Introduction to Programming

Matthew X. Curinga

[**Computer Science 0145-602, Fall 2013**](https://moodle.adelphi.edu/course/view.php?id=63723)

**Keywords:** computer programming, CS1, python, computational thinking, critical computational literacy, Python

**Description:** This course introduces students to programming and some core concepts of computer science, using a modern, object oriented programming language (currently Python). Students learn concepts of variables, functions, repetition/loops, basic data structures (arrays, lists, hashtables), and basic object oriented programming.

**Class meetings:** Tues. 6:30-8:20PM, Harvey 104

**Instructor**

* [Matthew X. Curinga](http://matt.curinga.com), [mcuringa@adelphi.edu](mailto:mcuringa@adelphi.edu)
* [Post Annex, Room 1](http://goo.gl/maps/XReYB)

**Dr. Curinga’s Office Hours**

* Tuesday, 4:30-6:30PM
* Thursday, 3-5PM
* Online or in person, by appointment

**Teaching assistant**

* Hannah Groves, [hannahgroves@mail.adelphi.edu](mailto:hannahgroves@mail.adelphi.edu)

**Ms. Groves’ Lab Hours**

* Thursday, 6pm-8:30pm, Harvey 104 and/or Google Hangout

# Learning Goals

* understand the types of problems that can be solved using computational techniques
* understand the basic concepts of computation (CPU, RAM, permanent storage, GUIs, file systems, network connections)
* learn core computer programming concepts (abstraction, variables, conditions, functions, repetition, recursion)
* think algorithmically to design and test computer programs
* master the basic syntax and idioms of the Python programming language
* use technical documentation, APIs, and the internet to learn new technical concepts
* develop step-by-step problem solving and debugging practices

# Required Text

*None.*

# Readings & Bibliography

*This is a selected bibliography of computer science and Python texts and other materials that you may explore as references or further reading. Many weekly readings will come from these materials. Specific readings will be posted on Moodle for each week.*

Alvarado, C., Dodds, Z., Kuenning, G., & Libeskind-Hadas, R. (2013). [*CS for All*](http://www.cs.hmc.edu/csforall/). Claremont, CA: Harvey Mudd College.

Barry, P. (2009). *Head first programming: [a learner’s guide to programming using the Python language]*. Beijing ; Sebastopol, CA: O’Reilly.

Downey, A. (2012). *Think Python*. Sebastopol, CA: O’Reilly. [free](http://www.greenteapress.com/thinkpython/html/index.html) [py v3](http://faculty.stedwards.edu/mikek/python/thinkpython.pdf)

Pilgrim, M. (2009). [*Dive into Python 3*](http://www.diveinto.org/python3/index.html). New York: Apress.

[*The Python Tutorial v.3.3*](http://docs.python.org/3/tutorial/).

Zelle, J. (2010). *Python Programming: An Introduction to Computer Science*. Franklin, Beedle & Associates Inc.

# Class meetings

Introduction to computer programming meets every Tuesday, 4:30-6:20 in Harvey 104. All students are expected to come on time and prepared for class. You may bring your own laptop to class, or use one of the lab computers for your work. If you need source files to work on in class

|  |  |  |  |
| --- | --- | --- | --- |
| Session | Week | Topic | Due |
| 1 | 4-Sep | Critical Computational Thinking | - |
| 2 | 11-Sep | Data, Math, & Conditions | Quiz 1 |
| 3 | 18-Sep | Strings | Quiz 2 |
| 4 | 25-Sep | Functions | Quiz 3 |
| 5 | 2-Oct | Lists and loops | Quiz 4 |
| 6 | 9-Oct | Testing & Exceptions | **Project 1** |
| 7 | 16-Oct | Dictionaries, Sets, and Tuples | Quiz 5 |
| 8 | 23-Oct | Content Analysis | Quiz 6 |
| 9 | 30-Oct | Advanced Functions | Quiz 7 |
| 10 | 6-Nov | Sorting | Quiz 8 |
| 11 | 13-Nov | Libraries and modules | **Project 2** |
| 12 | 20-Nov | Graphics with SVG | Quiz 9 |
| - | 27-Nov | *No Class. Happy Thanksgiving* | - |
| 13 | 4-Dec | Custom datatypes with Classes | Quiz 10 |
| 14 | 11-Dec | Review/Project Lab | - |
| 15 | 18-Dec | Final Project Presentation | **Final Project** |

# Programming Lab

Every week the teaching assistant will host a programming lab/workshop. It is highly recommended that you attend every lab session. The TA will lead you through sample problems very similar to what will be on the quizzes, will help debug/troubleshoot your code, and will

# Assignments and Grading

|  |  |
| --- | --- |
| Assignment | Pct |
| Discussion Leader | 10% |
| Quizzes | 20% |
| Project 1 | 20% |
| Project 2 | 20% |
| Project 3 | 30% |

## Computational Thinking Discussion Leader

Each student will take a turn leading a discussion about an area of topical interest related to computational thinking and computational literacy. As the leader, you may, optionally, send out a short reading to the class on the discussion as background reading.

When you lead the discussion, be prepared to:

1. introduce the topic with relevant information
2. describe why you think it is an important issue
3. discuss how it is related to computational literacy
4. facilitate a 10-15 minute discussion with the class

This assignment can be completed individually or in teams of 2.

Here are a few potential topics, to give you a sense of themes for this assignment:

* changes to friendship, love, and other social relationships in the age of Facebook and other social media
* use of data encryption
* the split-attention and distraction caused by text messages, twitter, etc.
* role of universities and faculty/professors given a (possible) abundance of online learning opportunities like MOOCs, YouTube, Khan Academy, …
* Wikipedia, knowledge, learning, etc.
* video games and their social effects
* the balance between security and privacy

## Quizzes

There will be 10 pass/fail quizzes which will be completed individually at the start of class. Quizzes consist of 3-5 short answer programming questions drawn directly from the reading for the week. Quizzes are designed to be passed easily by students who are keeping up with the reading and the concepts in the course, and will help students and the instructor maintain a good pace for the development of the course. *If you fail a quiz, you will be required to attend the programming lab on Thursdays until you pass your next quiz.*

## Programming projects

You will complete three programming projects, of increasing complexity, as the major portions of your course.

### Project 1: Facebook Status

You will write a program to categorize Facebook status posts as either “happy” or “sad”.

**Programming competencies:**

* input
  + use variables as input to the program and functions
* output
  + use print() to display output to the user
  + display output in intuitive and useful ways
* variables
  + use string and int variables to hold data
* math
  + increment and decrement counters
  + find averages
  + make numerical comparisons
* conditions
  + test for equality
  + match strings
* style/readability
  + variable names
  + comments
  + docstrings
  + white space
* functions
  + use functions to organize the program and make it more readable
  + use functions for repeated operations

### Project 2: Text Analysis

For this project you will select a text or a collection of texts and write a program that uses computation to analyze the texts. It is up to you to both select the texts and to decide what type of analysis is “interesting.”

In addition to improving on the programming competencies from project 1, you should demonstrate the following in your code:

#### Marking guide:

There are a total of 20 possible points for this assignment, which will be evaluated on the following criteria:

1. **Question Quality** (*3 points*)  
    Does the question chosen reflect a problem that a computer program is good at analyzing? Is the question interesting or important? Is the answer obvious, or is it worthy of analysis?
   * 3 points: the *question can not be easily answered without the aid of software* because it requires a lot of input data, involves tedious/repetetive tasks which are prone to error, or requires complex calculations. Further, the *question is interesting*—it tells us something that is not already established in research or provides evidence for something that is incompletely or ambiguously understood
   * 2 points: the question meets one of the two criteria for 3 points, but not both
   * 1 point: the question is trivial or obvious after brief human analysis, or does not lend itself to generalization/abstract conclusions
   * 0 points: the question would be more easily analyzed by human rather than computer analysis; trying to write software to answer the problem actually makes it more difficult to get a clear picture of the problem
2. **Coding fundamentals** (*8 points*)  
    The program demonstrates a grasp of the programming concepts covered in this class, including:
   * *variables*
     + data is not “hard coded” and can be easily changed by using variables
     + data is separate from functionality
   * *functions*
     + abstraction through function parameters
     + encapsulation through function parameters and return statements; use “pure functions” with zero-side effects
   * *design*
     + the program is organized through the use of functions
     + functions “scale” is appropriate to the task and discrete: concerns are separated logically, such as one function for gather results and another for displaying results
     + functions are written in a way that they are used several times in the program
     + code is not repeated
   * *data structures*
     + use python built-in data structures appropriately: list, dict, tuple, set
     + use index/slice notation if needed
     + sort data structures

* *Point values:*
  + 6-8 points: creates new functions that perform *new analysis* not available in the content.py program provided by the instructor. The program would easily be able to work on a different data set (i.e. different input text files) to provide good results. Functions are consistent (in that the parameters expected and results returned work well with other functions in the program) and can be combined in different ways.
  + 3-5 points: code is clear and organized, but does not add significant new functions, some code may have unintended side effects, such as modifying list or dictionary data in unexpected ways; other code may take parameters, but not use them. Code cannot easily operate on other data sets.
  + 0-2 points: repetitive tasks are not factored into functions, but exhibit more of a copy-paste approach, functions don’t (always) return the expected results or contain logic errors; code doesn’t run due to syntax errors or runtime errors

1. **Risk Taking** (*4 points*)  
    How “adventurous” is this code? Does the student show that they move beyond the template given to them? Do they incorporate ideas from other projects into this? Is there evidence they read online docs or the course text to learn addition techniques to approach the problem?
   * 4 points: in several places, the program use advanced features such as optional functional parameters, list comprehensions, advanced sorting techniques, or string formatting functions; libraries are imported to improve code performance and clarity; content.py functions are modified and improved by the student
   * 3 points: some core changes are made to content.py, other advanced techniques are evident 1 or 2 times
   * 1-2 points: tentative changes are made to content.py, mostly by adding composite functions that combine existing functionality
   * 0 points: only cosmetic changes are made to the initial content.py example, such as changing the input text files and the target words passed into the neighbors function
2. **Code Style** (*2 points*)  
    Is the program consistently formatted according to Python conventions?
   * 2 points: consistently follows the spirit of the *Think Python* and PEP 8 style guides.
   * 1 point: follows guide most of the time, shows internal consistency for style
   * 0 points: lack of consistency in style makes the program harder to read and (potentially) harder to debug and maintain
3. **Results** (*3 points*)  
    The quality of your results include both the information that your program produces and the way that it is formatted for the user. When your program runs, it should produce some type of report that sheds light on your hypothesis. It doesn’t matter if your hypothesis was correct or not—after your program is runs, the user should have more information to evaluate the hypothesis.
   * 3 points: results give clear support for or against the hypothesis by providing relevant information and are formatted in a way that makes them easy to interpret
   * 2 points: results provide some evidence for or against the hypothesis. they may not be formatted in a way that makes them easy to interpret or they may leave some ambiguity that could have been explored further in the code
   * 1 point: some evidence is provided but it is difficult to interpret either due to formatting or the output achieved
   * 0 points: results do not shed any light on the question posed

#### Deliverables:

You should turn in all files related to your project in a single archive (e.g., .zip, .tar, .7z). Project files *must* contain your **Python program** and the data/text files you’re analyzing. The program must start with a comment where you identify:

* the question you are investigating (i.e. your hypothesis)
* the method for investigating this question
* the results identified by your code

To create an archive file, Mac users can simply right-click the project folder from the finder and choose “Compress”. This will create a .zip archive of the project directory. Windows does not come with a compression utility by default. If you do not have one installed or are not sure, Adelphi IT recommends [7-zip, which you can download and use for free.](http://7-zip.org/)

Here is an example header comment:

# news.py  
# by: matt curinga  
"""  
Background:  
New York has 3 major daily newspapers:  
The New York Times, The Daily News, and The New York Post.  
  
It is commonly understood that the Times is written at  
higher level than the other dailies.  
  
Hypothesis:  
Computational analysis of articles will show that  
the Times is written at a higher level than the  
other two papers.  
  
Method:  
This program looks at four measures to compare  
the papers:  
  
1. Average sentence length  
2. Average word length  
3. Word diversity (unique words per 1k words)  
4. Average word frequency (computed against unique word frequency table)  
  
Results:  
The results of this program support the hypothesis in all  
four measures. The Times had longer sentences, longer  
words, greater word diversity, and lower frequency  
words.  
"""  
  
# the code would be here...

#### Text Resources

Here are a few links to places online where you can find full texts to download. This list is by no means exhaustive, but with these links you can certainly find a suitable text for your project. If you are looking for something more specific, or want to work with a text that is not in the public domain or released under a permissive copyright license, please contact the course instructor.

* [Project Gutenberg](http://www.gutenberg.org/) (out of copyright books, other stuff)  
   Project Gutenberg has a very large collection of texts, mostly classic works including novels, poetry, history, philosophy, etc. You can choose to download the texts as a text file (UTF-8 or ASCII). You probably want to delete the Project Gutenberg license and pre-amble stuff before you begin your analysis.
* [Nexis-Lexis](http://libproxy.adelphi.edu:2048/login?url=http://www.lexisnexis.com/hottopics/lnacademic/?)  
   Lexis-Nexis is a database of newspaper (and other news) articles, which affords a range of search parameters. You can export the full text articles for more recent articles (published in the last 20 years or so). You will probably have to download your articles in batches and copy-past them into one file for your analysis. *You must log in to Adelphi follow this link*.
* [JSTOR](http://libproxy.adelphi.edu:2048/login?url=http://www.jstor.org/cgi-bin/jstor/gensearch)  
   JSTOR allows you to search a range of academic journal articles. You cannot easily export the full text of the articles in one shot, but you can export the titles and abstracts, which is often enough for interesting analysis. Like Lexis-Nexis, *you must log in to Adelphi to follow the link*. **Tips:**
* use the basic search, which allows you to export abstracts and titles
* change the options to show 100 results at a time
* use the “select all” toggle to select the full page of results
* choose “Printer-friendly” as the export format
* copy-paste those results into one file
* repeat for next batch of 100 results
* [Library of Congress](http://www.loc.gov/rr/)  
   The Library of Congress maintains a decent online collection of materials, including the text of historical documents and more.
* [WikiSource](https://en.wikisource.org/wiki/Main_Page) WikiSource contains the full text of documents on Wikipedia and elsewhere there is a lot of overlap with Gutenberg, but it might be easier to find and access the WikiSource documents.
* [American Rhetoric Speech Bank](http://americanrhetoric.com/speechbank.htm) I can’t vouch for this source in particular, but they do have a collection of famous speeches. Many/most of these are in the public domain, but this might be a decent place to look for them, if you want to analyze speeches.

#### Sample project ideas

One of the intentional challenges of this project is for students to generate their own ideas for projects. It’s one thing to answer a quiz question or to work on homework problems where you know exactly what you are supposed to do; quite another when you need to decide what the goals for the program are, too, and what results it should find.

Here are some ideas, though, that might get you started.

1. Compare the ways that U.S. news and foreign news cover a topic:
   1. choose your topic
   2. download US news with a keyword search
   3. download foreign news with same keywords
   4. do a neighbor (n-gram) analysis of word counts near key terms
2. [Stylometry](https://en.wikipedia.org/wiki/Stylometry)  
    Instead of considering the context and content of important keywords, stylometry would let us look at the style of a text. By considering common words and phrases, we might be able to answer questions such as:
3. is a text written by a man or a woman?
4. do African Americans have a distinct writing style?
5. how has writing style changed over time? can we identify when a piece was written based on its style?
6. Compare 2 books or 2 authors  
    Create two texts from Gutenberg or another source and pick an area to analyze. Classic things to consider are the ways a text deals with issues of race, gender, sex, violence, humor, death, god/religion, etc.
7. Compare speeches:

* what are the difference and similarities between speeches from WWI, WWII, Vietnam, and today? Has the “security” message changed? Has the anti-war/peace message changed?
* compare the speeches from different political parties? what propaganda do they use? how do they use language to shade the issues that are most crucial to their supporters?
* what are the characteristics of a great speech? are the similarities between the rhetorical styles of Lincoln, Churchill, King, Malcom X and others?

### Project 3: Refactoring & Visualization

For your final project, you can team up with one of your classmates to revisit and improve your earlier code. In particular, you should make your code more robust, more flexible, and improve your tests. You should explicitly write code that demonstrates your understanding of abstraction, encapsulation, and algorithms. In the final class meeting you will present your project in class, and show the parts of your program that best exemplify these concepts, as you understand them.

In addition to improving and expanding your existing code, you will add a graphical visualization of your results using the a Scalable Vector Graphics (SVG) library. SVGs are graphics files that can be displayed in any modern web browser, and will let you create things like line graphs, bar graphs, labels, heat maps, and other visualizations.

This assignment must be completed in a team of 2. If you wish to work in a team of three, please seek instructor permission.