Draft Version: 2/6/2014

HW4: CS 110X C 2014

Note: This homework (and all remaining homework assignments) is a **partner homework** and must be completed by each partner pair. When you complete this assignment, you must not share your answers with any other student. Only one person from a partner pair needs to submit the assignment, but make sure that you submit before the deadline!

For this assignment, every function that you write must have a suitable documentation string as we present in class. Check the rubric to see the point values assigned for each question so you can maximize the points you get on this assignment. Note that the final question is quite complicated (though still within reach) even though it is only worth 8 points. Be sure to only begin this question after you have completed the rest of the homework assignment.

Domain Information

You have now worked with sound data for the past two homework assignments. In this homework you will pull it all together. Sound waves compose together in two ways. For example, assume you have generated two different one-second sound waves

- Wave1 = one second playing the middle C tone (a total of 44,100 samples)
- Wave2 = one second playing the E tone above middle C (a total of 44,100 samples)

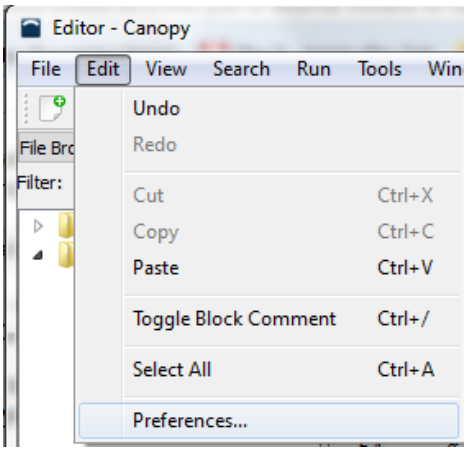
You can **concatenate** sound wave data together (this is described as **Wave1; Wave2**) to create a single wave file containing 88,200 samples. Since you are using a Python list to represent sound wave data, you can use the **+** operator with lists to concatenate the lists together to create a single list containing the wave data. In Python, if you have two lists `x` and `y`, then `x+y` concatenates the lists together.

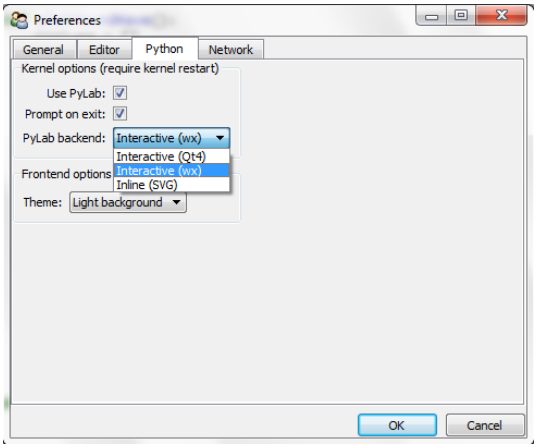
You can **combine** sound wave data together (this is described as **Wave1 • Wave2**) to create a single wave file containing 44,100 samples. When **combining** sound waves you have to be careful to **normalize** the resulting data so all sound wave data remains within the [-1, 1] allowed range. For homework 3 you defined a `combineSoundWave(frequencies, n)` method that you used to create a combined and normalized wave form containing just the given frequencies. For this assignment you will write a separate function that normalizes combined sound wave data.

Canopy Issues

If you are running Canopy then you have to make a small configuration change for this homework to work properly. From within the Canopy Editor, select menu item **Edit | Preferences…**.

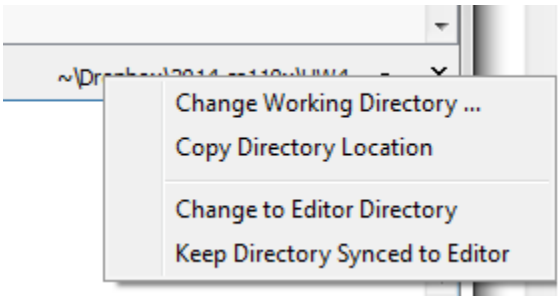
Draft Version: 2/6/2014





Then in the Preferences window, select the **Python** tab and be sure that **PyLab backend** is set to "Interactive (wx)". I have tested this on Windows and on a MacBook.

In Canopy, you need to set the working directory within the code editor. Near the right edge about one inch from the bottom you will see a small triangle that you can click on to change the working directory. Select the "Change to Editor Directory" so you will find the files that you need for this assignment.



Draft Version: 2/6/2014

Homework Themes

This homework will exercise your skills in **string manipulation**. String literals can be printed for users to read, but they can also contain information that can be used to write programs that are easy to use. In some ways, the ability to process strings is what separates mathematicians from computer scientists. Additionally, you will learn how to **read information from and write information to files** that you store on your computer. Often a tedious task can be automated simply by writing programs that process large quantities of information that are stored on disk.

With each passing week, the homework assignments can contain truly interesting problems. The last problem on this homework in particular pulls together material from the past two assignments to give you a glimpse of the limitless power of programming. This question has a number of sub-parts that you should only attempt once the entire homework is done.

Homework Instructions

This Homework has **Eight** questions.

For each question be sure you understand exactly the format of the output that is requested. You will lose points if you do not exactly follow the format of the output for the individual questions. Should you have any questions, be sure to review the HW4 rubric and post questions on the HW4 discussion forum.

Q1	Demonstrate ability to work with strings
Skills CS-1 CS-3 PF-3	Write a function `minuteOfDay(time)` that **returns** an integer representing the number of minutes since midnight for a time string containing five characters in the form "HH:MM". You can assume that the hours HH is a value ≥ 0 and ≤ 23. You can assume that the minutes MM is a value ≥ 0 and ≤ 59. Your function must return an integer value

Sample Output in IDLE	`>>> minuteOfDay('00:18')`
	`18`
	`>>> minuteOfDay('13:37')`
	`817`

Sample Output in Canopy	`In[1]: minuteOfDay('23:59')` `Out[1]: 1439`

Q2	Demonstrate knowledge of for loop, file access, string manipulation

Skills
SM-1
IO-4
CS-9
IO-6

Lecture Dependency
Feb-06

You manage the finances for a restaurant. Each day the workers at the restaurant start their shift at a certain time and work continuously until a later time (which may be after midnight. Be careful!)

Your must compute a report for all workers given a file containing a line of input for each worker. Each line contains 40 characters of input as follows:

```
1234567890123456789012345678901234567890
Alfie Curtis          08:17 17:33  07.95
```

That is, the name is shown in the first 22 characters of each line. The starting time of that worker's shift is contained in characters 23 to 27, the ending time of that worker's shift is contained in characters 29 to 33. Finally, the hourly wage of the worker is contained in characters 36 to 40. Alfie has worked a total of 9 hours and 16 minutes for a wage of $7.95 per hour, so he should receive a daily wage of $73.67

Your job is to write a `computeReport(inFile,outFile)` function that reads a wage report for a number of workers (contained in file `inFile`) and creates an output file `outFile` that contains a summative report of all wages owed to each worker together with a final summary of the total wages reported for the day.

For the sample file below, the output is shown on the right. Don't worry about the formatting of dollar amounts that extend beyond two decimal digits.

Sample Input File (workerLog.txt)

```
Alfie Curtis          08:17 17:33  07.95
Barry Copping         12:10 20:11  06.80
Doug Beswick          06:00 14:20  08.25
Frazer Diamond        11:47 16:57  08.00
Gilda Cohen           20:48 03:27  08.00
Janice Burchette      13:11 19:50  08.25
John Chapman          17:36 19:05  06.80
Jon Berg              13:29 17:53  06.80
Lightning Bear        22:09 05:15  07.50
Maria De Aragon       14:43 20:36  08.25
Mark Austin           16:55 18:47  06.80
Paul Blake            16:07 23:39  07.50
Peter Diamond         13:43 19:22  06.80
Robert A. Denham      16:41 22:55  08.25
```

Sample Output File (outputReport.txt)

```
Alfie Curtis          $73.67
Barry Copping         $54.5133333333
Doug Beswick          $68.75
Frazer Diamond        $41.3333333333
Gilda Cohen           $53.2
Janice Burchette      $54.8625
John Chapman          $10.0866666667
Jon Berg              $29.92
Lightning Bear        $53.25
Maria De Aragon       $48.5375
Mark Austin           $12.6933333333
Paul Blake            $56.5
Peter Diamond         $38.42
Robert A. Denham      $51.425
TOTAL WAGES FOR DAY:  $647.161666667
```

Q3	Demonstrate knowledge of indefinite loop
Skills: TBA Lecture Dependency: Jan-30	Define a function `taylorApproximation(x, delta)` that computes and **returns** the number of iterations needed for the taylor sin approximation (from the previous homework) to compute a value that is within a positive delta of the actual `math.sin(x)` computation. Make sure that your function **returns** a single integer that represents the number of iterations needed. You can always confirm this by using the `taylorSin(x, n)` function from the previous homework and manually confirming that the computed result is within `delta` of `math.sin(x)`. For example: • `taylorSin(5,7)` `-0.9375840490206782` • `taylorSin(5,8)` `-0.9609213406827259` • `taylorSin(5,9)` `-0.9587763690226112` • `taylorSin(5,10)` `-0.9589331651965962` • `math.sin(5)` `-0.9589242746631385` As you can see, using just SEVEN terms is not sufficient, since the approximation is more than 0.213 away from the actual value. However, with EIGHT terms, the approximation is .00199 away from the actual value, which is less than 0.01 as shown below.

Sample Output in IDLE	`>>> taylorApproximation(5, 0.01)`
	`8`
	`>>> taylorApproximation(0.02, 0.01)`
	`1`

Sample Output in Canopy	`In[1]: taylorApproximation(5, 0.00001)`
	`Out[1]: 10`
	`In[2]: taylorApproximation(3.14159, 0.00001)`
	`Out[2]: 8`

Q4	Demonstrate knowledge of string manipulation
Skills: SM-1, SM-6, SM-9 Lecture Dependency: Jan-30	Define a function `parseDate(dateString)` that **returns** a list of three integers representing the month, day and year of the date represented by `dateString`. The dateString will exactly contain a month name (capitalized in English) followed by a space followed by the day (an integer) followed by a space followed by the year (also an integer). Note that all values in the returned list are all integers.
Sample Output in IDLE	`>>> parseDate('January 10 2014')` `[1, 10, 2014]`
Sample Output in Canopy	`In[1]: parseDate('February 6 2014')` `Out[1]: [2, 6, 2014]`

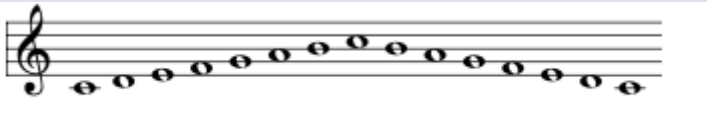
Q5	Demonstrate knowledge of string manipulation and raw input
Skills: TBA Lecture Dependency: Feb-4	Since we are moving away from using the `input` function, you need to supply a suitable replacement. Write a `readIntegerList()` function that has the user enter a string of values separated by commas. It returns a list of `int` values corresponding to the values in the string. If the user simply presses **Enter**, then the empty list must be returned. Note that all values in the returned list are all integers.

Sample Output in IDLE	`>>> readIntegerList()`
	`Enter a list of integers separated by commas: 4, 5, 2, 9, 10`
	`[4, 5, 2, 9, 10]`
	`>>> readIntegerList()`
	`Enter a list of integers separated by commas: 7`
	`[7]`

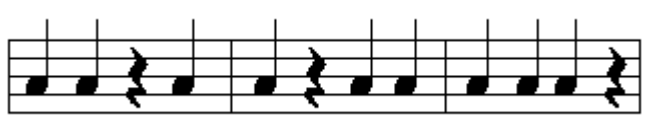
Sample Output in Canopy	`In[1]: readIntegerList()`
	`Enter a list of integers separated by commas: 9, 4, -3`
	`Out[1]: [9, 4, -3]`
	`In[2]: readIntegerList()`
	`Enter a list of integers separated by commas: <<ENTER PRESSED>>`
	`Out[2]: []`

Q6	Demonstrate for loop
Skills: TBA *Lecture Dependency*: *Jan-24*	Using the `generateSoundWave(frequency, n)` file from the previous assignment you can observe that the samples of sound data all are `float` values between -1 and +1. This is known as being **normalized**. When combining sound waves together (using the `listAdd(list1,list2)` method from the previous assignment) you will find that the samples fall outside this range. Write a `normalize(soundData)` function that takes a list containing sound data and returns a new list containing normalized sound data where every sample is between the values -1 and +1. To do this, find the **maximum positive value** in the `soundData` list and the **minimum negative value** in the `soundData` list. In the normalized list, all positive values are divided by **maximum positive value** while all negative values are divided by **–minimum negative value.** Feel free to use some methods in `numpy` that I have mentioned in class. Note that by definition 0 is already normalized; also, if the values in the list already are in the -1 to +1 range, then you can simply return the list itself since it is already normalized.
Sample Output in IDLE	`>>> normalize([-2, -1.5, -1, 0])` `[-1.0, -0.75, -0.5, 0]`

Sample Output in Canopy	`In[1]: normalize([0.5, 0.75, -0.33])`
	`Out[16]: [0.5, 0.75, -0.33]`
	`In[2]: normalize([1, 2, 3])`
	`Out[2]: [0.3333333333333333, 0.6666666666666666, 1.0]`

Q7	Demonstrate ability to solve problems by decomposing into existing functions
Skills: TBA Lecture Dependency: Jan-30	You can create sound wave data for a single note using `generateSoundWave(frequency, n)`. But what if you wanted to create a sound wave that represents a sequence of notes. As you can imagine, you need only **concatenate** the corresponding sound waves. Given the Wikipedia entry on piano frequencies you can identify the frequencies of the seven basic notes in the C scale, starting with middle C and moving upwards. These are the notes C4, D4, E4, F4, G4, A4, B4 in the table. You can see that the eighth note C5 is simply double the frequency of C4.  The above sequence of notes, musically, can be represented as 'CDEFGABXBAGFEDC'. We have to be creative in using the character X to represent the C5 frequency! You are to write a `createTrack(trackData, duration)` function that takes a string representation of notes in `trackData` and an integer `duration` representing the number of timed samples for each note. When duration is 44,100 then each note in trackData will play for one second; when 22,050 each note plays for ½ second.This function will return a list of sound data that represents the **concatenation** of the corresponding sound wave data for each note in succession. In the template provided the `playWave(data)` function plays the sound wave to your computer speakers, when run in Canopy. You don't need to demonstrate that the sound actually plays; rather make sure you can reproduce the sample output below. For example, `playWave(createTrack('CDE', 44100))` would play three seconds of sound, starting with the C4 note, then D4 and finally E4 note.

Sample Output in Canopy	`In[1]: createTrack('CDE', 5)`
	`Out[1]: [0.0,`
	` 0.03726675716643086,`
	` 0.07448173986101789,`
	` 0.11159324554186605,`
	` 0.14854971542704604,`
	` 0.0,`
	` 0.04182796613612572,`
	` 0.08358271891434575,`
	` 0.1251911731253657,`
	` 0.16658049963280974,`
	` 0.0,`
	` 0.0469467696588042,`
	` 0.09379001160049445,`
	` 0.1404264264088237,`
	` 0.18675317076582615]`

Q8	Putting it all together
Skills TBA *Lecture Dependency* *Jan-24*	Write a function that loads up individual "tracks" of music from a file. A file can contain a number of lines, each of which contains a sequence of notes as described in question 6. Using the `createTrack(trackData, duration)` function from Q7 you will create soundwave data for each track that you can **compose** together with each other, using the `listAdd(list1, list2)` function that you wrote in the previous assignment. Once you have composed all of the tracks, use the `normalize()` function to ensure that the data conforms to the -1 to +1 range. You can assume that the individual tracks all contain the exact same number of notes. In doing so, they will all have the same length, which is a requirement of the `listAdd` function. In music, there is the concept of a "Rest" which is a period of time that no note plays. For example:  The above represents three measures playing the A note. Each individual measure (from left to right) is represented as follows: • AARA • ARAA • AAAR The track containing these three measures would be encoded as 'AARAARAAAAAR'. You will have to modify the encoding in `createTrack()` to handle the rest notes. Your function `mergeTracks(filename, n)` must **return** a list containing the data composed from the individual tracks as found in the `filename` file. The parameter n refers to the number of samples to use for each note. Thus if n = 11025, each note will be played for ¼ of a second. If you succeed in this problem, you can pass the results of `mergeTracks` into `playWave` and hear the song which you have composed.
Sample Output in IDLE	`>>> mergeTracks("C chord.txt", 3)` `[0.0, 0.14003432054986548, 0.2797392677883622]`

Sample Input File (StarWars™ Theme Song)
CRRRGRRRFEDXRRRGRRRFEDXRRRGRRRFEFDRR

Sample Input File (Indiana Jones™ Theme Song). This File will have two lines for two separate tracks
RRRRRRRRRRRRRRRRRRRRRERFGRRXRRRRRDREFRRRRRRRRGRABRRFRRRRRCRDERRFRRGRRERFGRRXRRRRRDREFRRRRRRRRGRGERRDRGERRDRGERRDRGFRRERDCR GRRRGGGGGRRRGRRRGGGGGRRRGRRRGGGGGRRRGRRRGGGGGRRRGRRRGGGGGRRRGRRRGGGGGRRRGRRRGGGGGRRRCRRDRCCRRRRCCRRRRCRRRCRRCR

Draft Version: 2/6/2014

How To Get Started On This Assignment

A template HW4.py file is provided to you with some sample functions already provided.

Feel free to take the implementations in the provided HW3 solution, in case you do not have a working `listAdd` or `generateSoundWave` function.

Much of the work for this assignment will be spent trying to understand the domain of sound waves and writing the appropriate Python code. In many ways, that is as it should be! The job of a programmer is more than learning a particular syntax. You need to know how to produce code relevant for a specific problem. Sometimes the code you write is only 5 lines of code (but it will be just the right five lines of code).

Submit your HW4.py file using the web-based turnin system. As we have mentioned in class, only one of the team members needs to submit the assignment. But just make sure that something gets submitted!

Change Log

1. Refined the sample input (and output) for Question 8
2.