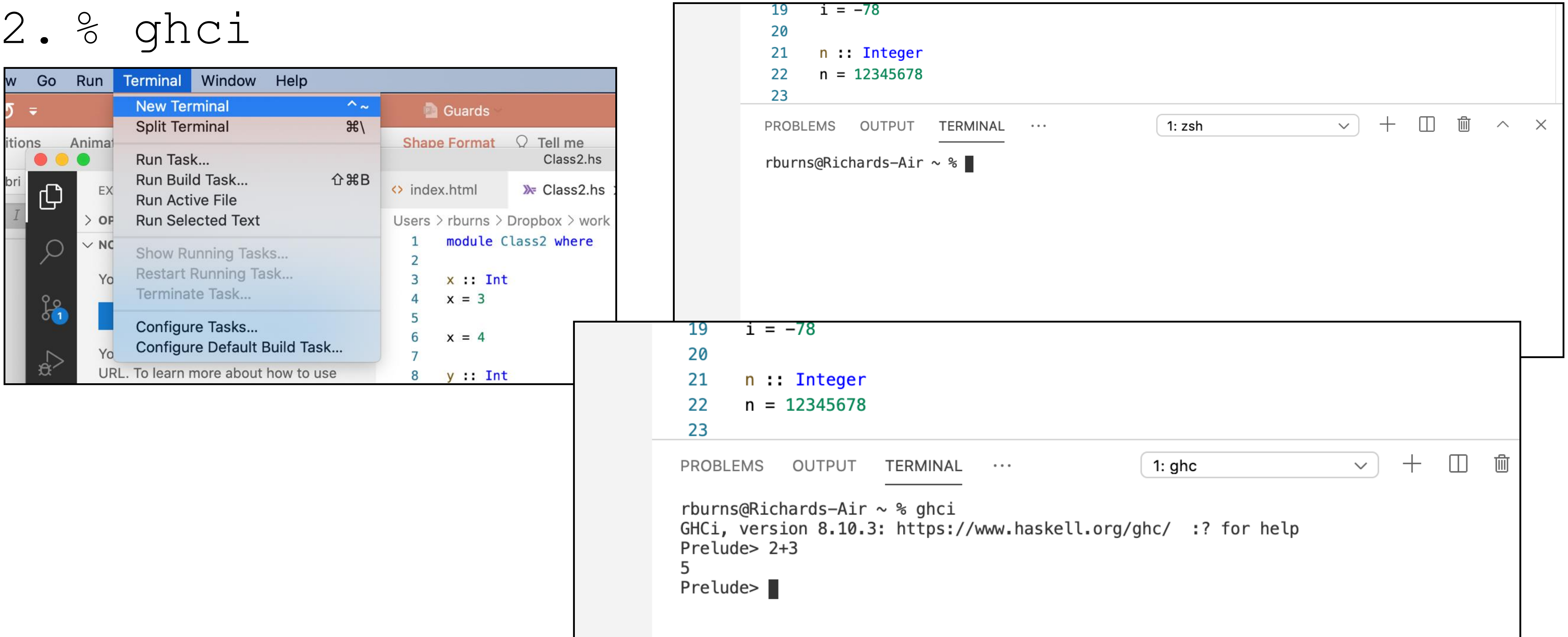
**Haskell Syntax Highlighting** 3.3.0
Syntax support for the Haskell program...
**Justus Adam**

```haskell
module Class2 where

x :: Int
x = 3

x = 4

y :: Int
y = y + 1

-- single line comment

```

# Opening ghci

1. From Visual Studio Code menu: *“Terminal” > “New Terminal”*

2. `% ghci`



```haskell
module Class2 where

x :: Int
x = 3

x = 4

y :: Int
```

```haskell
i = -78

n :: Integer
n = 12345678
```

```
rburns@Richards-Air ~ %
```

```haskell
i = -78

n :: Integer
n = 12345678
```

```
rburns@Richards-Air ~ % ghci
GHCi, version 8.10.3: https://www.haskell.org/ghc/  :? for help
Prelude> 2+3
5
Prelude>
```

# Try out Evaluating Expressions at `ghci` REPL prompt

```
Prelude> 2 + 3
5
```

```
Prelude> :load Class02.hs
Prelude> :reload
Prelude> :quit
```

`m` & `n` must have the same Type

# If expressions

- Currently our function definitions are limited to arithmetic, relational, and logical operators
    - Not very interesting

Conditional Expressions

*General Form:*

```
if condition then m else n
```

*Prompt:* what Type is **`condition`**, **`m`**, **`n`**?

# Example: a function that returns the maximum of two Integer args

# Example: a function that returns the maximum of three Integer args

# The functional paradigm way: Guards

Guards

*General Form:*

```
name x1 x2 ... xk
 | g1       = e1
 | g2       = e2
 ...
 | otherwise = e
```

```
g1, g2, ... :: Bool
e1, e2, ..., e :: t
```

# Example: writing the previous fn’s with guards

# Polymorphic Expressions
(Very ugly at first glance, but later very elegant!)

- No implicit type conversion

```
Int + Integer
```

- Demo

- Integer Division: **`div`**
- Floating-pt Division: **`/`**

```
i :: Int
i = 3
i / i  -- not allowed
3 / 3  -- allowed
```

- 3 is a <u>polymorphic</u> expression (can have multiple types)
- No implicit type conversion is going on

# built-in Type “conversion” functions

```
fromInteger :: Integer -> Int
toInteger :: Int -> Integer
fromInteger :: Integer -> Float
fromIntegral :: Int -> Float
floor :: Float -> Integer
floor :: Double -> Integer
ceiling :: Float -> Integer
round :: Float -> Integer
float2Double :: Float -> Double
double2Float :: Double -> Float
```

# Other Haskell functions to reference

Char / ASCII value converstion:

```
fromEnum :: Char -> Int
toEnum :: Int -> Char

toUpper :: Char -> Char
isDigit :: Char -> Bool
```

See pg. 54 Thompson for example usages.