Haskell: Programming with Functions

Overview of Day 1

Niki Vazou, Pablo Serrano
Clubes de Ciencia, Summer 2015

Announcements

- Complete the 2 questionaries
- 7+3 min presentation on Saturday
- Name cards
- Collaborations of 2

mathematical functions

same input gives same output

Functional Programming

no state

no-side effects

printing

observable effects,
other that return value

Side Effects

interaction with the world

modification of state

a function that calls itself

Recursion

haskell's way to do "loops"

maths's way to do "loops"

only evaluate things that I need

delay evaluation

Lazy Evaluation

not really intuitive

it is fine, since no side effects

the tiniest programming language

λ-calculus

equally expressive as any other pl

prevents errors at compilation stage

Type Checking

“If it compiles, it works!”

Types

Haskell has some build-in types:

any “small” integer
from –229 to 229

Int

Integer

any integer

True, False

Bool

Double

real numbers

Data Types

Let us build our own types,

Usually, a collection of labelled things

```
data Err a = Error a | Value a
```

Pattern Matching

The evaluation depends on the patterns

In faction definition

```
fact 1 = 1
fact n = n * fact (n-1)
```

Using case

```
fact n = case n of
           1 -> 1
           m -> m * fact (m-1)
```

Pattern Matching

The evaluation depends on the patterns

In faction definition

```
showErr (Error x) = “Error”
showErr (Value x) = show x
```

Using case

```
factErr n
 = case factErr (n-1) of
     Error x -> Error x
     Value x -> Value $ n*factErr (x-1)
```

Ordering of Definitions

In the same function is important

Are the following two the same?

```
fact 1 = 1
fact n = n * fact (n-1)
```

```
fact n = n * fact (n-1)
fact 1 = 1
```

Ordering of Definitions

In different functions is not important

Are the following two the same?

```
plus1 x = x + 1
plus2 x = x + 2
plus3 x = plus1 . plus2
```

```
plus3 x = plus1 . plus2
plus1 x = x + 1
plus2 x = x + 2
```

Comments

Why using comments?

```
--  This is an one line comment
```

```
{-
This is a multiple line comment
 -}
```

Identation

The empty space has meaning

The golden rule of indentation:

Code which is part of some expression should be indented further in than the beginning of that expression.

Today

Type classes

"The data type": Lists

Sorting Lists

Strings

The "non-silly" fibonacci function

Binary Search Trees

Type Inference: The theory break

Type Classes

interface that defines some behavior

```
class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool
```

What is the type of (==)?

Type Class Constraints

What is the type of (==)?

```
(==) :: (Eq a) => a -> a -> Bool
```

with int, bool fine because, with Err not

contraint propagation

two alternatives

Type Classes

```
instance Show (Err Int) where
  show x = showErr x
```

```
checkEq :: Eq a => a -> a -> Bool
checkEq x y = x == y
```

Some Notes

Identation: https://en.wikibooks.org/wiki/Haskell/Indentation

Sorting Lists

Strings

The “non-silly” fibonacci function

Show, Num, Eq, …