

# syllabus for computing for poets comp 131

Instructor: **Mark LeBlanc** SC-1322 508.286.3970  
mleblanc@wheatoncollege.edu Hours: W 2-3:30, F 10:30-12  
<http://cs.wheatoncollege.edu/mleblanc> or by appt

Meeting Times: Mon-Wed 12:30-1:50pm, Room 1349, Science Center

The use of computers to manage the storage and retrieval of written texts creates new opportunities for scholars of ancient and other written works. Recent advances in computer

software, hypertext, and database methodologies have made it possible to ask novel questions about a poem, a story, a trilogy, or an entire corpus. This course exposes you to leading markup languages (XHTML, CSS, XML) and teaches computer programming as a vehicle to

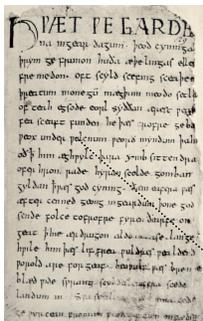


<poem>Three Rings for the Elven-kings under the sky,  
Seven for the Dwarf-lords in their halls of stone,  
Nine for Mortal Men doomed to die,  
One for the Dark Lord on his dark throne  
In the Land of Mordor where the Shadows lie.  
One Ring to rule them all, One Ring to find them,  
One Ring to bring them all  
and in the darkness bind them

In the Land of Mordor where the Shadows lie.</poem>  
The Fellowship of the Ring, JRRT

explore and “data mine” digitized texts. Programming facilitates top-down thinking and practice with computational thinking skills such as problem decomposition, algorithmic thinking, and experimental design. Programming on and with texts introduces students to rich new areas of scholarship including stylometry and authorship attribution. Prerequisites: A love of the written (and digital) word; no previous computer programming experience is required.

Using computers to analyze digitized texts is an exciting new area of research; it enables new forms of both a “close reading” of a single text and a “distance reading” of many texts. Although many tools exist for working *with* texts, what do you do when the tools you have



at your disposal cannot answer *your* questions? In this course, you will learn to write computer programs (also called “software” or “scripts”) that can answer your original questions.

The programming language we will learn is named Python – a modern, accessible, yet powerful language when computing with texts.



```
array = line.split()
for word in array:
    # CASE 1: working on poetry
    if word in dictionary:
        counts[word] = counts[word]+1
```



Alfred), and there are various problems with linking these names (much less their biographies) with more than a very few specific poems. Although a few prose texts are by known authors (particularly the homilists Ælfric and Wulfstan), even the majority of the prose is anonymous. Thus for years scholars of Old English have struggled to divine relationships between texts based on vocabulary, meter, and style. These results have been at best contentious and at worst completely unsuccessful. As we proceed to learn more and more scripting in this course, we will set up and run experiments using the entire Anglo-Saxon corpus. Some of the questions you could ask may never have been asked before. For J.R.R. Tolkien, we will analyze word patterns and design markup schemas that help define structural elements across texts, e.g., the Lord of the Rings trilogy.

---

We will learn to combine Python programs, comma-separated output from those programs, and Excel spreadsheets in a manner very similar to the way your instructors do when in their Lexomics Research Group (see <http://lexomics.wheatoncollege.edu>). **A major “take home story” for this course is to learn ways to view and analyze texts in an entirely new way.** By leaning on computing, new methods of analysis (that you could *not* do by hand) will now be at your disposal.

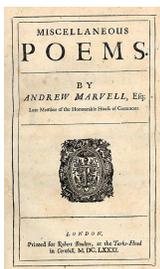
In addition to programming in Python and working in Excel, XHTML and XML, we will also study how computers store individual characters, including the traditional (English-only) ASCII character code and the international standard called UniCode.

At this point, you may be asking: “Why Python (as a choice of programming language)?” Good question (and it is one your instructor has considered at length). Python is an excellent choice for your first programming language. The interpreter environment encourages you to experiment. Note however, that Python is an industrial-strength programming environment. It is used increasingly in industry in a broad range of areas including but not limited to web sites, data and text mining, and scientific computing.

NOTE: This course is but an introduction to using computing to explore digitized texts. Computers allow us to study texts in exciting new ways that we could not otherwise do; however, as we'll discuss at length, **we are wise if we keep in mind what computers cannot do.** The following quotes can help us (1) stay humble and (2) stay focused.

*“As students of a powerful new form of scholarship, we have much to offer.  
We do ourselves no justice when we forget that the quantifiable features  
we deal in are but the shadow of a shadow.”*

John Burrows, *Computers and the Humanities*, v37, 2003, p30.



*“The onus of competency, clarity, and completeness is on the practitioner.  
The researcher must document and make clear every step of the way.  
No smoke and mirrors, no hocus-pocus, no ‘trust me on this.’ ”*

Joseph Rudman, *Computers and the Humanities*, v31, 1998, p353.

*In computer science, if you are almost correct you are a liability.*

Fred Kollett (1941-1997), *MathCS*, Wheaton College, Norton, MA

## Digital Humanities

Computing and the Humanities, eh? If you are reading this syllabus, I'll wager you are beyond the "yeah right" stage of thinking when hearing of the connection. Truth is, the Humanities as a discipline is meeting the challenge of a world of digitized texts and scholarship with a vengeance. You don't have to look too hard to find outstanding examples of the range of work in the digital humanities (e.g., check out: <http://digitalhumanitiesnow.org/>). Scholars from around the globe are marking-up, morphing, mashing, mapping, and mining. So truth in advertising: this course cannot do justice to all of the great work in the digital humanities.

Rather, as I hope you are gleaning from the syllabus at this point, **we will focus on an introduction to "text mining", in particular learning to write programs to explore digitized texts**. So, if you were hoping to learn the details of the Text Encoding Initiative (TEI) or how to mash Graphical Information System (GIS) data with Asian author birthplaces to make an interactive map, sorry ... we'll barely get an introduction to TEI and we won't touch mapping. To further explain the "take home" story for this particular course, read on about computational thinking.

## 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)<sup>1</sup>, 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:

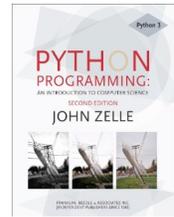
- a move away from "literacy" and toward "fluency," a broader concept including contemporary skills and intellectual capacities
- using abstraction and decomposition when attacking a large complex task
- choosing an appropriate representation for a problem
- modeling relevant aspects of a problem to 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 ... in our case "search texts!!"
- 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.

---

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

**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 a 3-ring binder. I'll pass out lots of handouts.

We'll be using lots of online websites as reference, including those from Scott Kleinman, Associate Professor of [English](#) at [California State University, Northridge](#). Scott is working with the Lexomics Research Group on some research.

He works in Old and Middle English literature (<http://www.csun.edu/~sk36711/WWW/>).

**Your Grade:**

Things to do	Grading Percents	Due Dates
Labs (lecture and lab are “blurred”)	<b>5% overall</b>	in class as needed
6 Assignments	<b>55% overall</b>	
a1: Set up your website	5%	Fri., Feb. 03
a2: JRRT Madlibs	10%	Mon., Feb 20
a3: Regex Play	10%	Mon., Feb 27
a4: “tall” ... “elf” in Tolkien	10%	Tues., Mar 06
a5: Anglo-Saxon poetry	10%	TBA
a6: Marking-up JRRT	10%	TBA
Quizzes	<b>20% overall</b>	
Quiz I	5%	Wed, Feb 08
Quiz II	5%	Wed., Feb 29
Quiz III	5%	Wed., Mar 28
Quiz IV	5%	Wed., Apr. 18
<i>No Final Exam</i>		
Final Project: Text Mining Experiment	<b>20% overall</b>	
Presentation	5%	last week of class
Paper: Methods, Results, Discussion	15%	Wed., May 02

**Late Submissions:**

Due is due. Always turn in whatever you have on time. Something turned in on time 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: **Python Programs** are due on various dates (see detailed syllabus in moodle (onCourse)); however, since I know from experience that many students like to use the last night for testing, I will allow you to submit your programs until 5am the following day. For example, assignment a2 is due Mon., Feb. 20<sup>th</sup> (see above), but you can submit it electronically until 5am Tues, Feb. 21<sup>st</sup> – 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!

**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 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 ever “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 your program 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.

**Tips for working on your own ....**

- (0) It is expected that you spend at least 2+ hours on reading, study and preparation for every 50 minutes of lecture and discussion.
- (1) It is expected that you spend at least 6-10 hours per week on your current programming assignment. **WARNING:** Programmers typically underestimate the time it takes to complete a software project; 6-10 hours per week on your programming assignment may be one of those “underestimations.”

**In classroom “LABS”**

- (0) The computer work in class (labs) are a critical part of the course. Appropriately so, it may be hard to distinguish if you are in lecture or “lab”. In a way, hands-on class time is your time to “hack”, solve unique problems, and show that you can focus on the problem at hand. Typically, your lab time will prepare you to work on your next programming assignment. You must be in class to get credit for the session. If you happen to miss a class, you are strongly encouraged to complete the in-class work on your own time, but please do not ask for credit.
- (1) In order to best grasp the material presented in lab, 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**

Your four quizzes will test your comprehension of Python scripts as well as your ability to write your own Python, regular expressions, and XHTML. During lecture, I will give “hints” of what a typical test question might be; take good notes! There will be no make-ups, nor will the lowest quiz be dropped. If you are an athlete and/or you have a conflict with a quiz date, please see me within the first week of classes.

**HELP**

I have listed my office hours on the syllabus at the top. But I'm usually near a keyboard so we can schedule alternate times to meet. Study, study, study and talk about it with me and others 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...*

Week	<b>Topic (more details and links via moodle (onCourse) )</b> Note: At the start of each lecture, I will write on the board the exact readings (pages and/or URLs) that are associated with the material covered in this class meeting.
1	<b>Jan 25</b> Intro to the web, (X)HTML, and our Linux Apache server (Dreamweaver, WinSCP, and W3C validation)
2	<b>Jan 30 / Feb 01</b> more X(HTML), CSS, and JavaScript working in hexadecimal
3	<b>Feb 06/08</b> Python Integrated Development Environment (IDE): Wing 101 Intro to Python: working with the string API Python output as XHTML
4	<b>Feb 13/15</b> Reading files; handling a corpus of files with glob Reading data from URLs Iterating over lists
5	<b>Feb 20/22</b> Introduction to Regular Expressions (“regex”) Searching for patterns in MS Word; practicing regex with the RegexpPlay .cgi <a href="http://cs.wheatoncollege.edu/regexpplay">http://cs.wheatoncollege.edu/regexpplay</a> Working with conditional expressions <code>if: else:</code>
6	<b>Feb 27/29</b> (hmmm, must be a leap year, eh?) Working with repetitional control: “loops”; more with lists Output to Excel
7	<b>Mar 05/07</b> Review of results from the “tall” ... “elf” experiment
8	<b>Spring Break – Monday, March 12 – Friday, March 16 – Spring Break</b>
9	<b>Mar 19/21</b> Generating word frequencies: Introduction to Python “dictionaries” (hash tables) Intro to XML (XML Copy Editor)
10	<b>Mar 26/28</b> Intro to the Text Encoding Initiative (TEI) Guest lecture by Patrick Rashleigh, Faculty Technology Liaison (“web geek”)
11	<b>Apr 02/04</b> Marking-up JRRT Intro to XPath (via ElementTree) in Python
12	<b>Apr 09/11</b> Text mining Anglo-Saxon poems (including Scott Kleinman’s lemmatized versions)
13	<b>Apr 16/18</b> Introduction to stylistics and authorship attribution, e.g., John Burrows’ DELTA method
14	<b>Apr 23/25</b> Working on Final Projects
15	<b>Apr 30/May 02</b> Final Project Oral Presentations Wrap-up - Evaluations