Outline

Today
  ▶ Programming paradigms? – What this module is about
  ▶ General module information

Programming paradigms

**Imperative** programming languages

Ada, ALGOL, BASIC, Blue, C, C++, C#, Ceylon, CHILL, COBOL, D, eC, FORTRAN, GAUSS, Go, Groovy, Java, Javascript, Julia, Lua, MATLAB, Machine language, Modula-2, Modula-3, MUMPS, Nim, Oberon, Object Pascal, Pascal, Perl, PHP, PROSE, Python, Ruby, Rust, Swift, Wolfram Language

**Functional** programming languages

Agda, Charity, Clean, Coq (Gallina), Curry, Elm, Frege, Futhark, Haskell, Hope, Idris, Joy, LISP, Mercury, Miranda, OCaml, Owl Lisp, Purescript, QML, SAC, SequenceL, Scheme, SML

What's the difference? An example

The **factorial** function takes a number \( n \) and returns

\[
    n \times (n - 1) \times (n - 2) \times \cdots \times 2 \times 1
\]

So \( \text{factorial}(5) = 5 \times 4 \times 3 \times 2 \times 1 = 120 \)

How would you write a program to compute \( \text{factorial}(n) \)?
Factorial the imperative way

def factorial(n):
    answer = 1
    current = 1
    while current <= n:
        answer = answer * current
        current = current + 1
    return answer

The imperative program tells a machine how to compute the answer

- Declare some variables
- Go around a loop
- Do these instructions each time

Factorial the functional way

def factorial(n):
    if n > 1:
        return n * factorial(n-1)
    else:
        return 1

The functional code still computes factorial but

- No variables declared
- No explicit looping (recursion used instead)

Functional Programming

Functional programming is a style of programming

It is very different from traditional imperative programming

- Imperative programs are a list of instructions to the computer
- Functional programs are more mathematical
  - No variables
  - No loops
  - No explicit idea of a sequence of instructions
  - Uses recursion

You can write in a functional style in any language

- But some languages are explicitly designed to support the functional style

A mathematician would write

\[
\text{factorial}(n) = \begin{cases} 
  n \times \text{factorial}(n-1) & \text{if } n > 1 \\
  1 & \text{if } n = 1
\end{cases}
\]
In COMP105 we study these two paradigms
- Imperative
- Functional

You already have experience with imperative programming, so our focus will be mainly on learning functional programming
- But we will take time to compare the two styles

We will learn Haskell, a pure functional language
- You cannot do imperative programming easily in Haskell

Why study functional programming?

1. Functional programming is becoming more important
   - There are industrial users of functional programming
   - Parallel systems are becoming more prominent, e.g., multi-core, GPUs. Functional programming is great for parallel systems

2. Learning functional programming will make you a better imperative programmer
   - You don’t have to use a functional programming language to do functional programming!
   - Sometimes the functional style is more appropriate
   - Languages like python and C++ support the functional style

3. Learning functional programming is good preparation for a Computer Science education
   - Algorithms in CS are often presented in a functional way
   - The functional paradigm is also used in the analysis of algorithms
   - Knowing functional programming will help you translate all that into real code

4. It’s fun!
   - Something entirely different to what you’ve seen before
   - You are free to disagree of course

Admin

Lecturer: John Fearnley
Email: john.fearnley@liverpool.ac.uk
Room: Ashton 322

The module has
- Three lectures per week
- One lab session per week

Attendance will be monitored
Labs in Computer Science

The department has several labs
▶ Located in the George Holt building
▶ You can use them at any time

Check them out this week!
▶ Make sure you can log in
▶ Ask at the CS helpdesk if you are stuck

Lab sessions

COMP105 labs start in week 2
▶ The timetable is still being finalized
▶ It will be available by the start of next week

Lab procedure
▶ Lab worksheets and are posted on the COMP105 website
▶ Staffed by John Fearnley and Ioannis Lamprou
▶ Make sure to catch our attention if you are stuck!

COMP105 Learning outcomes

At the end of the module, students will be able to

1. Describe the imperative and functional programming paradigms including the differences between them.
2. Apply recursion to solve algorithmic tasks.
3. Apply common functional programming idioms such as map, filter, fold and scan.
4. Write programs using a functional programming language.

Assessments

The module is 100% CA
▶ No exam next term!

There will be four assessments worth 25% each
▶ Three programming assignments
  ▶ Assignment 1: recursion
  ▶ Assignment 2: functional programming idioms
  ▶ Assignment 3: write a full program
▶ One class test
Provisional Schedule

Assignment 1
▶ Set during week 4, deadline during week 6

Assignment 2
▶ Set during week 7, deadline during week 9

Assignment 3
▶ Set during week 10, deadline during week 12

Class Test
▶ Monday the 3rd of December (week 11) at 11:00 in LIFS-LT3

Course Texts

There is no required text for the course. Recommended texts:
▶ Learn You a Haskell for Great Good! by Miran Lipovača
  ▶ £35.99 on Amazon
  ▶ Free online: http://learnyouahaskell.com/
  ▶ £29.99 on Amazon

Installing Haskell

Haskell is installed on the lab machines in CS
▶ This is the supported way of using Haskell in the module

You are welcome to install Haskell on your own hardware
▶ https://www.haskell.org/downloads
▶ There are three versions, any should be fine
▶ We can’t support you in this

The COMP105 Website

http://cgi.csc.liv.ac.uk/~john/comp105/
▶ All lecture notes will be posted before the lecture
▶ Lecture capture videos
▶ Lab worksheets
▶ Worked solutions for all exercises on the course
**What should you be doing**

Of course
- Attend lectures
- Attend labs

Self study: coding
- You *cannot* learn to code in a lecture
- The “ah-ha” moments will come at the keyboard
- Try to do a bit of coding every few days
- Every lecture (Lec 3 onwards) has exercises – try some!
- All solutions posted on the module website

**What should you be doing**

Functional programming is *completely* different from what you’ve seen before
- You won’t be able to fit it into an imperative mindset
- It might be like learning to code all over again
- It is easy to get frustrated

Stick with it!
- It will get easier as the course goes on
- Try to put aside your imperative experience

**How to get help**

If you get stuck during the module
- Ask questions during lectures
- Ask questions after lectures
- Ask questions by email
- Come to the labs and ask demonstrators
- Check the worked solutions on the website
- Read one of the texts
- Google/Stack Exchange

Don’t give up!

**Summary**

- Programming paradigms? – What this module is about
- General module information

Next time: What is a function?