syllabus for

computing for poets

comp 131

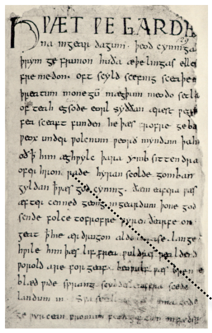
Instructors: **Mark LeBlanc** SC 1322 508.286.3970
mleblanc@wheatoncollege.edu Office Hours: **TBD** or *appt*
Josh Stenger MEN 216 508.286.5436
stenger_josh@wheatoncollege.edu Office Hours: **TBD** or *appt*

Meeting Times: Tue-Thur 2:00 – 3:20pm, Room SC 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 (HTML, CSS, XML) and teaches computer programming as a vehicle to 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.

<poem>It was many and many a year ago,
In a kingdom by the sea,
That a maiden there lived whom you may know
By the name of ANNABEL LEE;--
And this maiden she lived with no other thought
Than to love and be loved by me.
She was a child and I was a child,
In this kingdom by the sea,
But we loved with a love that was more than love--
I and my Annabel Lee--
With a love that the winged seraphs of heaven
Coveted her and me. **</poem>**
Annabel Lee by Edgar Allan Poe (1849)

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 disposable cannot answer *your* questions or the tool(s) expect data in another format? In this course, you will learn to write computer programs (also called "software" or "scripts") that can morph your data and answer your original questions.

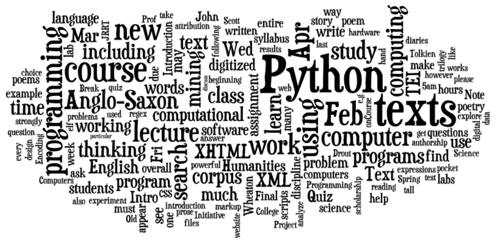


The programming language we will learn is named Python – a modern, accessible, yet powerful language when computing with and on texts.

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

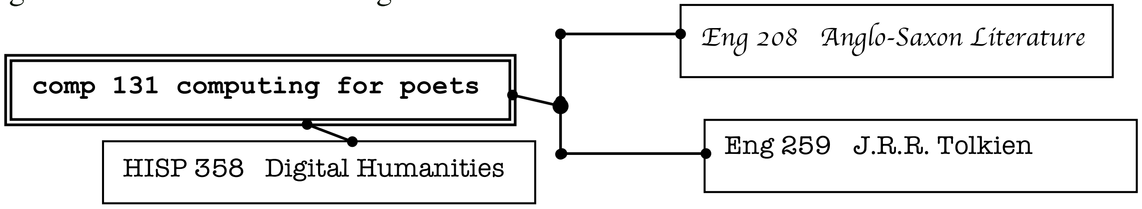
Some of the assignments and labs that you will work on this semester will analyze texts to:

- gain a beginning exposure to HTML, CSS, and JavaScript
- study the beginning steps in authorship attribution as you keep statistics for the relative frequencies of the most commonly used words in a poem, story, or corpus
- design and implement a text mining experiment on a corpus of your choice
  and most significantly …
- write Python scripts to mix/mash/morph data and text files in order to perform text mining experiments that produce results in Excel-ready or HTML files; your programs (scripts) could perform the following:
  - ✓ consider the +/- use of Ngrams with Google's Ngram Viewer; learn to export Ngrams from articles in JSTOR's 50+ million pages of academic journals;
  - ✓ compute the percentage of vowels in a text;
  - ✓ "break" Emily Dickenson poems to facilitate reading poems backwards
  - ✓ search for patterns of letters or words using the powerful pattern matching language of regular expressions ("regex"); for example, how many six-letter palindromes can you find?
  - ✓ search Tolkien's Lord of the Rings trilogy to test the conjecture that the author tends to use words like "tall" near elf names; (*do you think he does?)*
  - ✓ search your own papers that you have written in the past for questionable writing style; for example, what are your most used, top-100 words?
  - ✓ determine the top-10 most frequently used words in the Anglo-Saxon corpus;
  - ✓ while text mining the entire Anglo-Saxon corpus, find the words that appear in poetry that never appear in the prose; *(do you think there are any?)*
  - ✓ find *hapax legomena* (words that appear only once in an entire collection of works), for example, in fifty Shakespeare poems.



CONNECTION

This course is "connected" with three courses: First, two courses from Professor Mike Drout in English: J.R.R. Tolkien and Anglo-Saxon Literature.


Eng 208 Anglo-Saxon Literature
comp 131 computing for poets
Eng 259 J.R.R. Tolkien
HISP 358 Digital Humanities

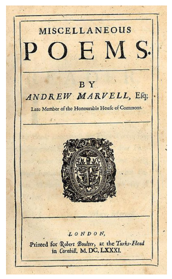
And this course is connected to Professor Domingo Ledezma's HISP 358 "Digital Humanities Methods and Tools".

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, HTML 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. We recently used Python in our Artificial Intelligence (AI) course.

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.


MISCELLANEOUS
POEMS.
BY
ANDREW MARVELL, Eſq;
Late Member of the Honourable Houſe of Commons.

LONDON,
Printed for *Robert Boulter*, at the *Turks-Head*
in *Cornhill*. M.DC.LXXXI.

"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 course and/or connection. The 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. 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 just won't touch those topics. 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)[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 … 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.

Online (free) Text:
Interactive Python: How to Think Like a Computer Scientist
`http://interactivepython.org` by Brad Miller and David Ranum

How to Think Like a Computer Scientist
python™

We also *strongly recommend* that you buy a 3-ring binder. We'll pass out lots of handouts.

We'll be using lots of online websites for both practice and reference, including codeAcademy.com (HTML/CSS and Python) and 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://scottkleinman.net/`).

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

Your Grade:

Things to do	Grading Percents	Due Dates
Labs (lecture and lab are "blurred")	**5% overall**	in class as needed
completion of online materials	**10% overall**	TBA
4 or 5 **Assignments**	**45% overall**	
a1: Set up a website -- Ngrams	5%	Wed., Feb. 7
a2: RPB: Reading Poetry Backwards	10%	TBA
a3: Regex Play	10%	TBA
a4: TBD	10%	TBA
a5: TBD	10%	TBA
Quizzes	**20% overall**	
Quiz I	5%	TBA
Quiz II	5%	TBA
Quiz III	5%	TBA
Quiz IV	5%	TBA
No Final Exam		
Final Project: Text Mining Experiment	**20% overall**	
Presentation	5%	last week of class
Paper: Methods, Results, Discussion	15%	Thurs., May 3

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: **Website** and **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 assignments until 4am the following day. For example, assignment a1 is due Wed., Feb. 7th (see above), but you can submit it electronically until 4am Thur., Feb. 8th – but be careful! The course website (onCourse) makes it appear as if the program is due on Thursday, but remember, that means Thursday at 4am!

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 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 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.

MOOCs (Massive Open Online Courses): We won't specifically be using a MOOC in this class, however, 10% of your grade will be based on you completing online materials at `codeAcademy.com`. To earn these points, you must *show me that you have* completed the "Badges" for each section assigned. If you complete the assigned sections that will be graded as "above average (B)." If you do more sections than assigned, this will be considered Superior (A) effort. If you have already completed these sections, then on your honor, find additional online materials that you can study (just let us know).

Tips for working on your own ....

(0) It is expected that you spend at least 3+ hours on reading, study and preparation for every 100 minutes of lecture and lab.

(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) You will need your laptop almost every day in class.

(1) The computer work in class (labs) is 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.

(2) 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).

Communicating your Results

Learning to share results in a professional manner is a significant "take-home" from this course. This includes your writing (including appropriate, non-ambiguous wording and professional formatting of Tables and Figures) as well as your oral presentations. Post-Wheaton, it will be assumed that you know how to do this well.

Final Projects

As you can see, the final projects are a significant part of your final grade (20% total: 5% for your final talk, 15% for your final report on your experiments). You will work with at least one other person on a team. When working on a team, the collective team will be given a certain number of points and those points will be, by unless directed by team members otherwise (see below), divided and assigned equally. The team will be graded together in this manner twice, once for the oral presentation and then for the final report. For example, if a team is awarded 170 points for their presentation, each person will earn a grade of 85. If that team is awarded 158 for their final report, each person will earn a grade of 79. Note: if a team agrees that one member should be assigned more of the points, it will be up to all members of the team to let us know how to split the grade. Again, unless we hear from all members of the team, the default grading system will be to equally divide the grades.

Quizzes

Your four quizzes will test your comprehension and ability to write your own Python, regular expressions, and HTML. During lecture, we will give "hints" of what a typical quiz question might be; take good notes! There will be no make-ups, nor will the lowest quiz be dropped. If you have a conflict with a quiz date, please see me asap!

Please don't wait too long before you see us; a quick chat in our office can often clear things up. We are here a lot...