Fachhochschule Nordwestschweiz

Functional Programming

Modules

Daniel Kröni
University of Applied Sciences Northwestern Switzerland

Learning Targets

You understand the concept of modules.

You can define your own modules.

Content

- **Module Systems**
- **Importing Modules**
  - qualified
  - hiding
- **Creating a Module**
  - flat
  - hierarchical

Modules

- **Haskell programs can be split into modules**
- **A Haskell module is a collection of related**
  - functions
  - types
  - typeclasses and instances
- **Main advantages of modular software**
  - **Better reuse**: More flexible than a single monolithic piece of code
    - More like Lego than like Playmobil
  - Parts can be developed separately from each other
  - Name space control
  - Separate compilation

Module Overview

- **Many modules are delivered with the Haskell Platform**
- **Examples**
  - Prelude : This module is imported by default
  - Data.List : List utilities
  - Data.Char : Character Utilities
  - System.IO : Utilities for general I/O using handles, etc.
  - Network : Utilities for network I/O
- **Use :browse in ghci to list the content of a module**

```
ghci> :browse Data.Char
Data.Char.digitToInt :: Char -> Int
Data.Char.isLetter :: Char -> Bool
Data.Char.isNumber :: Char -> Bool
...
```

https://hackage.haskell.org/package/base-4.13.0.0/docs/Prelude.html

Imports

- **Syntax for importing modules in a Haskell script**

```
import <module name>
```

- **Imports must be done before defining any functions**
- **A script can import several modules**
  - Put each import statement into a separate line
- **Example:**
  - Import the Data.List module in order to use nub

```
import Data.List

numUniques :: (Eq a) => [a] -> Int
numUniques = length . nub -- nub is defined in Data.List
```

Imports in GHCi

- **In GHCi a module can be loaded by**

```
ghci> import Data.List
```

- **Importing multiple modules in GHCi**

```
ghci> :m + Data.List Data.Char Network
```

- **Example session**

```
ghci> nub [1,1,2,3]

<interactive>:6:1: Not in scope: `nub'
ghci> :m + Data.List
ghci Data.List> nub [1,1,2,3]
[1,2,3]
```

Selective Imports

- **One can selectively import the required functions instead of importing all**

```
import Data.List (nub, sort)
```

- **One can hide functions when importing a module**
  - Useful when multiple functions have the same name

```
import Data.List hiding (nub)

nub :: Eq a => [a] -> [a]
nub [] = []
nub (x:xs) = x : nub (filter (/=x) xs)
```

Qualified Imports

- **Modules can be imported `qualified`**

```
import qualified Data.List
```

  - Functions must then be referred to including the module name

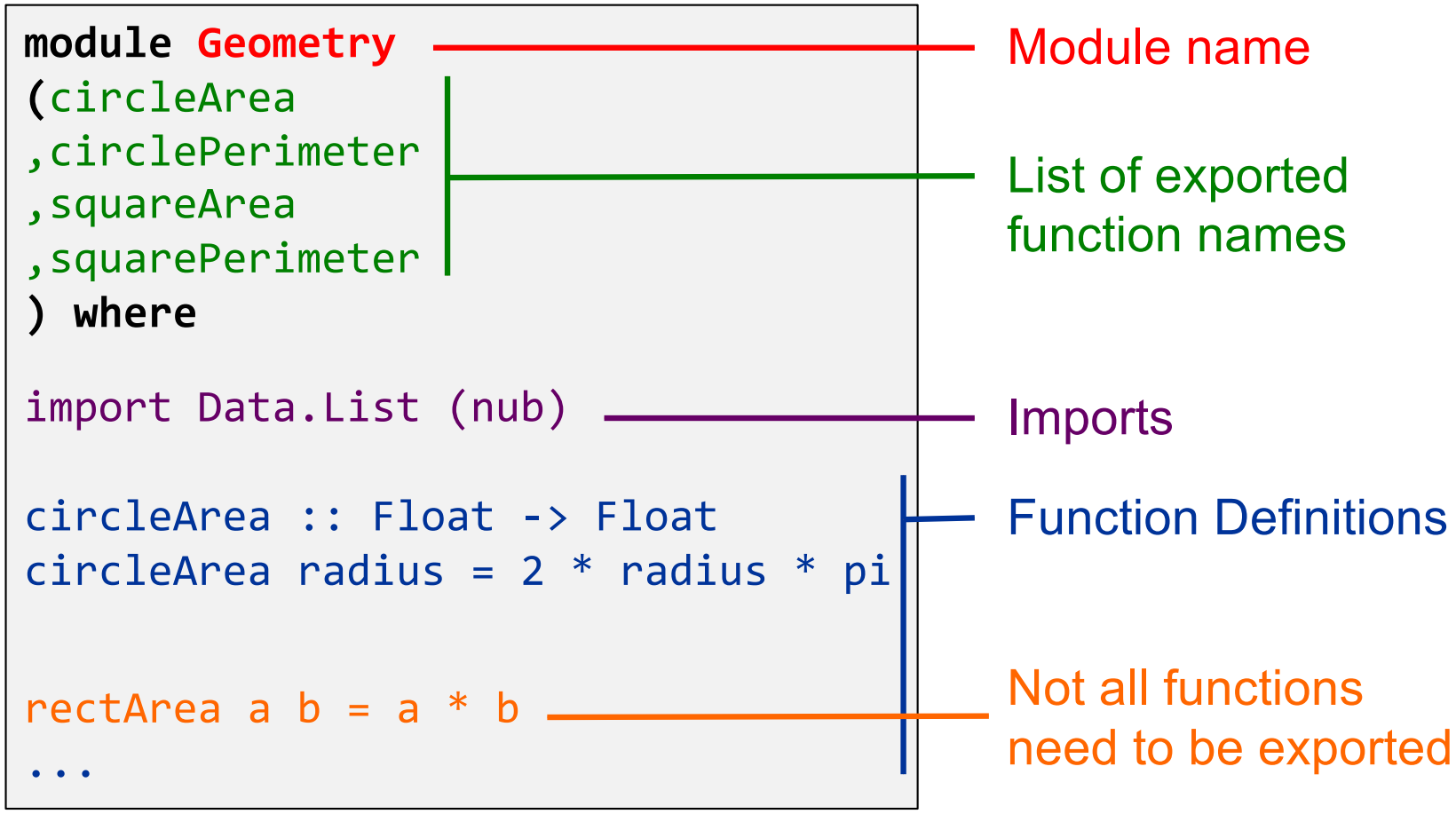
```
elems = Data.List.nub [1,1,2,3]
```

- **Qualified imports can be abbreviated**

```
import qualified Data.List as L

elems = L.nub [1,1,2,3]
```

Defining Modules

- **Syntax for module definition within a file named Geometry.hs**



Defining Hierarchical Modules

- **Modules can be organized in a hierarchical structure**
- **The two files need to be placed inside a folder called Geometry**

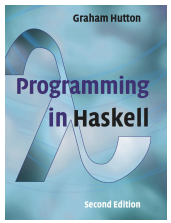
Circle.hs

```
module Geometry.Circle
(area
,perimeter
) where

area :: Float -> Float
area radius = 2 * radius * pi
...
```

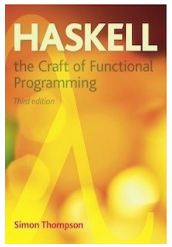
Square.hs

```
module Geometry.Square
(area
,perimeter
) where

perimeter :: Float -> Float
perimeter a = 4 * a
...
```

Further Reading



Not covered



Chapter 15

Book: Chapter 6

Web: http://learnyouahaskell.com/modules