Introduction to Haskell & some category theory

Wellington Functional Programming

Finlay Thompson

26 March 2015

DRAGONFLY
Data Science

Introduction

Haskell is strange ?

Haskell is strange ?

A functional programming language

Haskell is strange ?

A functional programming language

With lazy evaluation

Haskell is strange ?

A functional programming language

With lazy evaluation

Pure, with no side effects

Haskell is strange ?

A functional programming language

With lazy evaluation

Pure, with no side effects

Fairly old

Haskell is strange ?

A functional programming language

With lazy evaluation

Pure, with no side effects

Fairly old, fairly odd

Haskell is hard ?

Haskell is hard ?

My program won't compile,
and I don't know why ?

Haskell is hard ?

My program won't compile, and I don't know why ?

The tutorials online are confusing.

Haskell is hard ?

My program won't compile,
and I don't know why ?

The tutorials online are
confusing.

Oh god, I am reading math !

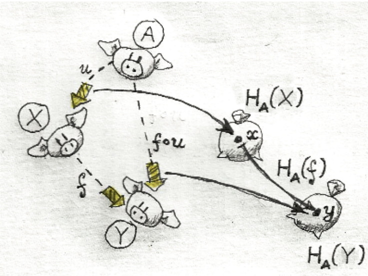
Haskell is hard ?

My program won't compile, and I don't know why ?

The tutorials online are confusing.

Oh god, I am reading math !

Huh ?



Haskell is impractical ?

Haskell is impractical ?

Strong type system gets in the way

Haskell is impractical ?

Strong type system gets in the way

Hard to install, and find good libraries

Haskell is impractical ?

Strong type system gets in the way

Hard to install, and find good libraries

Impossible to find other developers

Haskell is impractical ?

Strong type system gets in the way

Hard to install, and find good libraries

Impossible to find other developers

But, Haskell is awesome!

But, Haskell is awesome!

Is Haskell weird ?

But, Haskell is awesome!

Is Haskell weird ?

- No, just different. Its the other languages that are weird.

But, Haskell is awesome!

Is Haskell weird ?

- No, just different. Its the other languages that are weird.

Is Haskell hard ?

But, Haskell is awesome!

Is Haskell weird ?

- No, just different. Its the other languages that are weird.

Is Haskell hard ?

- No, it makes you think differently, which is good.

But, Haskell is awesome!

Is Haskell weird ?

- No, just different. Its the other languages that are weird.

Is Haskell hard ?

- No, it makes you think differently, which is good.

Is Haskell impractical ?

But, Haskell is awesome!

Is Haskell weird ?

- No, just different. Its the other languages that are weird.

Is Haskell hard ?

- No, it makes you think differently, which is good.

Is Haskell impractical ?

- Hackage has thousands of libraries
- Haskell is fast, and getting faster

To learn Haskell,
it helps to learn a little category theory.

To learn Haskell,
it helps to learn a little category theory.

Actually, I reckon you already know category theory!

Anatomy of a function

```python
def capitalise(name):
    f = name[0].upper()
    r = name[1:].lower()
    return f+r
```

DRAGONFLY
Data Science

this is a function

```
def capitalise(name):
    f = name[0].upper()
    r = name[1:].lower()
    return f+r
```

DRAGONFLY
Data Science

this is a function

```
def capitalise(name):   from text
    f = name[0].upper()
    r = name[1:].lower()
    return f+r
```

this is a function

```python
def capitalise(name):
    f = name[0].upper()
    r = name[1:].lower()
    return f+r
```

from text

to text

this is a function

from text

```python
def capitalise(name):
    f = name[0].upper()
    r = name[1:].lower()
    return f+r
```

noise

to text

```haskell
capitalise :: String -> String
capitalise [] = []
capitalise (a:as)
  = (toUpper a : map toLower as)
```

this is the function

```
capitalise :: String -> String
capitalise [] = []
capitalise (a:as)
  = (toUpper a : map toLower as)
```

DRAGONFLY
Data Science

this is the function

from text

```haskell
capitalise :: String -> String
capitalise [] = []
capitalise (a:as)
  = (toUpper a : map toLower as)
```

this is the function

from text

to text

```
capitalise :: String -> String
capitalise [] = []
capitalise (a:as)
  = (toUpper a : map toLower as)
```

Dragonfly Data Science

this is the function

from text

to text

```haskell
capitalise :: String -> String
capitalise [] = []
capitalise (a:as)
  = (toUpper a : map toLower as)
```

no noise now !

DRAGONFLY
Data Science

A little category theory

DRAGONFLY
Data Science

Category theory is all about arrows.

Category theory is all about arrows.

Need to define what the **objects** are.

Category theory is all about arrows.

Need to define what the **objects** are.

Between two objects, there are **arrows**.

Category theory is all about arrows.

Need to define what the **objects** are.

Between two objects, there are **arrows**.

There are some rules, more on that later.

Programming involves defining arrows.

Programming involves defining arrows.

The objects in the programming category are **types**. Types are sets of values.

Programming involves defining arrows.

The objects in the programming category are **types**. Types are sets of values.

Writing functions in a programming language involves defining arrows between data types.

Programming involves defining arrows.

The objects in the programming category are **types**. Types are sets of values.

Writing functions in a programming language involves defining arrows between data types.

Haskell emphasises category theory aspect of programming.

Warning, philosophical musing ahead…

Warning, philosophical musing ahead…

Category theory is **about** the common patterns that emerge when we consider diagrams of objects and arrows.

Warning, philosophical musing ahead...

Category theory is **about** the common patterns that emerge when we consider diagrams of objects and arrows.

A category defines some kind of objects, and the way we can transform these objects into each other. It is a very general concept, and so almost completely vacuous.

Warning, philosophical musing ahead...

Category theory is **about** the common patterns that emerge when we consider diagrams of objects and arrows.

A category defines some kind of objects, and the way we can transform these objects into each other. It is a very general concept, and so almost completely vacuous.

As is often the case with mathematical concepts, there is nothing more than the definitions.

Definition

A category C consists of

- a class of objects $Obj(C)$,
- $\forall X, Y \in Obj(C)$, $\exists$ a class of arrows $C(X, Y)$.
- $\forall f : X \to Y$, and $g : Y \to Z$, $\exists g \circ f : X \to Z$.

$$\begin{tikzcd} X \arrow[r, "f"] \arrow[dr, "g \circ f"'] & Y \arrow[d, "g"] \\ & Z \end{tikzcd}$$

Such that

- $\forall X \in Obj(C)$, $\exists id_X : X \to X$,
- $\forall f : X \to Y$, then

$$f \circ id_X = f = id_Y \circ f$$

- $\forall f : X \to Y$, $g : Y \to Z$, and $h : Z \to W$, then

$$(h \circ g) \circ f = h \circ (g \circ f)$$

Programming in patterns

The history of computer programming has involved the discovering patterns.

The history of computer programming has involved the discovering patterns.

Subroutines emerged as a pattern from assembly and fortran, and became the central concept in C.

The history of computer programming has involved the discovering patterns.

Subroutines emerged as a pattern from assembly and fortran, and became the central concept in C.

Associating functions with structs emerged as a pattern in C, and became object classes in C++.

The history of computer programming has involved the discovering patterns.

Subroutines emerged as a pattern from assembly and fortran, and became the central concept in C.

Associating functions with structs emerged as a pattern in C, and became object classes in C++.

Numerous patterns have emerged from mainstream object-oriented languages such as Java, C#, and C++.

Category theory provides Haskell with patterns.

Category theory provides Haskell with patterns.

The Haskell language has it's origins in the lambda calculus, and is strongly influenced by ideas category theory.

Category theory provides Haskell with patterns.

The Haskell language has it's origins in the lambda calculus, and is strongly influenced by ideas category theory.

The Haskell community draws on category theory to provide programming patterns.

Category theory provides Haskell with patterns.

The Haskell language has it's origins in the lambda calculus, and is strongly influenced by ideas category theory.

The Haskell community draws on category theory to provide programming patterns.

They *tend* to be good patterns, with good performance characteristics, and general enough to be useful.

Pattern: Functors

Functors are arrows between categories.

Functors are arrows between categories.

Definition

A Functor $F : C \to D$ consists of

- A map $F : Obj(C) \to Obj(D)$,
- $\forall X, Y \in Obj(C)$, a map $F : C(X, Y) \to D(FX, FY)$

Such that

- $\forall X \in C$, then $F(id_X) = id_{FX}$
- $\forall f : X \to Y, g : Y \to Z$, then $F(g \circ f) = F(g) \circ F(f)$

DRAGONFLY
Data Science

User defined data types in Haskell

User defined data types in Haskell

The **data** keyword creates new data types in Haskell.

User defined data types in Haskell

The **data** keyword creates new data types in Haskell.

They can take other types as parameters, creating mappings from Haskell types to Haskell types, the beginning of a Functor.

User defined data types in Haskell

The **data** keyword creates new data types in Haskell.

They can take other types as parameters, creating mappings from Haskell types to Haskell types, the beginning of a Functor.

For example, the maybe type allows for “nullable” types.

```haskell
-- Maybe :: a -> Maybe a
data Maybe a = Just a | Nothing
```

Functors modify the arrows too.

Functors modify the arrows too.

```
class Functor m where
    fmap :: (a -> b) -> m a -> m b
```

Functors modify the arrows too.

```
class Functor m where
    fmap :: (a -> b) -> m a -> m b
```

```
instance Functor Maybe where
    fmap f (Just x) = Just (f x)
    fmap _ Nothing  = Nothing
```

```haskell
-- Take a name, and capitalise the words in it
formatName :: String -> String
formatName = unwords . map capitalise . words

-- Now allow for names that might not be there
-- for example from an optional input html field
fn = fmap formatName

fn (Just "haskell CURRY")   -- > Just "Haskell Curry"
fn Nothing                  -- > Nothing
```

The Functor pattern successfully separates concerns

The Functor pattern successfully separates concerns

Define a function that does something in your domain.

The Functor pattern successfully separates concerns

Define a function that does something in your domain.

Separately, define a Functor instance for a wrapper data type.

The Functor pattern successfully separates concerns

Define a function that does something in your domain.

Separately, define a Functor instance for a wrapper data type.

Use the wrapped values as if they were not wrapped!

Further reading

The official Haskell website is good
`https://www.haskell.org/`

Newish book all about the Maybe data type
`https://gumroad.com/l/maybe-haskell/`

Good series of blog posts on category theory
`http://bartoszmilewski.com/2014/10/28/category-theory-for-programmers-the-preface/`