Course Description
This course introduces the basic models of computation, such as iteration, decision-making, searching, divide-and-conquer and simulation through exercises in Python's programming. Students will write programs in several application areas, e.g. mathematics, graphics and biology. This is a good course for those who have no programming experience or for scientists who wish to learn Python, a popular scientific scripting language.

Course Goals
1. To teach students how to think like a computer scientist.
2. To teach Python programming; Python is a scripting language that is used in many scientific disciplines.
3. To satisfy the Technology general education distribution area.
4. To engage students in writing narratives about the programs they write.
5. To engage students in some of the general education capabilities.

Distribution
This course satisfies the Mathematics and Technology distribution area.

It is designed to address the following two capabilities:
1. Quantitative reasoning and
2. the use of technology to further learning.

Rationale for Offering This Course
We have two reasons for offering this course:
1. If you have not programmed before, this will be a relatively gentle introduction to both programming and computer science.
2. If you are a natural scientist, many Python libraries have been written for use in the computations for many fields (e.g. astronomy and genetics); you may make use of such tools in your Python programs.

I don’t want to kid you; learning to program in any programming language is a challenge. You will be learning how to use many tools (e.g. an editor, the Python interpreter, and Python libraries) as well as how to write Python programming. You will be doing lots of programming.

Why Python? We use Python for two reasons:
1. Python is a programming language having a vary simple syntax and so it does not get in the way of the computations one wants to express. We can

---

1 The name derives from Monty Python, not from the snake.
focus on computational models rather than on programming language intricacies.

2. Python is a popular programming language in many disciplines. Scientists working in these disciplines have developed extensive libraries of Python tools for doing their (and your) work.

**Textbooks**

The principal text for this course is:


Other texts that you might find useful, and which you will find on line include


**Topics**

- Computers and programs; computability. (1 day, Zelle 1, Elkner 1)
- Editing (1 day, Zelle Appendix B)
- The assignment statement and numbers; approximation. (1 week, Zelle 2 and 3, Elkner 2)
- Simple control: conditionals, iteration and recursion. (1 week, Zelle 3, Elkner 3 - 6)
- Simple graphics programming using objects. (1 week, Zelle 4)
- Strings. Encoding and encryption. (1 week, Zelle 5, Elkner 7)
- Tuples and lists. (1 week, Zelle 5, Elkner 8 and 9)
- Functions, common decision patterns and iteration patterns. (1 week, Zelle 6 - 8)
- Simulation and program design (2 weeks, Zelle 9)
- Searching and recursion. (1 week, Zelle 13)
- Objects, classes, messages and methods (2 weeks, Zelle 10, 12, Elkner 14 and 16)
- Data collections (1 week, Zelle 11, Elkner 15, 17 - 20)
- Social issues arising in computing. As socially interesting IT topics are raised in our morning newspapers, we shall discuss them in class.

**Programming Exercises**

[These are example exercises, some of which have been taken from Eric Grimson and John Guttag, *Introduction to Computer Science and Programming*, an MIT course 6.00, taught in Fall 2008 and published under a Creative Commons License as MIT Open Courseware.]
In this section, we identify the capabilities that are addressed. We offer more detail on how capabilities are addressed in the sample exercise statements, which are attached.

We also identify the time and point value for each.

Suggested readings for each exercise are included in each of the (attached) sample exercises.

1. A program for computing and printing the first fifty Fibonacci numbers. Capabilities: quantitative reasoning. 1 week. 5 points. [Sample exercise included.]
2. A program for counting instances of a word in a body of text. Capabilities: quantitative reasoning. 1 week. 5 points.
3. A program that performs (simple) encryption and decryption on a piece of text. Capabilities: quantitative reasoning. 1 week. 5 points.
4. A program that counts all of the distinct words appearing in a body (file) of text. Capabilities: quantitative reasoning and the use of technology for further learning. 1 week. 5 points.
5. A (simplistic) graphical program for playing Conway’s Game of Life. Capabilities: quantitative reasoning and the use of technology for further learning. 10 days. 10 points. [Sample exercise included.]
6. A program for simulating the game of odd-or-even. Two little boys or girls flip quarters and one calls “odd” (one coin heads, one tails) or “evens” (both coins either heads or tails). Whoever wins gets both quarters. Over many flips, each of odds and evens should come out evenly; one should go home with the same amount of money one comes with. Plot the results of running many trials, each with many flips, using various types of graphs. Capabilities: quantitative reasoning and the use of technology for further learning. 10 days. 10 points.
7. A program to efficiently search for sequences that match or nearly match some target string of characters in some large sequence of characters. (This program can be used to search for similar gene sequences or protein sequences in biology.) Capabilities: quantitative reasoning and the use of technology for further learning. 2 weeks. 15 points. [Sample exercise included.]
8. A program that plays the popular word game “ghost”. In ghost, a player starts with a single letter, the other player adds a letter, and play goes back and forth adding letters to the sequence until the game ends. A game ends when a legitimate word is spelled; the last player to add the final letter forming the word loses. Given a dictionary of legitimate words, the program plays to win against a human player. Capabilities: quantitative reasoning. 2 weeks. 15 points.

Grading
Programming Assignments 70%
• Programming assignments will give you the opportunity to practice using the programming tools you learn in class.
• Each programming assignment must be accompanied by a narrative, describing the design choices made, problems encountered (and how you solved them), and how you tested your program(s). The narrative will account for at least 30% of an assignment’s grade.

<table>
<thead>
<tr>
<th>Examination</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Midterm Examination</td>
<td>10%</td>
</tr>
<tr>
<td>Final Examination</td>
<td>20%</td>
</tr>
</tbody>
</table>

**Accommodations**
Section 504 of the Americans with Disabilities Act of 1990 offers guidelines for curriculum modifications and adaptations for students with documented disabilities. If applicable, students may obtain adaptation recommendations from the Ross Center for Disability Services, Campus Center 2nd Floor, 2100 Street, Room 2010, 617-287-7430. The student must present these recommendations and discuss them with each professor within a reasonable period, preferably by the end of Drop/Add period.

**Academic Honesty**
All students are expected to follow the University’s Code of Student Conduct. If you are caught cheating, we will follow the guidelines for punishment outlined in the code.

When you turn in work that you have discussed with someone, or which contains ideas that you found in a book, you must indicate that fact. We expect you to talk to each other and to read materials other than those assigned. We also expect to see in your work evidence that you have done so. Learning to acknowledge intellectual debts is part of learning. You should be reading, talking to each other, and telling the world that you have done so. When group work is called for the group solution should note whenever a part of the project was done by only a part of the group.

Some kinds of sharing, however, are unacceptable. You may not use the computer to copy someone’s work and submit it as your own -- even if you acknowledge that theft! You may not have your friends do your work for you. Versions of some of the assignments in this course may have been given in previous years. You may not use answers to those assignments.

---

2 Much of the wording here is from Ethan Bolker.