First Lecture: Course Policies and Course Overview

Luís Pedro Coelho

Programming for Scientists

January 13, 2009

University of Pittsburgh

Carnegie Mellon

Introduction

Who am I

- Luís Pedro Coelho (lpc@cmu.edu)
- Third year Ph.D. student in computational biology
- My office is 409D, Mellon Institute

Introduction

This course

- Meets: Tuesdays & Thursdays 6.30pm 220 SH
- Tuesday: Lecture
- Thursday: Lab Session
- Office Hours: Tuesday 4.30pm (or by appointment).
- Course Website: http://coupland.cbi.cmu.edu/pfs

Homeworks

There will be homeworks.

777 Rule

Project

Project

- There will be a final project, which will replace homeworks towards the end of the class.
- You can submit your own project, but I will have my own proposals.
- You can work individually or in small groups.
  Expectations will scale linearly.

Course Structure

Two Lectures

1. Tuesday session: a lecture where basic concepts are presented.
2. Tuesday session: a more lab type session, where we discuss particular technologies that let us implement the basic concepts.

In fact, for the first module, the two sessions will resemble each other.

Programming
for
Scientists

Course Overview

Class Modules

1. Programming & Software Carpentry
2. Basics of Scientific Programming
3. Advanced/Applied Topics

Course Overview: Module I

Programming and Software Carpentry

- Structured programming and general good programming
- Object-based programming
- Object-oriented programming
- Software Carpentry

Course Overview: Module II

Scientific Programming

1. Representation of numbers, memory usage
2. Numerical optimisation
3. Aspects of stochastic programming
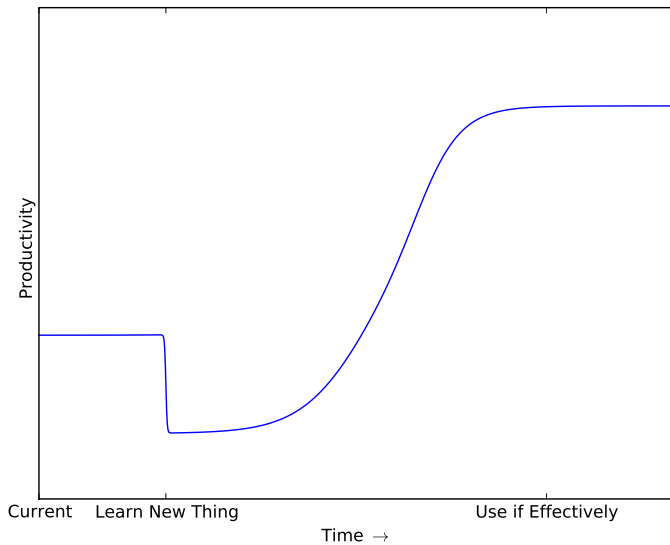4. Distribution of software

Course Overview: Module III

This is a mixed bag and subject to change (if you want to).

Advanced Topics

- Graphical user interfaces
- Concurrent programming
- Databases
- Interfacing multiple languages (lab session only)

Class starts here.

Glass's Law

Who Are Your Clients?

Different Classes of Users

1. Other programmers (or you, next week).
2. Actual users (maybe you).

In this class, we are focused on the programmers.

" Programs must be written for people to read, and only incidentally for machines to execute."

—Abelson & Sussman,
Structure and Interpretation of Computer Programs

Principles of Good Code

Principles of Good Code

- Correct
- Sufficiently efficient
- Robust
- Readable
- Tested
- Extendable

Code Organisation

Code Organisation

The first section is on how to organise code.

A Bit of History

Programming Languages

1. In the beginning, there was the bit.
2. Assembler
3. Fortran (1957, by IBM)
4. Structured programming (late 60s). Algol 68, C, . . .
5. Object-oriented programming. Simula, Smalltalk, C++, . . .

Procedural Programming

Procedural programming is code organised into procedures
(functions).

Functions

A Good Function

- Fits on the screen.
- Has a small number of obligatory arguments.
- Has a small number of outputs.
- Can be tested separately from the rest of the program.

Bad Code

```python
def xyf(x,y):
    t=x*0.5
    t2=t/2
    t3=x*p
    t4=t2*t2-y
    t4=sqrt(t4)
    t5=t2-t4,t2+t4
    return t5
```

Good Code

```python
def quadratic_solution(a,b,c):
    '''
    xp, xm = quadratic_solution(a,b,c)

    Computes the two solutions to
        a * x**2 + b * x + c = 0
    '''
    Root = sqrt(b**2 - 4*a*c)
    x_plus = (-b + Root)/(2*a)
    x_minus = (-b - Root)/(2*a)
    return x_plus, x_minus
```

Good Variable Names

Principle

Readibility

Heuristics

1. Don't be too short (except for $i, N, \ldots$).
2. Be conventional: loop iteration uses $i, j, k, \ldots$, counts use $N, \ldots$
3. Add a comment if you need.
4. Try to avoid need for comments.
5. Exception: implementing code from a paper.

Homework 0

Homework (for next week)

Run the following Python program and email me the results:

```python
email="YOUR EMAIL"
print email, hash(email)
```