syllabus for

robots, games, and problem solving

comp 115

Instructor: Mark LeBlanc (mleblanc@wheatoncollege.edu)
`http://cs.wheatoncollege.edu/mleblanc`
Office: SC-1322
Phone: 286-3970 (on campus: x3970)

Office Hours: by appt. *or*
MTW 10 – 11am

Lecture: TR 2:00 – 3:20pm (SC B234)
Lab: T 6:30 or 8:00pm (SC 1349)

Online Text:

Interactive Python: How to Think Like a Computer Scientist
`http://interactivepython.org` by Brad Miller and David Ranum

How to Think Like a Computer Scientist

Online playground:

Codecademy.com (online Python course)

I also *strongly recommend* that you buy a 3-ring binder. I'll pass out lots of handouts.

Preamble:

Programming. There seems to be no end of the urgent need for more and more people to know how to script, how to write software, how to program. Mobile phones, mathematical models, embedded medical devices, video-cams, scientific experimentation, and modern desktop computers need software to do the things they do. This course is about learning how to program and how to do it well. We use the programming language Python because it offers a full range of rigor and elegance. Glad you are on board; like riding a bike, once you learn to program, you'll never forget. Let's cut code ....

Content:

Problem-solving techniques and algorithm development with emphasis on elementary objects, software reuse, and numerical methods. Topics include abstraction, design and decomposition, control flow, elementary data structures of lists (arrays) and dictionaries, and an understanding of how these features form the building blocks of user-defined classes of objects. Our emphasis this semester will be best practices for safe, efficient, scientific programming.

No previous programming experience is assumed. Out-of-class assignments and in-class labs emphasize the physical limitations of problem-solving-machines as well as techniques to write programs that are both safe and correct.

Your Grade:

Things to do	**Grading Percents**	**Frequency**
5 Programs	50%	TBA
Labs (approximately one each week)	15%	almost every Tue.
Quizzes (approximately 3-5)	10%	TBA
Exams (three)	5%	Exam1 Thurs Sept 27th
	5%	Exam2, Thurs Oct 18th
	5%	Exam3, Thur Nov 15th
Final Exam	10%	Wed., Dec. 12th, 9am

Computational Thinking

There is much misconception about computer science these days. For many, computing means using an iPhone. Yeah, that's cool … but that is not computing. Computer Science is the study of computation – what can be computed and how to compute it. The discipline encompasses "computational thinking" (Wing, 2006)[1], a universal metaphor of reasoning that defines how creative and imaginative humans use computation to facilitate communication, model complex systems, and visualize content.

Given the tangible, ubiquitous, embedded, and rapidly evolving nature of computing in our lives, the discipline of computer science faces the challenge of how to attract students to study within a discipline where some perceive they are already "experts". Clearly, relative to the previous generations that defined technology and computing by the electronic technologies at hand, the thumb-wielding, wireless generation of new students appears undaunted as they master and demand new hardware. But hardware is not computer science; a power-user is not a computational thinker. This course is about becoming a computational thinker.

Computational thinking is:
- a move away from "literacy" and toward "fluency," a broader concept including contemporary skills and intellectual capacities
- using abstraction and decomposition when attacking a large complex task
- choosing an appropriate representation for a problem
- modeling relevant aspects of a problem to make it tractable
- using invariants to describe a system's behavior succinctly
- developing heuristics to posit if and when an approximate solution is good enough
- using randomization to our advantage
- planning and scheduling in the presence of uncertainty
- search, search, and more search
- about ideas, not artifacts – it's not just the hardware and software that will be physically present everywhere, it will be the computational concepts we use to approach and solve problems, manage our lives, and interact with other people.

"Our civilization runs on software."
Bjarne Stroustrup, lead designer of C++

DETAILS

Late Submissions:

Due is due. Always turn in whatever you have **on time**. Something turned in on time even if unfinished is much better than not having it accepted because it is late. **Late is not an option**. (*Good, glad we can all agree with this*). Note that the onCourse (moodle) page associated with this course will have submission areas that are time triggered; once you are late, submissions will be blocked. Before the deadline, you may resubmit assignments and I will only review your last submission.

Since I know from experience that many students like to use the last night before a program is due (hopefully for testing), I will allow you to submit your programs until 4am the following day. For example, Program #0 is due Friday, Sept. 7th, but you can submit it online until 4am Sat. 8th. But, be careful! The course website (onCourse) makes it appear as if this first program is due on Saturday, but remember, that means Saturday at 4am!

[1] Wing, J. (2006). A Vision for the 21st Century: Computational Thinking, *CACM* vol. 49, no. 3, pp. 33-35.

Honor Code Revisited:
It goes without saying that all submitted work will be the student's own, in keeping with the Wheaton Honor Code, unless the lab has assigned groups. For in-class labs, you may get "help" from fellow classmates, but remember that all completed work must be your own. Use discretion; don't ask others for "the" answer or for lines of code. However, I do encourage you to discuss the problem in general, such as the type of statements or functions one might use. For programming assignments, your answers and software must be your own from beginning to end. Here is an analogy. Almost no one would every "use/steal" a line or two from another person's poem. Consider it the same with your programs. Don't "borrow/use" lines or sections of code from another student. Your program is (like) your poem; everyone's program should be unique. Be wise. If someone is asking you for too much help, be honest and remind them that your program is just that, *your* program.

Work outside of class: It is expected that you spend at least 2-3 hours on reading and practice problems for every lecture. This computes to at least 6 hours of work in the online texts per week. This should be done throughout the semester and not just when studying for quizzes or exams. The material is cumulative in a big way; for example, week 5 depends heavily on weeks 1 through 4. Also, it is expected that you also spend at least 6 hours per week on your current programming assignment. WARNING: Programmers typically underestimate the time it takes to complete a software project; 6 hours per week on your programming assignment may be one of those "underestimations."

In-class Labs: The labs are a critical part of the course. In almost all cases, the current lab will be preparing you for the current programming assignment. That is, if you complete and understand the lab, you should be well on your way to a solution for the programming assignment. We will sometimes engage in "pair programming" where two programmers take turns at the keyboard and problem-solve together. In order to best grasp the material in some labs, I strongly suggest that you completely redo any labs that you find difficult. (Read that last sentence again, unless of course you've already reread it once). Labs are typically due and graded at the end of the two-hour session. No attendance = no credit.

Quizzes: In addition to three exams spaced evenly over the semester, you will be quizzed on a regular basis. Quiz dates will be announced in lecture, e.g., in Thursday's lecture I may announce a Quiz on Tuesday. Each quiz may address material that has been assigned in the text, but not yet presented by the instructor. NOTE: you'll have to read the material and practice the sample problems BEFORE coming to class in order to be successful with the quizzes. Quizzes start at the beginning of a lecture and last approximately 10-25 minutes. After some quizzes, immediate (peer) grading will be followed by a discussion of a question on the quiz in order to provide you with immediate feedback on how you are mastering the material.

Quizzes and Exams: There will be no make-ups, nor will the lowest grades be dropped. If you are an athlete and/or you have a conflict with a quiz or exam date, please see me. When studying for quizzes and exams, have someone pick problems from the notes and text, mix them up, and present them to you as if you were taking a quiz or exam (not telling you from what section a problem comes). While practicing, do not peek at the textbook or an earlier solution. NOTE: it's easy to "read" code; it's much more difficulty to "write" code from scratch. Quizzes and exams will sometimes ask you to write code from scratch; practice ownership of the concepts!

Our online text, the Codecademy.com site, and the lectures will continually provide homework exercises for you to complete. Of course, if you do not work on them consistently, it will be very difficult to work through them all later on. Most of these exercises are small in nature, at least compared to your programming assignments. Let me attempt to motivate you with an analogy: if you had to compete in a diving competition (and you had never dove from a diving board before in your life), you would certainly *practice* many, many times *before* you allowed yourself to be watched. So it is with learning a natural language *and* a programming language: it will take *much* practice. And note that I do not mean practice so that you "almost get it", rather I mean a level of practice where you feel that you *own* most if not all of the techniques … at a level where you can apply them correctly.

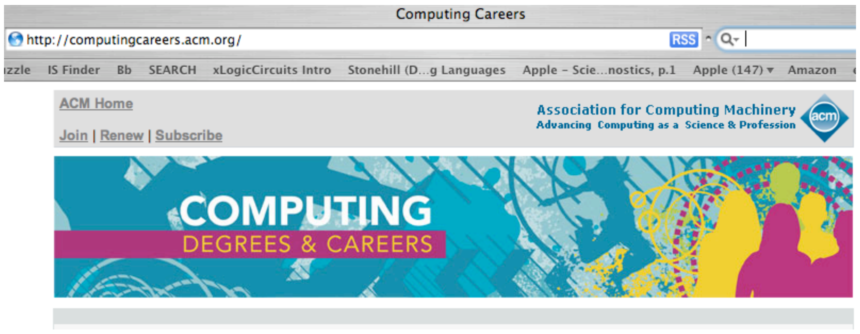
In computer science, if you are almost correct you are a liability.
Fred Kollett (1941-1997), Math/CS, Wheaton College

Accommodations for Disabilities
Wheaton is committed to ensuring equitable access to programs and services and to prohibit discrimination in the recruitment, admission, and education of students with disabilities. Individuals with disabilities requiring accommodations or information on accessibility should contact Susan Friedman or Kristine Smith, interim Accessibility Services Specialists, at the Filene Center for Academic Advising and Career Services. ~ accessibility@wheatoncollege.edu or (508) 286-8215 ~

HELP
I have listed my office hours on the syllabus, but I encourage you to be assertive: schedule a time to meet with me. Study, study, study … practice, practice, practice and talk about the material with me as often as you can.

Please don't wait too long before you see me;
a quick chat in my office can often clear things up.
I'm here a lot...

Wondering what you can do with a major or minor in computer science?
Check out this website to learn of the exciting potential of a career in computing ….
`http://computingcareers.acm.org`


Computing Careers
http://computingcareers.acm.org/
RSS
izzle IS Finder Bb SEARCH xLogicCircuits Intro Stonehill (D...g Languages Apple - Scie...nostics, p.1 Apple (147) Amazon
ACM Home
Join | Renew | Subscribe
Association for Computing Machinery
Advancing Computing as a Science & Profession
acm
COMPUTING
DEGREES & CAREERS