CSC 242 Course Guide

Last updated: 6/10/2015

Author: Amber Settle, based on materials and comments by Ljubomir Perkovic and Marcus Schaefer

Course Description

An intermediate course in problem solving, algorithms and programming. Programming skills are further strengthened through more complex and larger programming assignments. The assignments will also be used to introduce different Computer Science areas (e.g. a Client/Server application for the Distributed Systems area). Classes and object oriented programming are motivated and introduced. PREREQUISITE: CSC 241

Prerequisites and audience

It is very important that students taking CSC 242 have CSC 241 or an equivalent class. Students who have some programming experience should take CSC 243 instead of CSC 242 (or CSC 241).

In particular, the following text is helpful for the syllabus:

You must have taken CSC 241: Introduction to Computer Science I or an equivalent course that introduces problem-solving techniques and programming Python and earned a passing grade (C- or better). I will also assume that:

- You know how to create, debug, compile, and run Python, and you use a reasonable coding style (i.e. your code is easy to read and relatively concise)
- You know Python's basic control structures and types
- You can solve basic algorithmic problems

The abilities of the students vary significantly quarter to quarter. This impacts the topics that can be covered (see the week-by-week topics section below). Winter quarter students tend to be the strongest and Fall quarter students the weakest.

Learning goals

This course is the second of a two-course sequence introducing computer science skills, including problem solving, algorithm development, recursion, and programming using Python. In this course, we will apply these skills in several application areas of computer science: graphical user interface (GUI) development, database development, and Internet and distributed computing. The concept of a class and object-oriented programming will be motivated and introduced.

The following is helpful for the syllabus:

After you have taken this class:

- You will strengthen your Python programming skills
- You will know how to design classes and understand the fundamental principles of object-oriented programming
- You will be able to design basic graphical user interfaces
- You will be able to apply recursion as a problem-solving and programming technique
- You will be able to write simple Internet client programs
- You will have a basic understanding of the database API

Textbook

Sections of CSC 242 taught during or after Winter 2016:
Introduction to Computer Science with Python, 2nd edition (ebook), Ljubomir Perkovic, Wiley, 2015.

ISBN (ebook): 978-1-118-89105-6

Instruct students to obtain the ebook version of the 2nd edition. The print version no longer has the case studies (some of which are used in the course).

If a student buys a print copy and needs access to the case studies, they are available as an ebook supplement (currently for $12.40) at http://store.vitalsource.com/search?q=9781119185390&search.x=26&search.y=9&search=search. The ISBN for the supplement is 9781119185390.

Sections of CSC 242 taught during or before Fall 2015:
Introduction to Computer Science with Python, Ljubomir Perkovic, Wiley, 2012.

Week-by-Week Syllabus

The course follows the textbook through Chapters 7, 8, 9, 10, 11, and 12 in that order. Chapters 10 and 11 (recursion and web development) take the majority of the time, and Chapter 12 (databases) often gets slighted due to a lack of time.

Week	Topic	Chapter(s)
1	Namespaces and scope; an introduction to object-oriented programming	7 and 8
2	Object-oriented programming	8
3	Object-oriented programming and an introduction to graphical-user interface development and event-driven	8 and 9

	programming	
4	Graphical-user interfaces	9
5	Recursion and the midterm	10
6	A discussion of the midterm; recursion, sorting, and searching	10
7	More about recursion	10
8	An introduction to HTML and WWW application development	11
9	WWW application development	11
10	The database API	12

Depending on the abilities of the students, object-oriented programming may take longer than two weeks. This tends to delay everything and push databases at least partially out of the class.

Students generally find CSC 242 to be several orders of magnitude harder than CSC 241. The concepts covered are more complex, and students need to write significantly more code. It is worthwhile to tell them this. Students who earned a B- or lower in CSC 241 should be ready to devote extra time to the class. It's not unusual for students to drop an entire grade in CSC 242 as compared to their CSC 241 grades.

Particularly troublesome for students are object-oriented programming and recursion. Be ready to devote extra time to these topics. It's more important that they understand those topics than for them to see databases in any significant way.

Labs

Each section of CSC 242 has its own associated lab session. The instructor is responsible for organizing the lab session. The time in the lab will be supervised by a teaching assistant, and the instructor should convey the topics and assessments to the TA at least 2-3 days before the lab. A set of activities reinforcing new material or preparing them for class examples or assignments works well. Typically group work is highly encouraged for the lab, so the students learn from each other and learn to work in groups. It is best if the students submit work individually, acknowledging any collaboration in comments at the top of the file submitted. See Appendix C for a sample lab. Appendix D includes grading criteria developed for the lab sessions.

In the Classroom

Although it is helpful to show rather than tell, often a large amount of material must be covered initially for each chapter. Pre-prepared examples that are shared with the students prior to class can be invaluable in making reasonable progress through the

material. Those examples can be both modified and demonstrated throughout the class session.

It is also crucial that students have an opportunity to develop their own code based on the examples seen in class. For example, students simply do not grasp how communication between methods in a class works until they have written a class and written a program that uses the class. Allowing them practice on small examples during class time before they reach the lab or the assignments will help them to not crash as viciously as they might otherwise.

Warnings about Python and the curriculum

Note that if you give a module a name that coincides with an existing module (queue and random are two examples), then the built-in module will no longer be visible when Python is run in the directory containing that module. For the queue example, this causes Python to crash. Avoid calling any module you write queue.py.

When writing regular expressions, be aware that \b can lead to problems, since it is a special string character. One way to avoid the problem is to escape it by writing : '\\b'. You can also raw strings to find all words in a text as in: re.findall(r"\b(\w+)\b", text).

Be aware that many of the HTML parsing and crawling programs as written in the book will not work on regular web pages for two reasons. First, some pages block automatic down-loading. Some pages also use special characters, which trip up the parser, probably on purpose. Being aware of this when you write your solutions to exercises and assignments and warning students about this is advised.

When discussing the Python database API, be aware that tuples with single items can cause some problems. In particular, if you are using the ? style of pulling data from a database table, as is recommended for security reasons, you need to be careful if there is only one ?, e.g. cur.execute('SELECT * FROM enrolled WHERE courseID = ? AND studentID = ?', (cid,sid)) is fine, but cur.execute('SELECT * FROM enrolled WHERE courseID = ?', (cid)) will give you an error. The problem is that it does not recognize (cid) as a tuple, so the solution is: cur.execute('SELECT * FROM enrolled WHERE courseID = ?', tuple([cid])) or simply use a list: cur.execute('SELECT * FROM enrolled WHERE courseID = ? AND studentID = ?', [cid,sid]) or cur.execute('SELECT * FROM enrolled WHERE courseID = ?', [cid])

Homework

As with CSC 241, you should enforce high standards on homework submissions. For example, a no-late-submissions policy is fairly standard, with the caveat that the lowest score for each type of assessment is dropped.

It may be the case that you need to provide students with some code on assignments as a starting place in order to create an assignment that can be reasonably done in a week.

The students also develop complex, interacting classes and methods, so it's highly recommended that you require them to submit the Python files. Often figuring out what is wrong with their code and how far it is away from a correct solution is something that requires playing with the code. Doing that in your head from a printout can be a challenge.

In Chapter 8 of the text, many of the exercises focus on writing classes in isolation from any other program. The emphasis is on writing the classes and not on the use of those classes. I strongly recommend that you require students to both write classes and write programs that use those classes. Assignments that involve file processing are particularly useful. Students simply don't understand what classes do until they have created a series of objects and used them for some purpose. Appendix B gives an example of an assignment of this type.

Students find recursion to be difficult. They will often go to great lengths to avoid dealing with recursion, especially recursive functions that return values. It is strongly recommended that you disallow certain constructs on recursive assignments to ensure that they using recursion effectively. In particular, tell students that they are not allowed to use global variables, that they may not change the number of parameters of the function, and where appropriate, that loops are not allowed.

Exams

Both midterm and final exams are given in the lab, so the students can solve the problems on the computer. The best setting for the dropbox is one exam submission per student, since otherwise the student can leave the classroom and submit a new exam later. However, it is a good backup plan to have a USB stick ready to collect exams in case of emergency.

Tell techstaff (helpdesk@cdm.depaul.edu) well before the midterm date to turn off the Internet in the classroom with the exception of access to D2L, so students can download the midterm (and any supplementary files) and submit their solutions. Please note that the final exam typically involves writing a web crawler of some sort. The students need to have access to Web pages to test it, so don't turn off Internet access for the final exam.

Plagiarism

Plagiarism is less of an issue in this class than in CSC 241, but it is helpful to remind the students that they need to do independent work. The typical policy is to allow students to collaborate on lab assignments but to require independent work on assignments. Appendix A contains a sample academic integrity statement written by a CSC 241 instructor. It is useful to have students sign it before grading any of their homework.

One exception are the exercises found in the chapter on recursion. Many of the exercises in the book have solutions posted online. You are strongly advised to write your own variations on these exercise to ensure that students aren't copying.

Setting assignment due dates so that students are starting a new assignment during the lab session is advisable. Assignment due dates that have students completing an assignment close to the lab session encourages a lot of collaboration, which can be a problem.

References

The website for the textbook includes lots of useful information, including all the code from the text, a solutions manual, lecture PowerPoint slides, and errata. Use of the PowerPoint slides hasn't been typical for DePaul instructors, since using the IDLE editor and Python files have proven quite effective.

Python:

- Python.org includes:
  - Python 3.4.3 downloads (for Windows, MacOS, and others)
  - Tutorial
  - Reference

Appendix A: Academic Integrity Statement

Academic Integrity Pledge for Course Assessments

I, _(your name here)
_____, pledge that I have read the Academic Integrity policy of DePaul University. The full policy can be found here: http://academicintegrity.depaul.edu/AcademicIntegrityPolicy.pdf and other resources on Academic Integrity can be found here: http://academicintegrity.depaul.edu/ The following is an excerpt from the Academic Integrity policy:

> **Cheating**: Cheating is any action that violates university norms or instructor's guidelines for the preparation and submission of assignments. This includes but is not limited to unauthorized access to examination materials prior to the examination itself, use or possession of unauthorized materials during the examination or quiz; having someone take an examination in one's place-copying from another student; unauthorized assistance to another student; or acceptance of such assistance.
>
> **Plagiarism**: Plagiarism is a major form of academic dishonesty involving the presentation of the work of another as one's own. Plagiarism includes but is not limited to the following:
>
> - The direct copying of any source, such as written and verbal material, computer files, audio disks, video programs or musical scores, whether published or unpublished, in whole or part, without proper acknowledgement that it is someone else's.
> - Copying of any source in whole or part without proper acknowledgement.
> - Submitting as one's own work a report, examination paper, computer file, lab report or other assignment that has been prepared by someone else. This includes research papers purchased from any other person or agency.
> - The paraphrasing of another's work or ideas without proper acknowledgement.
>
> **Complicity**: Complicity is any intentional attempt to facilitate any of the violations described above. This includes but is not limited to allowing another student to copy from a paper or test document; providing any kind of material—including one’s research, data, or writing—to another student if one believes it might be misrepresented to a teacher or university official; providing information about or answers to test questions.

I understand that programming assignments must reflect individual work. I may consult the course notes, course recordings, course tutorials, the course textbook, other textbooks, or online resources as well as discuss the assignments with Dr. Settle, Mr. Summers, or the CDM tutors. However, I may not under any circumstances submit work that is not my own. I understand that I should not discuss the assignments in the course with my classmates. Further, I understand that working so closely with anyone so as to produce identical or near identical code, except for Dr. Settle and Mr. Summers, is a violation of the Academic Integrity policy.

During the scheduled lab for the class I understand I will be working on exercises that may be completed using the course notes, course recordings, course tutorials, the course textbook, other textbooks, or online resources. On the lab exercises I am allowed to consult with Dr. Settle (if available online), Mr. Summers, and other students in the class. If I write any code with another student on a lab exercise, that student's name must be included in the lab exercise submission. Not providing credit to my lab collaborators on exercise submissions is not allowed. If I spend lab time on assignments I understand that I must follow the rules described in the paragraph above, which are different than the rules for the lab exercises.

I understand that the midterm and final exams for this course are closed book. I may use a fixed number of pages of notes for the exams, with the number to be specified in the exam logistics later in the quarter. The sheet(s) of notes must be handed in with my exam. I may also use a calculator during the exam. I may not use any other written materials. The exams are scheduled in the lab, but I may only use the Python interpreter (either through IDLE or using the command-line interface) on the lab machines during the exams. I may not bring my own laptop, or any other electronic device, into the exams. I also may not communicate with anyone other than Dr. Settle during the exams.

I understand that failing to adhere to any of the above stated policies on the assessments for the course is a violation of the Academic Integrity Policy at DePaul University. I understand that if this occurs Dr. Settle will file a violation of the Academic Integrity policy for me and that I will receive a 0 on the associated assessment(s).

Signature: _____

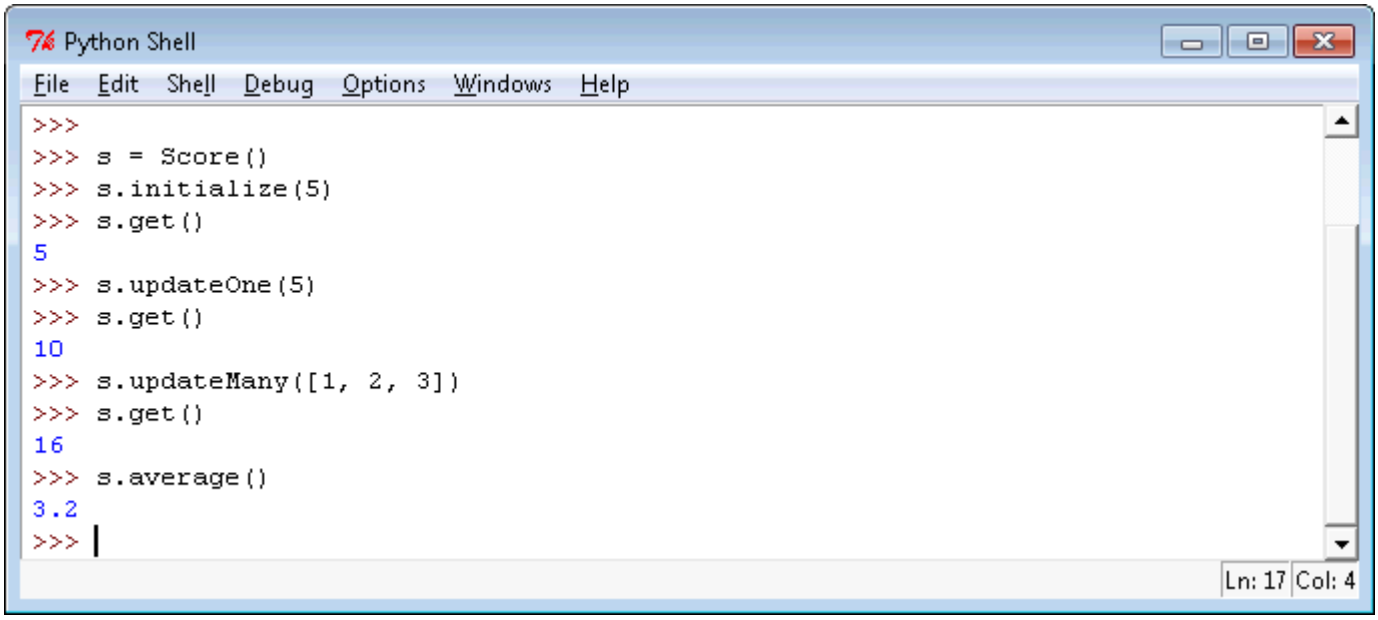
Date: _____

Appendix B: Object-oriented programming assignment

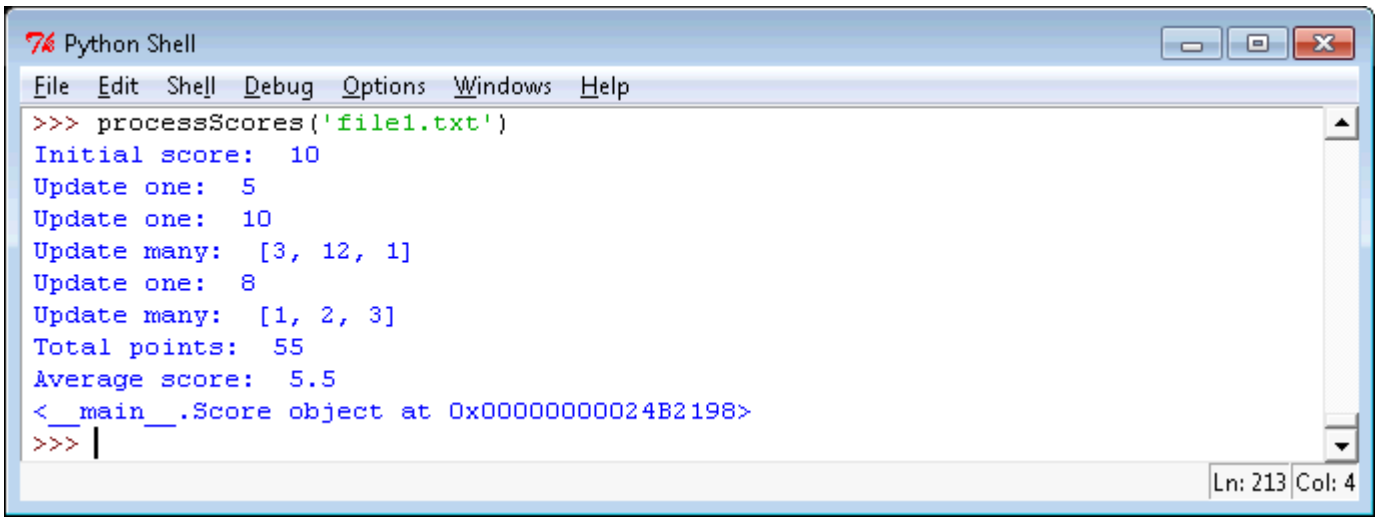
Implement the classes and functions below and save them into a file called csc242hw1.py. I am not providing any template file for this, so you must create it yourself. Please follow the formatting conventions found in the examples discussed in class. **You must also include appropriate doc strings** (e.g. strings that appear on the line following the class or function header) to the class and functions that clearly and concisely describe what the class and functions are doing. A submission without doc strings will not earn full credit.

1. Develop a class **Score** (that is a subclass of the object class) supporting five methods. The class is intended to hold a running score, for example for a game or a class. The methods it supports are:
    a. **initialize**(self, init) which takes a numeric value as a parameter and sets the initial score of the object to the parameter. It also sets the number of scores that have contributed to the total to 1.
    b. **updateOne**(self, amount) which takes a numeric amount as a parameter and increases the total score of the object by that amount. It also increases the number of scores contributing to the total by 1.
    c. **updateMany**(self, lst) which takes a list as a parameter. The list represents a series of scores. The method updates the total to include the sum of all the scores in the list and updates the number of scores contributing to the total by the number of items found in the list.
    d. **get**(self) which takes no parameters and returns the current score in the object.
    e. **average**(self) which takes no parameters and returns the average of the scores that have contributed to the total score.

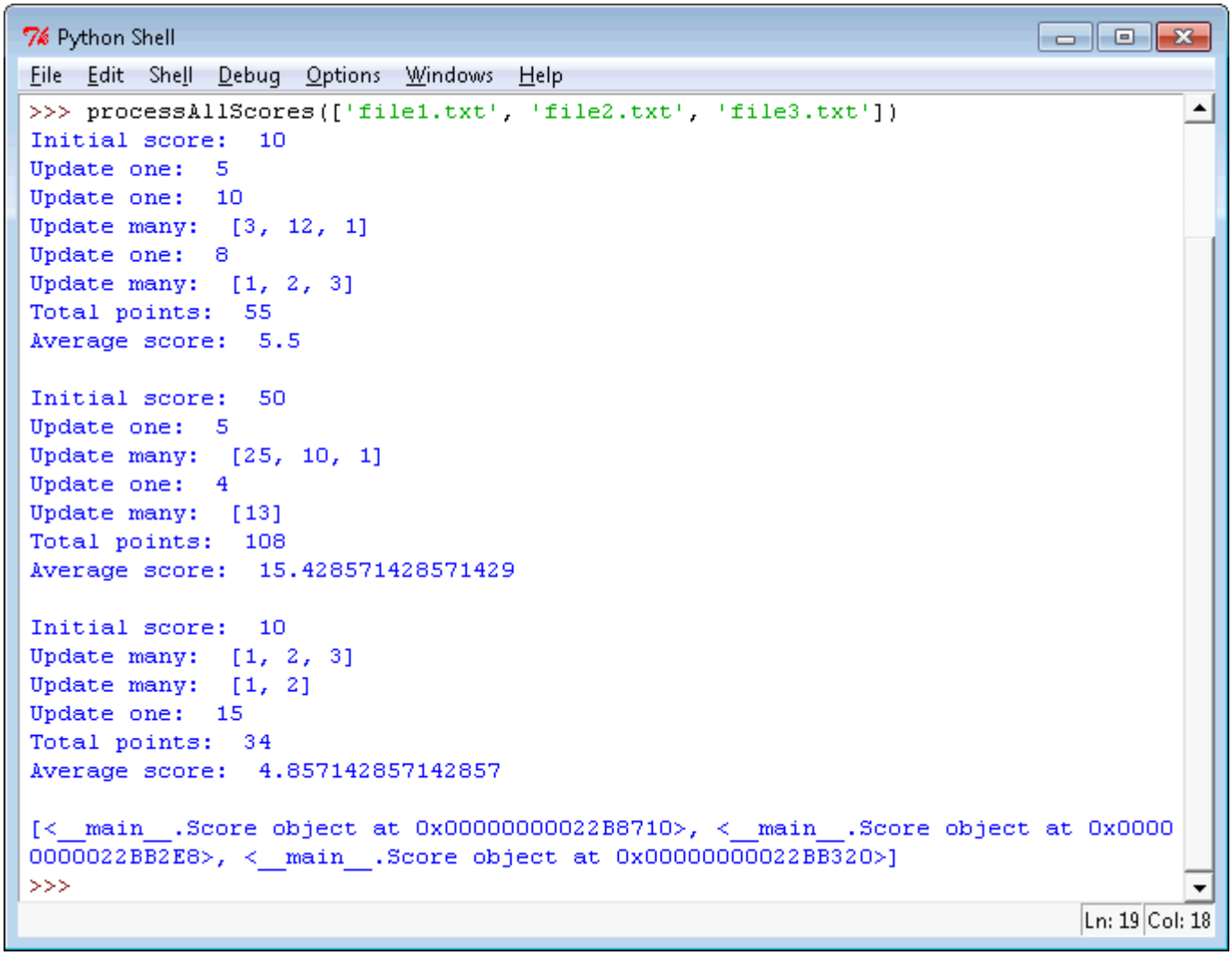
    The following shows how the **Score** class and its methods could be used:



2. Write a function **processScores**() that takes the name of a file as a parameter. The first line of the file is a number, which indicates the initial score for a Score object. The remaining lines are pairs: the even-numbered lines contain a character and the odd-numbered lines contain either a number or a list. The character will be one of 'o', 'O', 'm', or 'M', indicating that the next line contains either a single score ('o' or 'O') or a list of scores ('m' or 'M'). The function will create a new **Score** object and then process each line or pairs of lines of the file by calling the appropriate method of the**Score** object. As it processes each line or pairs of lines, it will print the information about the action being taken. The first line will be a call to the **initialize**() method and the remaining pairs of lines will be calls either to the **updateOne**() or **updateMany**() methods. Once the file has been processed, the function will print the final score, the average score, and return the **Score** object. The following shows what would be displayed for an example file. The example file can be found on the D2L site under the Assignment 1 section on the Content page. Please note that your function must work on an arbitrary file that consists of valid lines. You cannot assume anything about the file except that it contains lines that have the format described above.



3. Write a function **processAllScores**() that takes a list of file names as a parameter. Each of the file names corresponds to a text file that contains a list of actions to be taken on a Score object. The function will process each file in the list by calling **processScores**() function defined in the problem above. The object that **processScores**() returns will be added to a list. The **processAllScores**() function will return the list of objects once all files have been processed. The following shows what would be displayed for an example run with three files. The example files can be found on the D2L site under the Assignment 1 section on the Content page. Please note that your function must work on an arbitrary list. You cannot assume that you will always be given three files.



Submitting the assignment

You must submit the assignment using the assignment 1 dropbox on the D2L site. Submit only a single Python file (csc242hw1.py) with your implementation in it. Submissions after the deadline listed above will be automatically rejected by the system. See the syllabus for the grading policy.

You must also sign and submit an Academic Integrity pledge. The pledges will be made available in class during the first two weeks of the quarter. The Academic Integrity pledge can also be found in the Assignments and Logistics and general information sections on the Content page of the D2L site. By the deadline you must have signed the pledge and either submitted in person or upload it with your csc242hw1.py file to the assignment 1 dropbox.

Appendix C: A Sample Lab

Logistics

These exercises should be completed during the lab on Tuesday, September 11th. In order to receive full credit for the lab, you must attend the session, remain in the lab for at least 45 minutes, and submit a file that contains solutions to all these exercises. A grading rubric for lab sessions is available.

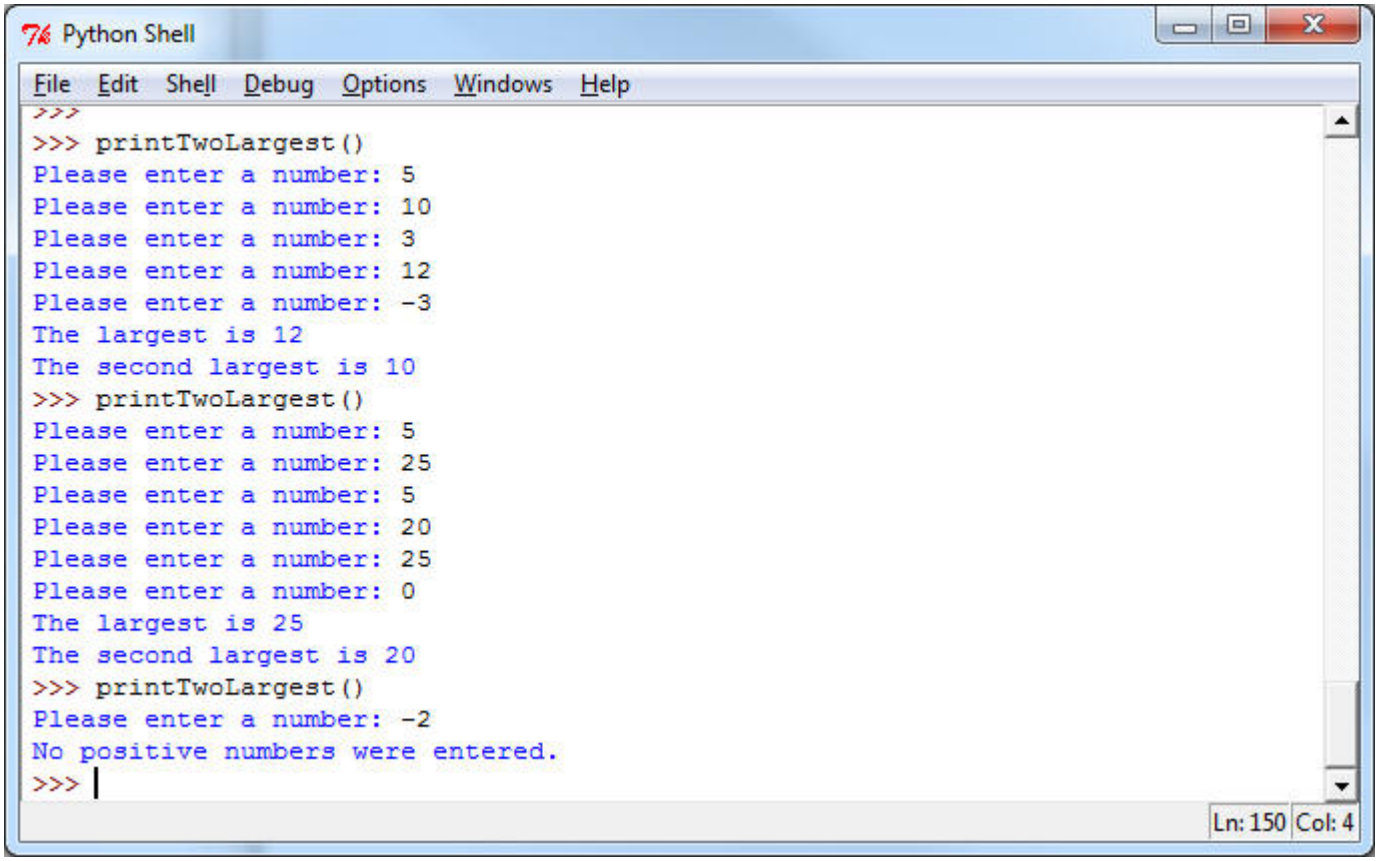
You are encouraged to work in groups on lab exercises. If you do work with someone, please include the name(s) of your collaborator(s) at the top of the file you submit. For more information about collaboration policies in this class, see the Academic Integrity Pledge posted to the D2L site.

If you complete the lab exercise early, please read Chapters 7 and 8 in the textbook and work on the first assignment. You must remain in the lab for at least 45 minutes to earn full credit. Please be aware that the assignments are individual assessments. You are not allowed to discuss the assignments with your classmates. If you need help on the assignment, please ask Andrew Summers.

Review exercise

This exercise is a review of material covered in CSC 241. You should find it to be straightforward and simple. If that's not the case, please be sure to review your CSC 241 materials as a part of your first week of work for this class.

Implement the function **printTwoLargest**() that inputs an arbitrary number of positive numbers from the user. The input of numbers stops when the first negative or zero value is entered by the user. The function then prints the two largest values entered by the user. If no positive numbers are entered a message to that effect is printed instead of printing any numbers. The information below shows how you would call the function **printTwoLargest**() and what it would display for a couple of different sample runs:

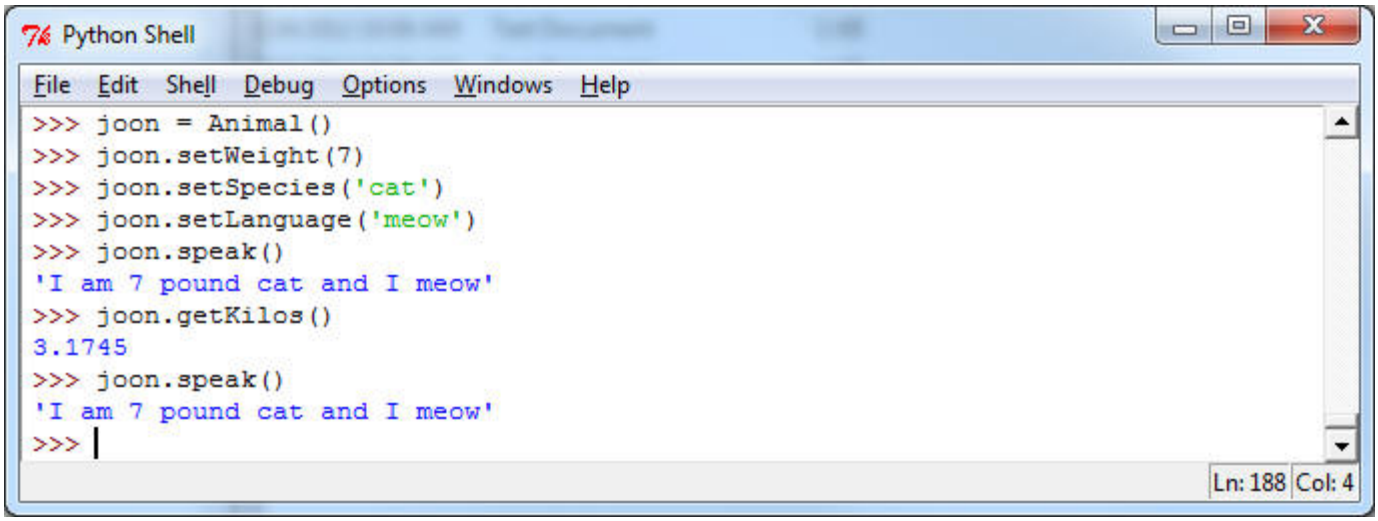


New exercises

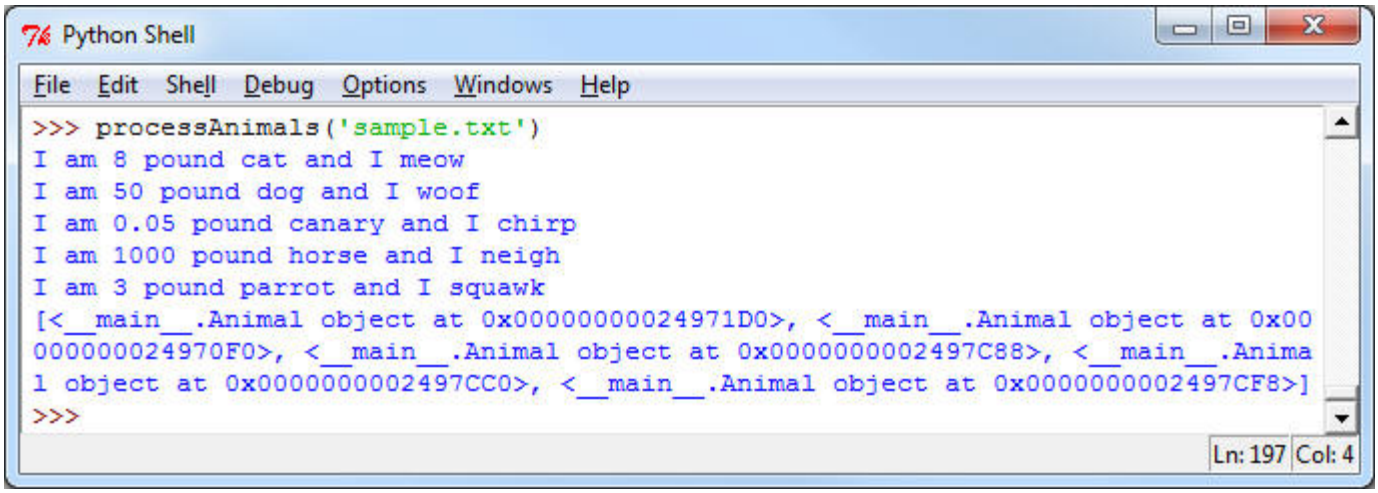
These exercises cover the material we have learned this week in class.

1. Modify the original **Animal** class found in the file csc242lab1.py included in the zip file for this lab to include two additional methods:
    a. **setWeight**() which takes an integer as a parameter and sets the animal's weight (in pounds) to the value of the parameter
    b. **getKilos**() which takes no parameters and returns the weight of the animal **in kilograms**

    You should also modify the **speak**() method to display the animal's weight (in pounds) in addition to the animal's species and language. The following shows how the **Animal** class and its new/modified methods could be used:



2. Implement the function **processAnimals**() that takes as a parameter the name of an input file. The input file contains zero or more lines with the format: species, language, weight where species is a string representing an animal's species, language is a string representing an animal's language, and weight is a number representing an animal's weight in pounds. The items on each line are separated by commas. The **processAnimals**() function should read all lines in the file, creating an Animal object for each line and placing the object into a list. The function also displays to the screen the result of calling speak() on each object in the list it created. The function returns the list of objects created or an empty list if the file didn't contain any lines. Don't forget to close the input file. The information below shows how you would call the function **processAnimals**() on an example file. The file used below is included in the zip file that contains the class lab1.py file provided on the D2L site:



Submitting the exercises

You must submit your solution to the exercises using the lab 1 dropbox on the D2L site. Submit only a single text file (**csc242lab1.py**) with each of the completed functions and classes for the lab exercises in it. Submissions after the deadline listed above will be automatically rejected by the system. See the syllabus for the grading policy.

Appendix D: Lab Grading Criteria

(http://facweb.cdm.depaul.edu/asettle/csc242/info/labRubric.html)

Lab grading rubric

This page describes a grading rubric for the lab sessions associated with the course.

Each lab session is worth 10 points. There are two things that contribute to your grade in the lab: your attendance at the lab session and your submission of solutions to the lab exercises. In order to receive full points for attendance, you must arrive no later than 10:15 am, leave when the exercises are complete or 11:00 am whichever comes later, and work the entire time on the lab exercises or some other activity associated with the course. To receive full points for the lab exercises, you must submit a file containing a solution to all exercises on the lab assignment by the deadline specified in the exercise set. A specific rubric for each area is given below:

Exercise completion	Points earned
Submits a file by the deadline containing a solution for all of the lab exercises	5
Submits a file by the deadline containing a solution to a majority of the lab exercises	4
Submits a file by the deadline containing at least a partial solution to a majority of the lab exercises	3
Submits a file by the deadline containing at least a partial solution to some of the lab exercises	2
Does not submit any solutions to the lab exercises	0

Lab attendance	Points earned
Arrives on time (i.e. no later than 10:15 am), stays for the duration of the lab (i.e. when the exercises are done but no earlier than 11:00 am), and works the entire time either on the lab exercises or on something related to the course, including doing the assignment, reading the textbook or other Python documentation, or studying for exams.	5
Arrives on time (i.e. no later than 10:15 am), stays for the duration of the lab (i.e. when the exercises are done but no earlier than 11:00 am), but does not work the entire time on something related to the course, including doing the assignment, reading the textbook or other Python documentation, or studying for exams.	4
Arrives late (i.e. later than 10:15 am) or does not stay for the duration of the lab (i.e. leaves before the lab exercises are complete and before 45 minutes have passed) but not both, and works the entire time on something related to the course, including doing the assignment, reading the textbook or other Python documentation, or studying for exams.	3
Arrives late (i.e. later than 10:15 am) or does not stay for the duration of the lab (i.e. leaves before the lab exercises are complete and before 45 minutes have passed) but not both, and does not work the entire time on something related to the course, including doing the assignment, reading the textbook or other Python documentation, or studying for exams.	2
Arrives late (i.e. later than 10:15 am) and does not stay for the duration of the lab (i.e. before the lab exercises are complete and before 45 minutes have passed).	1
Does not attend the lab	0