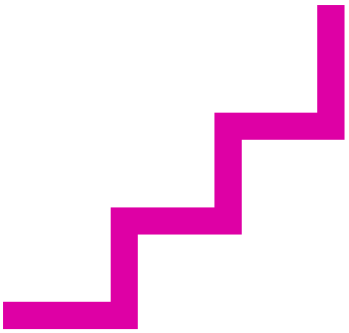

Functional Programming in Haskell

The University of Glasgow

1.1

Welcome to the Course

Video

Haskell Basics: Expressions and Equations

1.2

Basic Elements By Example

Video

1.3

Introduction to Expressions and Equations

Article

1.4

Do it Yourself: Expressions, Functions and Equations

Exercise

1.5

Test Your Understanding

Quiz

1.6

Summary

Article

Haskell Basics: Reduction, Functions and Lists

1.7

More Basic Elements by Example

Video

6.11

1.8

6 more steps to go

and Lists

Here be dragons

Do it Yourself: Functions and Lists

Exercise

1.10

Example: the Maybe monad

Test Your Understanding

1 comment

Quiz

- We've already seen the Maybe type. Let's look at the Maybe monad, which makes using the Maybe type a lot easier.

1.11

The *Maybe* type constructor

Summary

Article

You already know the definition of the *Maybe* type:

```
data Maybe a = Just a | Nothing
```

Finding Out More

Example use of Maybe: Safe *head* and *tail*

1.12

The *head* and *tail* functions from the Prelude are not safe in the sense that they fail when called on an empty list. We can define safe versions using *Maybe*:

Recommended Reading

```
myHead :: [a] -> Maybe a
myHead [] = Nothing
myHead (x:xs) = Just x
```

Video

1.13

```
myTail :: [a] -> Maybe [a]
myTail [] = Nothing
myTail (x:xs) = Just xs
```

Spot the Difference

Discussion

Monad instance of Maybe

1.14

Now we can make *Maybe* an instance of the *Monad* type class, simply by providing the appropriate definitions for *return*, *bind*, *then* and *fail*:

End of Week 1

Video

```
import Control.Monad
```

```
instance Monad Maybe where
   return x       = Just x
   Nothing  >>= f = Nothing
   (Just x) >>= f = f x
   fail _         = Nothing
```

Haskell blocks

There are a few additional functions defined in the *MonadPlus* type class:

```
instance MonadPlus Maybe where
   mzero             = Nothing
   Nothing `mplus` x = x
   x `mplus` _       = x
```

2.1

Welcome to week 2

That's it, we now have a *Maybe* monad!

Video

Note: for users of `ghc 7.10` and higher, we need to do a little bit more work.

2.2

Explicit Maybe versus the Maybe Monad

Bonus Useful Monadic Variants and Expressions

Exercise

Let's see what this monad gives us:

2.3

A computation using explicit Maybe

Zip that List

```
foo :: [a] -> Maybe a
```

```
foo xs =
  case myTail xs of
    Nothing -> Nothing
    Just a -> case myTail a of
                Nothing -> Nothing
                Just b -> myHead b
```

To combine computations that use the *Maybe* type, we need explicit `case` expressions to pattern match against the type.

A computation using the Maybe monad

Let's write this computation using the *Maybe* monad, first using the `>>=` operator:

```
bar :: [a] -> Maybe a
bar xs =
  myTail xs >>=
    (\a -> myTail a >>=
      (\b -> myHead b))
```

Now let's change the line breaks and indentation a bit to make it look nicer:

```
bar2 :: [a] -> Maybe a
bar2 xs =
  myTail xs >>= (\a ->
  myTail a >>=  (\b ->
  myHead b))
```

Thanks to the associativity law, we can also remove unnecessary parentheses:

```
bar3 :: [a] -> Maybe a
bar3 xs =
  myTail xs >>= \a ->
  myTail a >>=  \b ->
  myHead b
```

This is already a lot cleaner, but finally we can use the `do`-notation:

```
bar3 :: [a] -> Maybe a
bar3 xs = do
  a <- myTail xs
  b <- myTail a
  myHead b
```

Clearly, the final monadic code is a lot more readable than the original non-monadic code.

Example: Reduction of bar [5,6]

```
bar [5,6]

-- > substitute [5,6] for xs in definition of bar

myTail [5,6] >>=
  (\a -> myTail a >>=
    (\b -> myHead b))

-- > def. myTail

Just [6] >>=
  (\a -> myTail a >>=
    (\b -> myHead b))

-- >  def.2 of (>>=)
```

2.12

What do you know about Haskell?

Quiz

2.13

End of Week 2

Video

Week 3

Data Structures and Types

Functions on Lists

```
        (\a -> myTail a >>=
         (\b -> myHead b))
          [6]
    -- > beta reduction, substitute [6] for a
        myTail [6] >>= (\b -> myHead b)
    -- > reduce myTail
        Just [4] >>=  (\b -> myHead b)
    -- >  def.2 of (>>=)
        (\b -> myHead b) []
    -- > beta reduction, substitute [] for b
        myHead []
    -- > def.1 myHead
        Nothing
```

3.1

Example: Reduction of bar [5,6,7]

Welcome to Week 3

Video

```
        bar [5,6,7]
    -- > substitute [5,6,7] for xs in definition of bar
```

3.2

Recursive Functions on Lists

Article

3.3

Functional Maps (and) Folds versus Imperative Loops

Video

3.4

Do it Yourself: Lists and Recursion

Exercise

3.5

Do it Yourself: Function Composition

Exercise

3.6

What Have We Learned About Lists?

Quiz

```
        myTail [5,6,7] >>=
         (\a -> myTail a >>=
         (\b -> myHead b))
    -- > def. myTail
        Just [6,7]  >>=
         (\a -> myTail a >>=
         (\b -> myHead b))
    -- > def.2 of (>>=)
        (\a -> myTail a >>=
         (\b -> myHead b))
           [6,7]
    -- > beta reduction, substitute [6,7] for a
        myTail [6,7] >>= (\b -> myHead b)
    -- > reduce myTail
        Just [7] >>=  (\b -> myHead b)
    -- >  def.2 of (>>=)
        (\b -> myHead b) [7]
    -- > beta reduction, substitute [7] for b
        myHead [7]
    -- > def myHead
        Just 7
```


3.7

Share this article:

Mini Sudoku Book Generator

Exercise

-
-
-

Summary

1 comment

Article

Previous

Mark as complete

Next

Mark as complete

Comments

Video

Tomas P.

Add a comment... (plain text and markdown available)

0/1200

Post Learn more about markdown

Show:

All comments

Sort by:

Newest

Filter

- GY

Gwee YC

Follow

05 NOV05 NOV

Why MonadPlus is necessary for the a Monad is not clear? How is MonadPlus applied? Great if we can have some illustrative examples here.

(edited)

Like

Reply

Bookmark

Report

- Help Centre
- Child safety
- Privacy
- T&Cs

Video

Week 4

When Programs Get Bigger

Program Structure

4.1

Welcome to Week 4

Video

4.2

Keep Your Programs Tidy

Article

4.3

Guards, Guards!

Article

4.4

Dealing with Uncertainty

Video

4.5

Idiomatic Haskell

Quiz

Parsing Text

4.6

Parsing Text Using Higher-Order Functions

Article

4.7

Parsing using Parsec: a practical example

Video

4.8

Parser Puzzles

Quiz

4.9

Summary

Article

Am I Right?

4.10

Check my Program is Correct

Video

4.11

Using QuickCheck

Article

4.12

Talk with a Haskell Teacher

Video

Week 5

Hardcore Haskell

Laziness and Infinite Data structures

5.1

Welcome to Week 5

Video

5.2

Lazy is Good

Video

5.3

Infinite Data Structures

Article

5.4

To Infinity (but not beyond)

Quiz

More about Types

5.5

Type Horror Stories

Discussion

5.6

Types, lambda functions and type classes

Article

5.7

Curry is on the menu

Video

5.8

Type Inference by Example

Video

5.9

You are the type checker

Quiz

5.10

Summary

Article

Haskell in the Real World

5.11

Haskell at Facebook

Video

5.12

Haskell in the Wild

Article

5.13

Course Feedback

Article

Week 6

Think like a Functional Programmer

Type Classes

6.1

Welcome to Week 6

Video

6.2

Types with Class

Video

6.3

Type Classes in more Detail

Article

6.4

Summary

Article

Geek Greek

6.5

Introduction to the Lambda calculus

Article

6.6

There are Only Functions! (Optional)

Video

6.7

We Love Lambda!

Quiz

6.8

Summary

Article

The M-word

6.9

We Already Know About Monads

Video

6.10

Introduction to monad theory

Article

6.11

Example: the Maybe monad

Article

6.12

Monad metaphors

Discussion

6.13

Summary

Article

So long and thanks for all the fun(ctions)!

6.14

Functional Programming in Other Languages

Video

6.15

Will You Use Haskell in the Future?

Discussion

6.16

The End of the Affair

Video