Sarah Lawrence College — Department of Computer Science

Introduction to Computer Programming

Course Syllabus[1] — Fall 2017

http://science.slc.edu/msiff/courses/cs1/

Meeting times

Lectures: Mondays *and* Thursdays, 1:30–3:25, Science 103

Labs: Wednesdays, Science 104: 11:05–1:00 *or* 1:30–3:25

Students will attend one of the labs, not both. Official lab placement will be made after the first lecture (Monday, September 4), prior to the first day of lab (Wednesday, September 6).

Instructor

Michael Siff, Faculty of Computer Science
Science Center 213
`siff.michael@gmail.com`
914/395-2490

Description

This lecture presents a rigorous introduction to computer science and the art of computer programming, using the elegant, eminently practical, yet easy-to-learn programming language Python. Students will learn the principles of problem solving with a computer while gaining the programming skills necessary for further study in the discipline. The course emphasizes the power of abstraction, the theory of algorithms, and the benefits of clearly written, well-structured programs.

Fundamental topics include: how computers represents and manipulate numbers, text and other data (such as images and sound); variables and symbolic abstraction; Boolean logic; conditional, iterative, and recursive computation; functional abstraction ("black boxes"); and standard data structures such as arrays, lists and dictionaries. The course covers introductory computer graphics and how to process simple user interactions via mouse and keyboard. By the conclusion, students should also be exposed to the role of randomness in otherwise deterministic computation; basic sorting and searching algorithms; and some principles of game design.

Toward the end of the semester, topics will turn to aspects of programming somewhat larger projects and so will introduce file processing; modules and data abstraction; and object-oriented concepts such as classes, methods, and inheritance.

Weekly hands-on laboratory sessions will reinforce the programming concepts covered in class.

Open to any interested student.

[1] Draft syllabus as of August 26, 2017, for consideration during registration.

Required text

The required text for this course is *free* (as in speech):

> Peter Wentworth, Elkner, Allen B. Downey, and Chris Meyers. *How to Think Like a Computer Scientist: Learning with Python 3 (RLE).* `openbookproject.net/thinkcs/python/english3e/`

An *interactive* version of the text is also available at:
`interactivepython.org/runestone/static/thinkcspy/index.html`

Python

The course relies on version 3 of Python (which is very similar, but definitely not the same as version 2). Students are encouraged to install the latest official release (3.6.2) from the Python Web site (`python.org`). It is a fairly straightforward installation process regardless of whether the machine is running Max OS X, Windows or Linux. The installation includes an interactive development environment (IDE) for Python called IDLE — it allows users to write and execute programs from within the same piece of software. IDLE is installed on all the iMacs in the computer lab (Science 104). Time will be set aside during the first lab meeting (Wednesday, September 6) to assist students with installing Python on their own machines if they have not already done so.

Tools

1. repl.it
2. Snakify
3. PEP8 online

For some of these tools (notably repl.it and Snakify), it will be helpful for you to create accounts using your @gm.slc.edu email. Please note before signing up for either of these services you should read over their privacy policy (a good practice when using the Internet, in general). If you object to their privacy policies, let me know and we can attempt to make accomodations.

Examinations

There will be two in-class examinations, one in the middle of the semester (Thursday, October 26) and one at the end of the semester (Thursday, December 13). The exams will not ask students to write code (except, perhaps, to complete a line or two); but they will ask them to *read* code. *Important: No make-up exams will be given.*

Homework

Regular reading. Students are expected to keep up with assigned readings. There will be an assigned reading for just about every class (lectures and labs). The readings will not be long, but they are fairly technical which means to get the most out of the readings it is best not to rush, and, ideally, to sit near a computer that has Python installed so that the reader may experiment as she reads. (In fact, given the format of the book, students might read the text on the same computer. Even better, there is a companion version of the text that allows readers to experiment with Python *as they read.*) It may also be useful to revisit some readings after they have been discussed in class to reinforce specific concepts.

Programming assignments. There will be eight programming assignments; they are the primary form of written homework. Programming assignments are due Fridays at 5pm. (The first assignment is due September 8; the last December 15.) Programming assignments, unless explicitly stated otherwise, are to be completed independently (see more under the Policy section).

Problem sets. There will be two problem sets (the first due Monday, October 23, the second Monday, December 11). The problem sets serve as untimed practice for the two examinations. Each problem set will have two due dates. By the start of class on the first date, students need to have seriously attempted each of the assigned problems; time will be set aside during that class to answer questions concerning some of the problems (and hints may be offered to the solution of others). A typed, clearly organized, well written set of solutions to the problems is due on the latter date. Work on problems sets should be performed independently; all questions should be addressed directly to the instructor.

Late submissions will not be accepted.

The calendar for written assignments is as follows:

#	due date	theme
0	09/08	getting started
1	09/15	fruitful functions
2	09/29	ifs and loops: text-based REPL game
3	10/13	lists: code making and breaking
	10/23	*first problem set*
4	11/03	2D-arrays: Conway's Game of Life
5	11/17	dictionaries: randomly generated poetry using Markov chains
6	12/01	event handling: game project, part I
	12/11	*second problem set*
7	12/15	objects & inheritance: game project, part II

Laboratories

Laboratory sessions will be held weekly, on Wednesdays (except immediately preceding Thanksgiving — school is closed that day). Labs on Ocotber 25 and December 13 will be reserved for reviewing for the two exams (Thrusday, October 26 and December 14.) The first lab will be September 6; the last regular (hands on lab) will be Wednesday, December 6.

Each student will be placed in one lab section according to their schedule constraints and available space in that section (due to physical limitations of the computer classroom — Science 104 — labs are capped at 15 students). Once placed in a lab section, you will remain in that section for the semester.

The purpose of the labs is to offer instructor-supervised, hands-on experience writing Python programs, to reinforce concepts covered in class during the past week, and to prepare students for upcoming programming assignments. (New material will rarely be introduced during lab meetings.) At the beginning of each lab, students will be directed to a set of programming exercises to work on individually or with a partner. If assigned a partner, the pair should collaborate equally on typing at the keyboard and writing, testing, and debugging code. The lab problems will not be formally graded, but students are expected to finish them on their own time if necessary; several of the exercises will pertain directly to subsequent homework assignments. Familiarity with the solutions to the labs will improve comprehension of general programming principles and practices, streamline future labs and programming assignments, and prepare students for problem sets and examinations.

The calendar for laboratories is as follows:

#	**dates**	**topics**
0	09/06	intro to Python programming in IDLE
1	09/13	fruitful functions
2	09/20	ReadEvalPrintLoops (REPLs)
3	09/27	loops, strings, slices
4	10/04	lists, bits, binary
5	10/11	more fun with lists
6	10/18	2D-lists, matrices
	10/25	*review for first exam*
7	11/01	randomness
8	11/08	sets, dictionaries
9	11/15	event-driven programming
	11/22	*no lab (Thanksgiving)*
A	11/29	objects, classes
B	12/06	inheritance, more OO
	12/13	*review for second exam*

Attendance

As stated in the Student Handbook: “Consistent attendance at all classes and conferences is expected.” Unexcused absences or persistent lateness, whether in class, lab, or group conference, may lead to reduced credit. In this era of ubiquitous connectivity, an absence for which a student did not inform the instructor of in advance is *unexcused*. (That is *not* to say that merely informing in advance will excuse an absence, but it will often help.) As this is a lecture course, students are responsible for making sure they are marked present. If they arrive late or hide silently in the back row (not that the instructor recommends ever doing either), they should check in with the instructor at the end of class.

Checking email and the course Web page

Students are expected to check their Sarah Lawrence (`@gm.slc.edu`) email account frequently (upward of once per day). In addition, students should know that the combination of what is communicated in lecture, lab and conference and what is presented via the course Web page and course-related email forms the only authoritative syllabus. (In other words: *The course may depart from this document.*)

Plagiarism

Copying code, whether electronically (as in downloading, file sharing, copying and pasting) is plagiarism. Retyping someone else's solution is also plagiarism.

Paraphrasing someone else's code by using different spacing and variables names is still plagiarism.

Web searching (whether using Google, StackOverflow or other sources) for solutions to assigned problems on line (for example, googling "python counting letters in a string") is prohibited.

If you are at all unsure if how your work habits are in keeping with the spirit of "working independently" be sure to check with the instructor.

Additional assistance

Many students may find find some of the material daunting. They should not fear. If a student finds it difficult, most likely so does the vast majority of the rest of the class. Anyone who feels the course is moving too fast or if is having trouble understanding a particular topic should talk to the instructor in class or out, in person or via email.

Grades

The instructor for this course does not put much stock in grades. Discouraged questions:

> "What kind of grade am I looking at now?"
>
> "How much is this assignment worth?"

Encouraged questions are those that deal with course content and learning fundamental concepts:

> "Why do programmers count from zero?"
>
> "Can you suggest exercises for me to feel more comfortable with recursion?"

Possibly acceptable:

> "Am I at risk of losing credit?"

(However, if students are in academic trouble in this course, the instructor will inform them.)

A rough formula for grading exists, but it will likely seem inscrutable. It is of this form (only more complicated):

```
(sum(D6:N6) - min(E6:J6) + sum(M6:N6))/12.75 + O6
```

Elements that factor into your evaluation:

- programming assignments
- examinations
- problem sets
- participation (to the extent possible in a lecture course)

- attendance
- effort and demonstrated improvement

The course is *not* graded on a curve. In principle 100% of the class may receive an A — in practice, that will not happen, but there is no predetermined distribution for grades.

To rephrase: Focus on learning, forget about grades.

Calendar

week	date	topics
1	09/04	what is computer science? why Python?
	09/07	fruitful functions
2	09/11	definite (for) loops
	09/14	Boolean basics and conditionals
3	09/18	indefinite (while) loops
	09/21	strings, basic I/O, REPLs
4	09/25	string slicing
	09/28	lists
5	10/02	bits, binary, and beyond
	10/05	algorithmic complexity: linear v. binary search
6	10/09	sorting, more algorithmic complexity
	10/13	lists of lists (2D arrays)
7	10/16	*no class — October Study Days*
	10/19	files, I/O, streams
8	10/23	randomness v. unpredictability
	10/26	*first examination*
9	10/30	unordered sets
	11/02	dictionaries
10	11/06	more fun with dictionaries
	11/09	*tba*
11	11/13	event-driven programming
	11/16	introduction to concurrent programming
12	11/20	*tba*
	11/23	*no class — Thanksgiving*
13	11/27	introduction to object-oriented (OO) programming
	11/30	modules, software design
14	12/04	OO continued: inheritance
	12/07	recursion
15	12/11	recursion, continued
	12/14	*second examination*

Week 1 **What is computer science?**
abstraction, abstraction, abstraction
algorithms, pseudocode and programs
primitive, conditional, iterative operations
desirable properties of algorithms
Python v. other languages
Interactive Development Environments (IDEs) — Python's IDLE
numbers and basic arithmetic operations
syntax v. semantics; expressions v. statements
why are introductory algorithms so often numerical in nature?
text (strings), rudimentary input/output (I/O)
what's in a name? variables, assignments, symbolic abstraction
functions and functional abstraction
fruitful functions (return statement).

Lab 0 **Getting started**
programming with blocks
simple arithmetic calculations in IDLE
using variables to compactly express long strings
introduction to turtle graphics.

Program 0 **Warm up**.

Week 2 **Computers $\geq$ calculators**
repetition via definite (for) loops
simple conditional statements
multi-way decisions
formal parameters, actual arguments
function composition.

Lab 1 **Fruitful functions**.

Program 1 **Fruitful functions**.

Week 3 **With power (and repetition) comes responsibility**
indefinite (while) loops
basics of Boolean logic
truth tables
some fun with strings
ReadEvalPrintLoops (REPLs)
some very rudimentary games.

Lab 2 **REPLs**.

Week 4 **Sequences and collections**
strings as sequences of characters
string slicing
mutable v. immutable data
collections in general, lists in specific
lists and arrays.

Lab 3 **Looping and slicing**.

Program 2 **text-based REPL game**.

Week 5 **Everything is bits**
positional number systems
bits, bytes, binary
alternate bases (hexadecimal)
strings as sequences of numbers
very basics of multimedia representation
desirable properties of algorithms, revisited
what is computational complexity?
linear v. binary search
Siff's Law of Logarithms.

Lab 4 **Lists**.

Week 6 **Sorting and searching**
algorithmic complexity, continued
linear v. quadratic behavior
sorting and searching
lists of lists (2D arrays).

Lab 5 **More fun with lists**.

Program 3 **Code making and breaking**.

Week 7 **Operating on larger data**
(*October Study Days*)
files, I/O, and streams.

Lab 6 **Representing grids with turtle graphics**.

Week 8 **That's so random!**
first problem set due
randomness v. pseudorandomness
deterministic v. nondeterministic computation
randomness v. unpredictability
our first exam.

Week 9 **Unordered collections**
faster-than-binary search
sets and dictionaries.

Lab 7 **Randomness**.

Program 4 **Conway's Game of Life**.

Week 10 **More fun with dictionaries**.

Lab 8 **Sets and dictionaries**.

Week 11	**Event handling** sequential processing v. thinking out of order basics of event-driven programming timers introduction to concurrent programming.
Lab 9	**Event-driven programming**.
Program 5	**Randomly generated poetry**.
Week 12	**Events and games** (*no lab*) (*Thanksgiving*).
Week 13	**Objects of our affection** objects and classes introduction to object-oriented (OO) programming modules, software design.
Lab A	**Objects and classes**.
Program 6	**Game project, part I**.
Week 14	**Objects and beyond** OO continued: inheritance recursion and self-reference.
Lab B	**Inheritance**.
Program 7	**Game project, part II**.
Week 15	**Recursion is recursion** *second problem set due* recursion, continued *our second exam.*