syllabus for

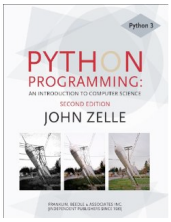
Robots, Games, and Problem Solving
COMP 115

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*
W 2:00-3:30, F 10:30-12

Lecture: MWF 9:30-10:20
Room SC 1349

Required Text:



Python Programming: An Introduction to Computer Science (2nd Edition)
by John Zelle, Franklin, Beedle & Associates Inc, 2010.

I also *strongly recommend* that you buy a USB flash drive to store your work as well as well as 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 we feel 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, 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. Avatars (virtual robots) in the ALICE environment will help you begin to think in terms of objects and the algorithms that operate on those objects. Games will provide examples for some programming concepts. However, 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
Practice (*aka* "homework")	20%	relentlessly
7 Programs	50%	TBA
Quizzes (approximately 5-8)	10%	TBA
Exams (two)	10% 10%	Exam1 Fri Feb. 24th Exam2 Fri April 20th
No Final Exam		

Note: Like employees at Google, I am allotting 20% of your time/grade to practice outside of class. Much of this work will be at your own pace; how much you do is up to you. If you want to be a good programmer, you must practice. The practice/homework will make you better.

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 for testing, I will allow you to submit your programs until 5am the following day. For example, Program #0 is due MON, Feb. 6th, but you can submit it online until 5am TUE, Feb. 7th. But, be careful! The course website (onCourse) makes it appear as if the program is due on Tuesday, but remember, that means Tuesday at 5am!

[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 assignment 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 your colleague 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 homework, 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 classmate. Your program is (like) your poem; everyone's program should be unique. Be wise. If a colleague is asking you for too much help, be honest and remind them your program is just that, *your* program.

Homework: It is expected that you spend at least 2-3 hours on reading and practice problems for every 50 minutes of lecture. This computes to at least 6 hours of work in the text 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. It is expected that you 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 often 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).

Quizzes: In addition to two exams spaced evenly over the semester, you will be quizzed on a (near) weekly basis. Quiz dates will be announced in lecture, e.g., in Friday's lecture I may announce a Quiz on Monday. 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 10-15 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!

Practice/homework

As mentioned earlier, like employees at Google, I am allotting 20% of your time/grade to practice outside of class. Much of this work will be at your own pace; how much you do is up to you.

I will continually provide homework exercises for you to complete. Many of these will *not* have traditional due dates, that is, you can finish and submit them throughout the semester. Of course, if you let these slide and do not work on them consistently, it will be very difficult to work through them all later in the semester. Most of these exercises are small in nature, at least compared to your programming assignments. The motivation is this: 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.

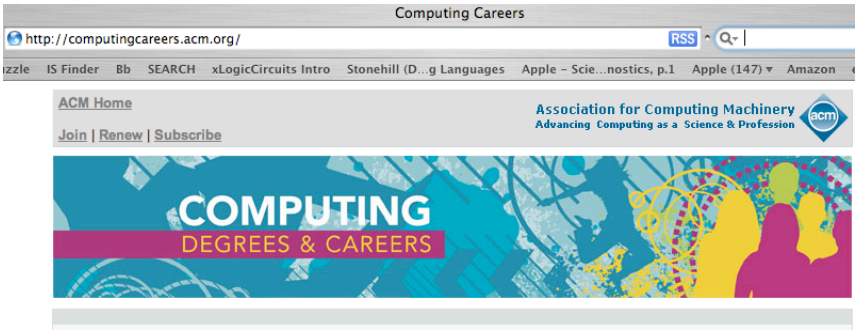
In computer science, if you are almost correct you are a liability.
Fred Kollett (1941-1997), Math/CS, Wheaton College

HELP

I have listed my office hours on the syllabus, but 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`



Week	Topic Note: At the start of each lecture, I will write on the board the exact pages that are associated with the material covered in this class meeting.
1	**Jan 25/27** Introduction to computational thinking Working in ALICE: - breathing life into your avatar avatar, *n.* "computer user's representation of himself/herself or alter ego" - object-oriented lingo; algorithms (computing "recipes") - control structures: sequential, conditional, and repetition (Did I say, repetition? Did I …)
2	**Jan 30 Feb 01/03** Problem decomposition The importance of documentation and good coding style Working with random number generators `if-else` and `while` statements handling events
3	**Feb 06/08/10** Introduction to Python variables, types, arithmetic operators, expressions a peek at BNF rules for valid variable names working with the Wing IDE and debugger Syntax, Logical, Runtime, and WTF "Bugs" *Monday, Feb. 6: a0 due – Your ALICE Avatar Plays "Rock, Paper, Scissors"*
4	**Feb 13/15/17** mathematical functions, writing mathematical expressions more Input/Output (I/O) - from the keyboard (stdin), to the console (stdout) limitations of computation binary numbers *Wednesday, Feb. 15: a1 due*
5	**Feb 20/22/24** selectional control – `if-elif-else` statements in-depth dangling else, trapping potential errors, short-circuit evaluation exception handling English to boolean conversions *Wednesday, Feb. 22: a2 due* *Exam I – Friday, Feb. 24*

6	**Feb 27/29 Mar 02** Repetitional control – `while`-, `for`-, `foreach`-loops computing with booleans
7	**Mar 05/07/09** Introduction to numerical methods ✓ algebraic results are not necessarily equivalent to computational results ✓ never compare two real's for equality, rather agree on "close enough" ✓ always trap o division by zero o bad arguments to library functions, e.g., `ln(-2), sqrt(-3)` o use explicit type conversions Fencepost problems Introduction to file Input/Output (I/O) *Friday, March 09: a3 due*
8	**Spring Break** – Monday, March 12 – Friday, March 16 – **Spring Break**
9	**Mar 19/21/23** More file I/O: reading files of DNA Working with/on strings Manipulating characters as data: encryption
10	**Mar 26/28/30** Defining your own functions User-defined libraries and object methods mutable and immutable parameters *Friday, March 30: a4 due*
11	**Apr 02/04/06** Intro to Data Structures: Lists (arrays) Python's "mutable sequences of arbitrary objects" *vs.* other languages Parallel arrays Passing arrays to functions
12	**Apr 09/11/13** Arrays revisited Linear Search Scope Introduction to "Big Oh" notation: O(lg(n)), O(n), O(n^2), O(n^3), …, O(2^n)

13	**Apr 16/18/20** Dictionaries – non-sequential collections *Wednesday, April 18: a5 due* Exam2 – Friday, April 20
14	**Apr 23/25/27** Object-Oriented Programming (OOP) Experimental Design
15	**Apr 30 May 02/04** Intro to Data Structures: two-dimensional (2D) arrays: matrices Summary of introductory numerical methods: *"the curse of being almost correct"* *Friday, May 4: a6 due* Evaluations