COMP105 Lecture 1

Programming Paradigms

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 `factorial(5)` $= 5 \times 4 \times 3 \times 2 \times 1 = 120$

How would you write a program to compute `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

A mathematician would write

$$\texttt{factorial}(n) = \begin{cases} n \times \texttt{factorial}(n-1) & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$$

We can code this up in python too

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

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)

These are the hallmarks of functional programming

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

COMP105

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, eg., 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

Why study functional programming?

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