

***Computer Science 380:
Building Modern Applications In
Python and C/C++***

Summer 2009

Lecturer: Richard T. Saunders

Cheating

- ◆ DO NOT CHEAT!!
 - ◆ I will give 0s and Fs
- ◆ You are all responsible for your own work
- ◆ This will be a fun course
 - ◆ Just do the work
- ◆ Remember: You can't cheat at a Google job interview
- ◆ I will try to help as much as I can if you are having problems (office hours, e-mail, etc.)

Office Hours

- ◆ Monday, Wednesday, Thursday right after class 5:15-6:15?
 - ◆ If there's people waiting, I'll hang out longer
 - ◆ Office is in Gould Simpson 834
- ◆ E-mail me to set-up appointments
 - ◆ saunders@lectura.cs.arizona.edu

Who am I?

- ◆ I am Richard T. Saunders
 - ◆ Work full-time at Rincon Research Corporation for last 14 years
 - ◆ 14 years of Object-Oriented C/C++ Programming
 - ◆ 7 years of Python Programming
 - ◆ Worked a Newmonics
 - ◆ Real-time Java startup
 - ◆ Lecturer at the U of A in Computer Science from 1992-1996, 2007, 2008, 2009

Materials

- ◆ Books
 - ◆ Core Python Programming (2nd Edition) by Wesley J. Chun (Required)
 - ◆ Good for Examples and introduction
 - ◆ Python in A Nutshell (2nd Edition) by Alex Martelli(optional)
 - ◆ Good for reference and different perspective
 - ◆ ...O'Reilly Books good in general...
 - ◆ (Don't need me, could learn out of book!)
 - ◆ course is an excuse to learn Python

Systems

- ◆ Course is ENTIRELY Linux based
 - ◆ All programming assignments turned in on Lectura (a Fedora Core 9/Linux machine)
 - ◆ You CAN install Python on Windows and do all your programming there but I don't support it. MUST TURN in on Lectura
- ◆ Those of who who feel “deficient” in Linux or Unix, we will spend some time getting up to speed

Class Format

- ◆ Monday, Wednesday, Thursday
 - ◆ 4:00-5:15
 - ◆ Take a break at 4:50 for 2-3 minutes (soda, rest room break, stretch)
 - ◆ First part of class typically lecture
 - ◆ Last part of class typically discussion, systems issues “like a lab”
 - ◆ Showing how Python work, showing examples, etc

Prerequisites

- ◆ C SC 127B or C SC 227
- ◆ Or come talk to me ...
 - ◆ Programming Experience in Java or C/C++ or Pascal
 - ◆ Experience with simple data structures
 - ◆ stacks, lists, queues
 - ◆ Experience with control flow
 - ◆ while loops, for loops
- ◆ Or you are a System Administrator...

Some Examples of Real Games and Applications written with Python

◆ Games:

- ◆ Freedom Force: Superhero game
- ◆ EVE online: MMORPG Sci-Fi game
- ◆ Star Trek Bridge Commander
- ◆ Minions of Mirth (<http://www.prairiegames.com>)
- ◆ more:

http://en.wikipedia.org/wiki/Python_software

◆ Applications:

- ◆ Mercurial: Revision Control System
- ◆ X-Midas: Digital Signal Processing Package

Real World

- ◆ RedHat
 - ◆ Installation scripts written in Python
- ◆ Google
 - ◆ uses Python and C/C++ for a lot of their software
- ◆ Rincon Research Corporation
 - ◆ uses Python and C/C++
- ◆ ...all sorts of jobs... (some can't talk about)

Course Premise

- ◆ **“Modern Applications Should Be Written In A Combination of a High-Level Dynamic Language and a Low-Level Compiled Language”**
- ◆ High-Level Language: focus on Python
 - ◆ applicable to other scripting/dynamic languages: Perl, Ruby, Tk, Javascript
- ◆ Low-Level Language: focus on C/C++
 - ◆ applicable to other compiled languages
 - ◆ FORTRAN, Java (*), Assembly

Why this split? (Part 1)

- ◆ Python: High-Level Dynamic Language
 - ◆ easy to use (hope to convince you!)
 - ◆ easy to write code, easy to read code
 - ◆ “fast enough” for most work
- ◆ C/C++: Low-Level Compiled Language
 - ◆ harder to use
 - ◆ harder to write code, harder to read code
 - ◆ blazingly fast
 - ◆ Derek Jones: “Squint and see the OP codes”
- ◆ Use the right tools:
 - ◆ 80% of application in Python, 20% in C/C++

Why this Split? (Part 2)

- ◆ Some applications may be fast enough to write entirely in Python
- ◆ BUT real-world applications have a **need for speed**
 - ◆ **Need C/C++ for that speed**

Example: Mercurial (1)

- ◆ Mercurial: Revision Control System
 - ◆ keeps ALL versions of ALL files for the lifetime of a project (all files from version 1.0, 1.1, 2.0, etc.)
 - ◆ “manages” version control for you
 - ◆ (quick example on board)

Example: Mercurial (2)

- ◆ Mercurial mostly written in Python
- ◆ BUT 3 files written in C:
 - ◆ base85.c, bdiff.c, mpatch.c
- ◆ Main repository manager uses the “diff” algorithm
- ◆ All Python Version: minutes to get a version of your software
- ◆ Mostly Python Version, some C: seconds to get version of software

Example: X-Midas

- ◆ X-Midas: DSP (Digital Signal Processing) language
 - ◆ core functionality written in FORTRAN/C++ component
 - ◆ glue components together with scripting language
 - ◆ Show example on board

Example: X-Midas (2)

- ◆ FFT: (Fast Fourier Transform)
 - ◆ critical it be fast
- ◆ Filters, Correlations, Convolutions
 - ◆ critical they be fast
- ◆ Experience in the domain (and the fact that a lot of DSP is based on FFTs, filters, etc) gives us knowledge of the **NEED FOR SPEED**

80-20 Rule

- ◆ AKA 90-10 rule, and 70-30 rule
- ◆ Rule of Thumb of most software:
 - ◆ 80% of the execution time of a program is spent in 20% of the code
 - ◆ Example on board
- ◆ Corollary: write the easy 80% in Python, write the hard 20% in C
- ◆ Rule of Thumb comes from “An Empirical Study of FORTRAN Programs” by Donald Knuth (1971)

80-20 Rule

Discussion:

Why?

Why not?

Costs?