CS8: Computer Programming
Syllabus, Summer 2009

Dept. of Computer Science, UC Santa Barbara

Instructor:	Dr. Phill Conrad, Lecturer (PSOE), Dept. of Computer Science (Joint Appointment, College of Creative Studies).
TA:	Hakan Yildiz
Website:	http://www.cs.ucsb.edu/~pconrad/cs8
Course Meetings:	**Session:** B (August 3rd-Sept 11) **Lecture:** Mon, Tue, Wed: 1100am-1225pm, Phelps Hall 1401 **Discussion sections:** ESB 1003 (Cooper Lab), Thursdays Enrollment code: 18119: 11am-12:25pm Enrollment code: 18127: 12:30pm-1:55pm
Instructor Email	pconrad@cs.ucsb.edu
TA Email:	hakan@cs.ucsb.edu

Course Office Hours are shown in the table below
(This table may be updated later in the quarter, check syllabus link on the website for the latest updates).

	Mon	Tue	Wed	Thu	Fri
Who	Phill Conrad	TBA	Phill Conrad	TBA	TBA
Where	HFH 1113	TBA	HFH 1113	TBA	TBA
When	1:30-3pm	TBA	1:30-3pm	TBA	TBA

Prof. Conrad is also available by appointment—email him to request an appointment. Include CS8 in your subject line, and indicate your available times.

What this course is about

This course is an introduction to Computer Science, and programming. Computer Science is the study of **algorithms**. An **algorithm** is a well-defined, step-by-step sequence of instructions that can be used to mechanically determine the solution to some well-defined problem.

You probably use algorithms every day—for example:

- If you are looking in the index of a U.S. history textbook for "Gettysburg" you'll probably use an algorithm to find the entry quickly, assuming the index is alphabetized. Here the input to the algorithm is some topic, and the output is a list of pages on which that topic appears.
- If you are searching through a parking lot to either (a) find a parking space, or (b) determine that there are no spaces left, you probably use an algorithm to do that—again, without even thinking about what you are doing.

In the case of using an index, this is probably an algorithm you may have learned in grade school, and it has been so long since you learned it, that now you don't even think about it—you just do it. Finding a space in a parking lot—and knowing when to give up and look elsewhere—is "just common sense"; this probably isn't something you were ever "taught", or even have to think very much about. You just do it.

Computers don't currently have this capability—i.e. the capability to "pick up things by common sense"—and it seems unlikely that they will within our lifetime—unless there are major breakthroughs in the field of Artificial Intelligence. Such breakthroughs have been predicted for a while, but they haven't happened yet. (Maybe you'll be the one to figure out how to acheive this!)

So, for the time being at least, it falls to humans to design algorithms that computers can use to solve problems. In many cases, these algorithms are "just common sense"—the computer equivalent of looking for an empty parking space in a parking lot (and knowing when to give up). Algorithms like this are easy to design. Many of the algorithms we'll see in this course are like that.

In other cases, the algorithms are very complex, or very subtle, and coming up with them is a deep intellectual challenge. Furthermore, the impact of a better algorithm on society can be very large. For example, new algorithms in the field of computational science—modelling chemical and biological reactions with computer simulations—can lead to breakthroughs such as new drugs to fight disease, or renewable sources of energy.

Human languages such as English and Spanish are not very well suited for expressing algorithms—at least not for expressing them to a computer (that have their problems for communicating with humans too!). So, special languages are used. In this course, we'll learn the Python programming language. We choose Python rather than Java or C++ because:

- If you are learning your first programming language, Python is easier to learn than the others
- Learning Python provides a good foundation for learning C, C++ or Java
- If you only learn one programming language, Python is a good choice—in spite of being easy to learn, it is not a "toy" language by any means. It is widely used by scientists and web application developers just for starters. Many internal systems at Google are based on Python code.

This course provides you with the opportunity to become a pretty good beginning programmer, and be well prepared for an intermediate programming course such as CS16 (the first course that counts towards the CS major at UCSB, and which requires at least one quarter of prior programming experience.)

I say that the course "provides an opportunity", because you will only become an good beginning-level programmer if **you** put a lot of time and effort into this course—that is true no matter how much thought and attention I put in my lectures, assignments, and exams.

The swimming/guitar/painting analogy

You cannot learn to swim, play guitar, or paint from a textbook or a lecture. You can only

- learn swimming by spending many hours in the pool,
- guitar by spending many hours playing the instrument
- painting, by spending many hours putting brush to canvas.

The same is true of programming. Programming is not a serious of facts to be memorized—you cannot "cram" for a computer science exam. You must practice, practice, practice.

You may find the workload heavy

As a result, the workload in this course may feel heavy. It may even feel unreasonable compared to your other courses.

However, I assure you that it is not unreasonable, given the goal of making you an skilled beginning programmer.
Programming is a skill, and the only way to get good at it is lots and lots of practice, which takes lots and lots of time.

Special note for summer session programming courses

The compressed 6 week term means your work load—reading assignments, homework, and programming assignments—which is already heavy in this class, will have to 166% of what you'd experience during the normal quarter, just to keep up.

The usual rule of thumb is 8-12 hours per week for a normal college class*. That means you should expect, at a minimum to put in 14-20 hours per week on this summer course, on top of the 6 hours you spend in lecture and lab each week.

Finally—note that just as in a math class, everything we do builds on all the work that came before. So, *everything* is *cumulative*—so, you can't afford to miss any classes unless absolutely necessary.

(*see CliffsNotes.com. How much outside class study time is recommended for every hour of class time for college freshmen? 26 May 2009 <http://www.cliffsnotes.com/WileyCDA/Section/id-305397,articleId-7601.html>)

What you need to learn to become a skilled beginning level programmer.

So, what is it that you need to know to be an skilled beginning-level programmer in Python? Here's the list of what you'll need to be ready for CS16 (the next programming course):

- Problem solving
  - breaking down a problem into a sequence of steps
  - abstracting specific problems into general ones and finding general solutions
- Memory concepts
  - variables, primitive vs. reference variables, name, type, value
  - assignment statements
  - scope of variables
- Control structures
  - for loops, if/else, while loops
- Lists in Python (similar to arrays in other languages)
  - index vs. value, finding sum, min, max, average, count of elements matching some condition, making a new list of elements containing only those that match some condition

- Functions
  - function call vs. function definition
  - formal vs. actual parameters (arguments)
- Testing
  - How to test your code
- Input/output concepts
  - Writing to the terminal
  - Reading from the keyboard
  - Reading and writing to files
  - Neatly formatting output
- Program style
  - How to write code that other people can read and understand

Catalog Description

Introduction to computer program development for students with little to no programming experience. Basic programming concepts, variables and expressions, data and control structures, algorithms, debugging, program design, and documentation.

Prerequisites

None: but we do expect familiarity with basic concepts from high school algebra, geometry and trigonometry—we'll review those as needed.

Textbook(s)

Main textbook: Python Programming in Context

Authors: Bradley N. Miller, and David L. Ranum
Publisher: Jones and Bartlett, 2009.
ISBN-10: 0-7637-4602-5   ISBN-13: 978-0-7637-4602-5

For more information, See: http://www.cs.ucsb.edu/~pconrad/cs8/09M/textbook/

Grading

50% Assignments/Quizzes/Homework + 30% Midterm Exams (2 at 15% each) + 20% Final Examination

Quizzes may occur at anytime, announced or unannounced. Missed quizzes may not be made up.
Thus **attendance is required**, and **reading the assigned readings is required.**

A conventional 10 point scale will be used to map your numeric average into a letter grade, with the lower three and upper three points of each range representing plus/minus.

grade >= 93	A	73<= grade < 77	C
90 <= grade < 93	A-	70<= grade < 73	C-
87 <= grade < 90	B+	67 <= grade < 70	D+
83<= grade < 87	B	63<= grade < 67	D
80<= grade < 83	B-	60<= grade < 63	D-
77 <= grade < 80	C+	grade < 60	F

This 10 point scale represents the *minimum* letter grade you will be assigned—at the instructor's discretion, the letter grade scale may be altered *in the students' favor* if this will be better reflect the students' mastery of the material. Thus, *if* there is a "curve", it will be applied at the *end*, not to individual assignments.

Policies

Attendance

This course moves quickly. So attendance is very important.

We'll be trying to master the material from about 9 chapters in the book over six weeks. We'll sometimes cover two or even three chapters in a given week. We need to go at that pace, last few lectures the quarter, you can't really start anything new, because there isn't time to put it into practice with programming assignments. If you don't put it into practice, you aren't very likely to learn it in any way that is going to stick with you, so there isn't much point in just "going through the motions".

As a result, **there will be something you have to turn in at almost every class**. In this way, attendance is taken, and required.

These things you have to turn in will be a combination of in-class activities, and homework completed outside of class, but handed in on paper during class.

Missing in-class activites.

If you miss a class, you miss the opportunity for the points on that in-class assignment, or homework that was due. Period.

There is no makeup, except for

- excused absences arranged and agreed to by the instructor **in advance**, for official UCSB activities
- one "sick-day/personal day" per student, per quarter (see below)

To make up an assignment from a "sick-day/personal-day", you must **email me within 48 hours of the absence**, to make an appointment to make up the assignment during the next scheduled office hours following your absence (or at an appointment time to be negotiated, if you have a conflict with those hours.) This make up must happen within one week of the absence, or 24 hours before the final exam, which ever is earlier.

In rare cases, if there is a documented family emergency, documented extended illness, documented required court appearance, or other situation beyond the students' control (with documentation) the instructor may grant additional make up days entirely at the instructor's discretion—but this is **not** a guarantee or a right.

A special note about collaboration

As mentioned above, one of the things we really want to convey in this course is that real-world software development is very seldom an 'individual sport'—is it much more often a 'team sport'. Companies want to hire CS and CE graduates that know how to collaborate with others on producing software.

In the CS Department at UCSB, we understand the value of this. However, it puts us in a tricky position.

On the one hand, we want to encourage working together in ways that help you develop your skills and teamwork, and help you understand that programming can be a social, collaborative, creative activity—not something done only by loner nerds in cubicles. The sooner you start with activities such as pair programming, code reviews, and other collaborative software development activities, the more skill you'll develop, and the sooner you'll be ready for the real world. Plus, for many people, working together with others is a lot more enjoyable and fun than being a loner.

On the other hand, we need to avoid any situations where freeloaders are "coasting" through courses by leaning too much on others—never developing independent skills as programmers. This situation creates huge problems. Mostly it is damaging to the freeloaders themselves, who eventually crash and burn, perhaps far too late to choose another career path without significant difficulty. However, it also creates problems for everyone else—some hardworking students become demoralized by the unfairness of it all, and the value of a UCSB education is diminished by the freeloaders' lack of accomplishment.

Thus, we must strike a balance.

Our emphasis on collaboration means:

- We will try to create opportunities for you to work in pairs on assignments—in some cases, we may even require it.
- We will try to create opportunities for you to develop skills at talking about and reflecting on your own code and other people's code in small groups (code reviews).
- Some in-class assignments will permit discussion with other students.

It doesn't not mean, however:

- That you can "just copy" homework or code from others and claim it as your own work.
- That you can work together on assignments where you've been specifically told not to.

The bottom line:

- We'll try to be very specific about what kinds of collaboration are permitted, and what kinds of collaboration are not permitted, and are considered academic dishonesty.
- If you are not sure about whether some kind of collaboration is permitted or not, it is your responsibility to **ask questions.**

A final note: the emphasis on collaboration in this course does not necessarily extend to other CS courses you make take in the future.

- Each course will have its own policies, and the default policy is still: **no collaboration.**
- Please be sure you understand each instructors policy on collaboration carefully, and don't assume it will be the same as that from previous courses.
- And, finally, be sure to review the UCSB Academic Honesty Policy. You should read and understand the UCSB policy on academic honesty listed below. You should also understand that I take academic honesty and personal integrity very seriously, and will do my best to uphold and enforce this UCSB policy.

UCSB Policy on Academic Honesty

> It is expected that students attending the University of California understand and subscribe to the ideal of academic integrity, and are willing to bear individual responsibility for their work. Any work (written or otherwise) submitted to fulfill an academic requirement must represent a student’s original work. Any act of academic dishonesty, such as cheating or plagiarism, will subject a person to University disciplinary action. Using or attempting to use materials, information, study aids, or commercial “research” services not authorized by the instructor of the course constitutes cheating. Representing the words, ideas, or concepts of another person without appropriate attribution is plagiarism. Whenever another person’s written work is utilized, whether it be a single phrase or longer, quotation marks must be used and sources cited. Paraphrasing another’s work, i.e., borrowing the ideas or concepts and putting them into one’s “own” words, must also be acknowledged. Although a person’s state of mind and intention will be considered in determining the University response to an act of academic dishonesty, this in no way lessens the responsibility of the student.
>
> *(Section A.2 from: **http://www.sa.ucsb.edu/regulations**, Student Conduct, General Standards of Conduct)*

Disclaimer

This syllabus is as accurate as possible, but is subject to change at the instructor's discretion, within the bounds of UC policy.