02-201: Programming for Scientists

Carl Kingsford
Fall 2015

1. Course Information

1.1 Course description

Provides a practical introduction to programming for students with little or no prior programming experience. Extensive programming assignments will illustrate programming concepts, languages, and tools. Programming assignments will be based on analytical tasks that might be faced by scientists and will typically include parsing, statistical analysis, simulation, and optimization. Principles of good software engineering will also be stressed. Most programming assignments will be done in the Go programming language, an industry-supported, modern programming language, the syntax of which will be covered in depth. Several assignments will be given in languages such as Python or Java to highlight the commonalities and differences between languages. No prior programming experience is assumed, and no biology background is needed. Analytical skills and mathematical maturity are required.

1.2 Course philosophy

Our first goal for this course is: if you successfully complete the projects of this course, you should be comfortable writing your own programs and perhaps even be able to contribute to a research group's projects. This skill will make you more valuable for internships and in-semester research projects, and it will really help you in later classes.

Our second goal is to convince you how much fun programming is! Writing a program is like solving a sudoku puzzle — programming tests (and builds!) your powers of concentration and logical thinking — but programming is better than sudoku since at the end you have something useful instead of a square of numbers. Programming also pays better as a career that being a sudoku solver.

This can be a challenging course for some because thinking "like a programmer" might be new to you. Rest assured: the difficulty of the course and the material we cover is well within that of a "typical" introductory programming course. Second, I think it's better to challenge students and assign grades with the knowledge that the material was challenging, rather than be easy and have you not learn as much. What's the point of taking a course in which you easily understand everything? That certainly doesn't expand your mind or make you a better thinker. This course is partially based on Princeton's COS 126, which is taken by freshman Integrated Science majors there. I'd like to quote the syllabus of one of Princeton's integrated science major courses, since it captures the philosophy of this course so well (if you replace "problem sets" with "programming assignments"):

> "You cannot learn the material without doing the problem sets. We have tried to write problems which are challenging, and hence probably require substantial effort, but without too much 'busy work.' On the other hand, some students feel more comfortable with some 'warmup exercises' that aren't so hard. These can be found in standard textbooks, which also provide a more traditional point of view on the material of the course."[1]

[1] `http://www.princeton.edu/integratedscience/curriculum/sample-syllabi/ISC231_1152.pdf`

We will provide some "warmup exercises" in the form of problem sets and quizzes. But the bulk of the course will be 7–10 programming assignments, which are chosen to be both interesting and not "busy work". These programming assignments are the heart of the course. They are carefully designed and are calibrated to be challenging (otherwise, what is the point?) but very reasonable for the time you will have to do them.

1.3 Pre-requisites

- Some analytical skills and mathematical background.
- No biology or programming knowledge is assumed.

1.4 Course Details

Submitting Assignments. The programming assignments (homeworks and projects) will be submitted via Autolab, and will be graded in conjunction with the policy outlined below.

Discussion Forum. An online forum is provided on Piazza as an area for discussion and questions. The forum will be moderated by the course staff who will respond to questions, but students are encouraged to help each other via discussion. However, specifics of assignment or project solutions should not be discussed — any hints will be provided by the teaching staff.

Programming Expectations. You are expected to produce clean, readable, and well-documented code. The coding practices should be consistent with what is taught in the lectures (e.g. consistent naming conventions, descriptive variable names, and short functions). A component of each assignment's grade will be based on coding style.

Programming Languages. Most of the course will be taught in the Go programming language. This is a relatively new language that has industry backing and that is simpler than many languages, but not too simple (it still contains important concepts like types, pointers, and type interfaces). However, unlike most introductory programming courses, we will not only look at a single language. Think of it as "comparative computational linguistics" — we may have a few assignments in a couple of other languages. The best way to understand programming concepts is to see how they are realized in a few different languages.

2. Tentative Topics

We hope to cover the following topics. We won't necessarily cover them in this order, however.

- **The machine and how programming languages abstract it.** The connection between programming language constructs and the underlying machine will be discussed. The syntax for the programming language "Go" will be covered in depth, and comparison between common programming language syntaxes will be given.

    1. Imperative programming constructs: functions, if-statements, loops (for, while), switch-statements, expressions
    2. Basic data structuring constructs: variables, arrays, strings, structs, types, and pointers
    3. Reading and writing files
    4. How to build Go large programs
    5. Basic execution and memory model (Von Neumann architecture)

- **Basic data structures and algorithm design techniques:** Several data structures and algorithms will be introduced.

    6. Linear data structures: arrays, lists, stacks, queues; binary search
    7. Dictionary data structures: binary search trees including tree traversals (DFS, BFS, pre-, in-, post-order); hash tables.
    8. Divide and conquer, recursion

- **The tools of programming:** Throughout the course, we will cover important tools for good software engineering practice.

    9. Assertions, preconditions, postconditions
    10. Code documentation
    11. Unit tests — testing small sections of code
    12. Debugging — strategies, common errors
    13. Profiling — figuring out what's taking so long

- **Abstraction and modularization:** How we control complexity through well-defined interfaces.

    14. Bigger units of code: Modules, namespaces, packages
    15. Type interfaces and user-defined types
    16. Object-oriented programming
    17. Design patterns

- **Parallelism:** Using Go's parallel features to let your program do more than 1 thing at a time.

    18. Goroutines
    19. Channels

- **Computability and the limits of computers.** Are there problems computers fundamentally *cannot* solve?

    20. Turing machines
    21. Halting problem and uncomputable functions
    22. NP and NP-complete problems

3. Coursework

Coursework will consist of extensive programming plus a midterm and final exam:

Homework assignments (programming and problem sets). (50% of your grade) Approximately 8 – 15 assignments designed to give you familiarity with a particular language feature, programming idea, common programming task, or algorithm.

Two midterms and final. (40% of your grade) The midterms and final exam will tests your knowledge of the material from the class, and your ability to read and design programs. The midterms will be held in class and the final held during the university's scheduled time. The midterm dates are:

- Midterm 1: October 9
- Midterm 2: November 13

The midterms will not be cumulative: midterm 2 will cover whatever we get to after midterm 1.

The final date and time is set by the university and will be posted when available. The final will cover all the material from the class.

Quizzes and participation (10% of your grade) We will have occasional problem sets or quizzes that serve to reinforce the concepts we have covered. Quizzes will be announced at least the class before.

Programming assignments must be completed on your own (unless noted) and turned in to the autograder by a given deadline. **No late assignments will be accepted.** All programming assignments will be graded by the autograder, which will check that your programs are outputting the expected results. In addition, a fraction of your grade for every programming assignment will be based on following appropriate programming style. The autograder will give you an estimated grade immediately, excluding the style portion of your grade. The TAs will subsequently go over your code and assign the final grade, taking into account coding style.

4. Collaboration Policy

You may discuss programming assignments with classmates. **However, you must not share or show or see the code of your classmates.** You must write your own code entirely. You can post general coding questions (with code snippets) on the discussion board.

You must write all programming assignments on your own and cannot share code with other students or use code obtained from other students. In addition to manual inspection, we use an automatic system for detecting programming assignments that are significantly similar.

You may *never* use, look at, study, or copy any answers from previous semesters of this course.

5. Other policies

Classroom etiquette: To minimize disruptions and in consideration of your classmates, I ask that you please arrive on time and do not leave early. If you must do either, please do so quietly. Laptop use is *encouraged* so long as you are trying examples or following along with the lecture slides. Use of laptops for other purposes is *strongly* discouraged. Texting during class is particularly discouraged.

Excused absences: Students claiming an excused absence for an in-class exam must supply documentation (such as a doctor's note) justifying the absence. Absences for religious observances must be submitted by email to the instructor during the first two weeks of the semester.

Academic honesty: All class work should be done independently unless explicitly indicated on the assignment handout. You may *discuss* homework problems and programming assignments with classmates, but must write your solution by yourself. If you do discuss assignments with other

classmates, you must supply their names at the top of your homework / source code. No excuses will be accepted for copying others' work (from the current or past semesters), and violations will be dealt with harshly. (Getting a bad grade is much preferable to cheating.)

The university's policy on academic integrity can be found here: `http://www.cmu.edu/policies/documents/AcademicIntegrity.htm`. In part it reads "Unauthorized assistance refers to the use of sources of support that have not been specifically authorized in this policy statement or by the course instructor(s) in the completion of academic work to be graded. Such sources of support may include but are not limited to advice or help provided by another individual, published or unpublished written sources, and electronic sources." You should be familiar with the policy in its entirety.

In particular: use of a previous semester's answer keys or online solution manuals for graded work is absolutely forbidden. Any use of such material will be dealt with as an academic integrity violation.