Hugh Baldwin
up2157117

Discrete Mathematics and Functional Programming

M21274
TB2

University of Portsmouth
BSc Computer Science
2nd Year

Contents

Part I

Discreet Mathematics

Part II

Functional Programming

Lecture - Introduction to Functional Programming

12:00 22/01/24 Matthew Poole

- For this module, we will be using the GHC (Glasgow Haskell Compiler), or more specifically it's interactive shell, GHCi

Imperative VS Functional Programming

- Most programming languages are imperative
  - Such as Python, JavaScript, C, etc
- Functional programming is another programming paradigm, which is based upon the mathematical concept of a function
- Imperative programming has state, statements (or commands) and side effects
- Pure Functional programming has no state, statements, or side effects
- A side effect is the change of state caused by calling a functionl assigning a variable, etc
  - This means that it is not always possible to predict the result of running a program, even with access to it's source code
- Since most programs need to cause a side effect (usually outputting data), most functional programming languages are not purely functional, but tend to organise the code such that only one part causes side effects

Functional Programming Languages

- There are two types of functional programming languages
- Pure
  - Languages such as Haskell
  - Has absolutely no state or side effects
- Impure
  - Languages such as ML, Clojure, Lisp, Scheme, OCaml, F#
  - Has some state or side effects, either everywhere or in a specific part of code
- There are also some functional constructs in major imperative langages such as Python, JavaScript, and more

FP Basics

Expressions

- An expression is a piece of text which has a value
- To get the value from the expression, you evaluate it
- This gives you the value of the expression
- e.g.

-
  ```
  Expression -> evaluate -> Value
  2 * 3 + 1  -------------> 7
  ```

Functions

- A function whose output relies only upon the values that are input into it
- The result will always be the same, given the same values
- This is the same as a mathematical function, which is where the name Functional Programming comes from

Haskell Basics

- In Haskell, all functions have higher precidence than operators
- This means that you have to explicity use brackets to ensure the correct order of operations