Class 01: An Introduction to the Course

Held: Tuesday, January 26, 2010

Summary: We begin the class by considering a variety of motivating examples.

Related Pages:

- EBoard.

Notes:

- The Course Web is under development. Expect some updates over the next week or so.
- Assignment for Thursday: Introductory Survey and Such.
- Beautiful Code Reading for Thursday: Chapter 18 (Python’s Dictionary Implementation).
- Additional Reading for Thursday: Skim Python Pocket Reference (if you have it); Read/skim Python 101.

Overview:

- A bit about the course.
- Group programming exercise: Exponentiation.
- Beautiful Code: Regular Expressions.

About the Course

Software Design is a course with many (related) goals:

- You will learn new methodologies (or improve your knowledge of existing methodologies)
  - For designing programs (Object-Oriented Analysis & Design: OOA&D)
  - For working in teams on projects (Extreme Programming: XP)
- You will learn some new approaches to programming
  - Refactoring
  - Design patterns
- You will learn some tools to support your programming
  - Version control with subversion
  - Project control with make
- You will think about a variety of issues, small and large, in the design of code
  - We will often rely upon Beautiful Code for examples.
- You will learn a new language (Python)
- You will ground this learning in a variety of projects
  - Some toy projects
  - Some contributions to existing projects
Group construction of a moderate-sized project (TBD)

As you might expect, this course emphasizes the “practical” side of computer science.

Because this is a 300-level course, I have many expectations of my students.

- You should be *self-motivated*. I will not always attempt to use carrots or sticks to ensure that you do your work.
- You should be *responsible*. You should complete homework on time, contribute to class as appropriate, and so on and so forth.
- You should be *proactive*. When things don’t make sense, you should contact me with questions. You should also be willing to search the Web for answers.
- You should be *collaborative*. These days, few large programs are written by individuals. Hence, I expect that you will do most of your work with others.
- You should have the *appropriate background*. Ideally, every student in this class should be comfortable with
  - The three primary paradigms: Functional, imperative, and object-oriented.
  - The three primary languages we use at Grinnell: Scheme, C, and Java.
    - I realize that we’re going to have some differences in language background. You’ll need to be particularly careful to support each other in this. It’s also one of the reasons we’ll primarily use Python this semester.

- Introductions!

**Exponentiation**

- Consider the problem of computing $x^n$ for real $x$ and non-negative integer $n$.
- One strategy: A for loop
- A better strategy (in pseudocode):

  ```plaintext
  function expt(x, n):
      if 0 == n:
          return 1
      else if even(n):
          return square(expt(x, n/2))
      else
          return x*expt(x, n-1)
  ```

- So, let’s think about implementing this in C.
- Where do you begin?
- Once we’ve implemented it, how do we eliminate the recursion?

**Beautiful Code for Regular Expressions**

- Who is Brian Kernighan?
- Why does he consider this code beautiful?
- What is different in SamR’s implementation?
Why do you think SamR made these changes?

- What is the running time of this code?
  - Or what are some particularly problematic inputs?
- What do you see as potential issues in the design of this code?

- In-class assignment: Using this as a library, build a simple grep-like program.
  - We’ll look at your code together.