COMP 1012 Fall 2013 Assignment 3

Due Date: Thursday, November 7, 2013, 11:59 PM

Material Covered

- user input
- if … elif … else
- string functions
- numpy arrays and functions

Notes:

- Name your script files as follows: `<LastName><FirstName>A3Q2.py` and `<LastName><FirstName>A3Q3.py`. For example, `LiJaneA3Q2.py` is a valid name for a student named Jane Li. If you wish to add a version number to your file, you may add it to the end of the file name. For example, `SmithRobA3Q2V2.py` is a valid name for Rob Smith.
- Follow the posted programming standards to avoid losing marks.
- You must submit a ***Blanket Honesty Declaration*** to have your assignment counted. This one honesty declaration applies to all assignments in COMP 1012.
- To submit the assignment follow the instructions on the course website carefully. You will upload Python both script files and output via the course website. There will be a period of about a week before the due date when you can submit your assignment. ***Do not be late!*** If you try to submit your assignment after the deadline, you will get a message indicating that the deadline has passed.

Question 1—Programming and People [3 marks]

Answer each question ***briefly***. Record your answers in the text boxes of the online assignment submission form. Make sure you read all the footnotes.

a. What was the first commercial airplane to be designed entirely on a computer? When?
b. What is CAD software, and which CAD package was used to design the aircraft in part a?
c. What is one computer package you could download and use to design and implement your own photorealistic computer games, with realistic physics?

Question 2—Hangman Game [10 marks]

Description

In the traditional Hangman game (see http://en.wikipedia.org/wiki/Hangman_(game)), a participant guesses letters one at a time until they guess a word, or they have made so many errors the game is lost. You will program a Hangman game, including drawing the picture. There is a sample session on the next page.

```
WELCOME TO HANGMAN!
YOU WILL GUESS A WORD WITH 6 LETTERS!
Each time you get the answer wrong, we will add a stroke.
 _____
|     |
|
|
|
|
|
|       Letters already guessed:                                  _ _ _ _ _ _

What is your next letter? (or type your guess) e                                ①

There are 2 Es!
 _____
|     |
|                                                                               ②
|
|                                                                               ③
|
|        Letters already guessed: E                               E _ _ _ E _

What is your next letter? (or type your guess) t

There are no Ts
 _____
|     |
|     O
|                                                                               ④
|
|
|        Letters already guessed: ET                              E _ _ _ E _

What is your next letter? (or type your guess) t                                ⑤

Invalid entry: enter a single letter you have not already guessed

What is your next letter? (or type your guess) ao

Invalid entry: enter a single letter you have not already guessed

What is your next letter? (or type your guess) a

There is 1 A!
 _____
|     |
|     O
|
|
|
|
|        Letters already guessed: ETA                             E _ A _ E _

What is your next letter? (or type your guess) n
```

```
There is 1 N!
 _____
|     |
|     O
|
|
|
|
|        Letters already guessed: ETAN                                  E N A _ E _


What is your next letter? (or type your guess) enamel                              ⑥

Congratulations! You win!!
The word was E N A M E L

Programmed by Instructors
Date: Mon Oct 21 16:03:53 2013
End of processing
```

Notes:

① Prompt the user to enter a single letter. It can be entered in upper or lower case, but it should be converted to upper case.

② If the letter is found in the mystery word, the hangman figure does not change.

③ If a letter appears more than once, show all the locations.

④ If the guessed letter is not found, draw in a single new character on the hangman. The full hangman figure looks as shown below on the left. If for some reason you do not want to draw the traditional hanged man, you may draw the sun symbol on the right instead. The order of adding symbols is also shown for each, with T and E used for 10 and 11.

```
 _____          \|/          _____          \45
|     |        --O--        |     |        ET367
|     O         /|\         |     1         928
|   --|--         |         |   54267         1
|     |           |         |     3
|    / \                    |    8 T
|   /   \                   |   9   E
|                           |
```

⑤ No letter, a symbol that is not a letter, a letter that has already been guessed, or a combination of letters that is not the same length as the word (that is, it could not be a guess) is rejected and the user gets another chance. No extra strokes are added to the figure.

⑥ A user response of the same length as the mystery word is taken as a guess. If it is correct, announce a winner. If it is not correct, say so (not shown) and add a stroke to the figure. A guess of a single letter that finishes the word should be treated as a successful guess. If the entire figure is drawn, they lose if their next guess is not correct.

Functions

You may find it useful to use the following built-in functions and methods.

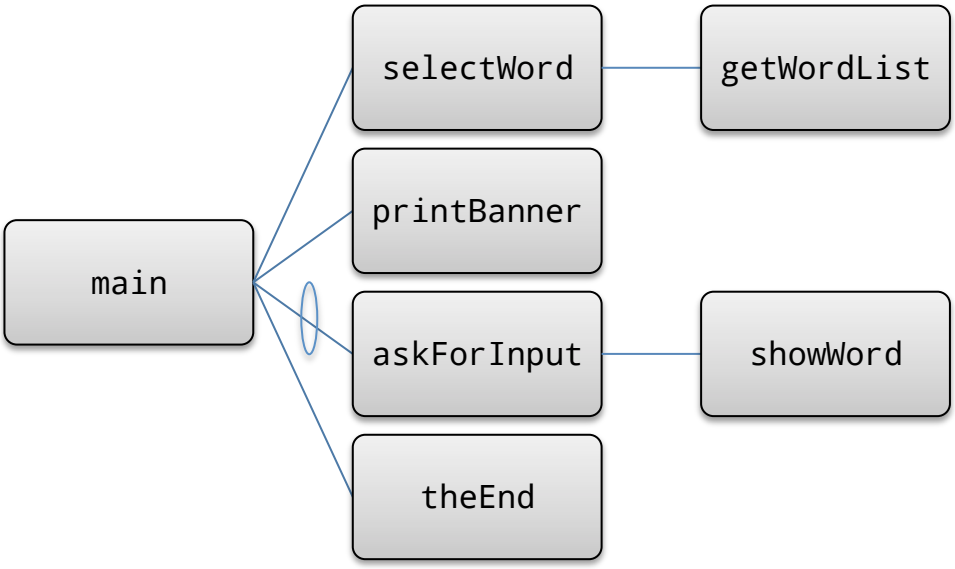
- `set(word)` creates a *set* of the unique characters in `word`; `list(set(word))` returns a list of the unique letters.

- `word.count("x")` counts the number of times *x* appears in `word` and returns the count.
- `word.replace("xyz", "ab")` creates a new string from the contents of `word`, with *xyz* replaced by *ab* everywhere. Note it doesn't change `word`; you have to save the changed version.
- `word.strip()` returns the string referenced by `word` without leading and trailing blanks.
- `word.upper()` returns the string referenced by `word` all in upper case.

All the code you write should be in functions, with the exception of imports, assignment of global constants (if any) and a single line following all function definitions that reads:

```
main()
```

Here is the organization of functions you must define and use:



Your code must define the following functions, using the parameters specified. You may define and use other functions if you want. All functions but `main` must have a clear docstring explaining their usage.

- `main():` This function manages the entire game. It calls `selectWord` to get a random word to be guessed. It calls `printBanner` to display the opening information. Then it has a loop to handle guesses from the user. In the loop it calls `askForInput` to get *valid* user entries (invalid entries are not returned). It detects when the game is over, either because there are no more letters to be guessed, or because there are no more changes to make to the figure. Finally it calls `theEnd` to print out the usual termination message.
- `askForInput(word, guessed, pic):` This function displays the current picture (that is, the parameter `pic`), the letters that have been already guessed (from parameter `guessed`) and the mystery `word` as formatted by a call to `showWord`. It gets an input from the user, converts it to upper case and checks it for validity. A valid input is a single letter from A to Z that has not already been guessed, or else it is a guess of the entire word, of exactly the right length. Loop to prompt the user over and over until a valid input is received, and then return it to the calling code. You may use different wording from the sample output, but it must display the same information.
- `getWordList():` This function reads the content of the file `2of12inf.txt`. This file contains one word per line. Make a list of the words and return the list. Some of the words

that are unusual plurals contain a '%' character. Remove it before adding the word to the list. Use code something like this, where `urllib` is a library you must import:

```
url = "http://cs.umanitoba.ca/~comp1012/2of12inf.txt"
flink = urllib.urlopen(url)
for eachline in flink :
    # eachline contains a word to be saved
```

- `printBanner(word):` This function prints out the introductory message to the user, telling the user how many letters there are in the mystery word. You may use your own wording instead of the message shown in the sample output, but it must clearly express the same facts.
- `selectWord(minLength, maxLength):` This function returns a word of at least `minLength` letters and no more than `maxLength` letters. It gets the word by selecting it randomly from a word list it receives from `getWordList`. All words of an appropriate length in the word list should have an equal chance of being selected.
- `showWord(word, guessed):` This function returns a string containing the mystery `word`, with spaces between letters, and an underscore for letters not in `guessed`. For example, if `word` is `"FLEECE"` and `guessed` is `"E"`, it will return `"_ _ E E _ E"`
- `theEnd():` This function prints out the termination message for the program. It must be called as the last instruction in `main`.

Hand-in

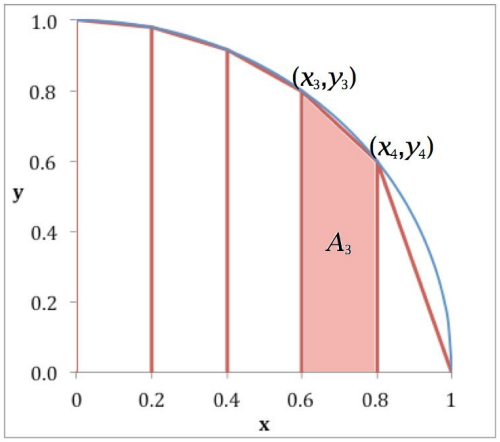
Hand in a sample run of your code like the sample output shown. It must include:

- Guesses of letters that are present and letters not present in the mystery word. At least one letter present must occur in more than one location.
- Invalid entries including no letter, a non-letter symbol, 2-5 letters together, and a letter that was previously guessed.
- An incorrect guess of the whole word, and a correct guess of the whole word.

Question 3—Area Under a Curve

Description

When you study integral calculus you learn to evaluate the area under a curve mathematically. This is easy to do for some functions, but much more difficult for others. The area under a curve can also be found numerically by doing a large number of computations. We use a method that approximates the correct area by breaking it up into a large number of small regions with areas that we can compute easily.



For example, to find the area of a circle, we can approximate it by the sum of the areas of a large number of trapezoids. Let's focus on ¼ of a circle, as we can always find the area of the entire circle by multiplying by 4.

If two points (x_3,y_3) and (x_4,y_4) lie on a curve, and we join them by a straight line, and also draw lines to the horizontal axis, we form a trapezoid. In the figure, the

blue line outlines a quarter of a circle, and the pink shaded area between (x_3,y_3) and (x_4,y_4) defines a trapezoidal shape. The area of the trapezoid is the width of the base times the average height:

$$A_3 = (x_4 - x_3)(y_3 + y_4)/2$$

Similar areas can be computed for the other trapezoids in the figure. You can add them all together to get an approximation of the area of the quarter circle.

In more general terms, if you want to find the area between the horizontal axis and the function $f(x)$ with x between two positions a and b, then compute the following sum:

(1) $$A = \sum_{i=0}^{n-1} [\text{area of trapezoid between } x_i \text{ and } x_{i+1}] = \sum_{i=0}^{n-1} \frac{(x_{i+1}-x_i)(f(x_i)+f(x_{i+1}))}{2}$$

where the sequence of positions $x_0, x_1, \ldots, x_n$ is in increasing order, and $x_0 = a$ and $x_n = b$.

For example, to find the quarter area of the unit circle, use $a = 0$, $b = 1$, the equally-spaced points shown in the figure, and

(2) $$f(x) = \sqrt{1 - x^2}$$

the following table shows the calculations. The total is a rough approximation to the actual area of $\pi/4 = 0.785$. With more intervals, the approximation would get much better.

i	x_i	$f(x_i)$	A_i
0	0.0	$1.00^{0.5} = 1.00$	$0.2(1.00+0.98)/2 = 0.198$
1	0.2	$0.96^{0.5} = 0.98$	$0.2(0.98+0.92)/2 = 0.190$
2	0.4	$0.84^{0.5} = 0.92$	$0.2(0.92+0.80)/2 = 0.172$
3	0.6	$0.64^{0.5} = 0.80$	$0.2(0.80+0.60)/2 = 0.140$
4	0.8	$0.36^{0.5} = 0.60$	$0.2(0.60+0.00)/2 = 0.060$
5	1.0	$0.00^{0.5} = 0.00$	
		Total	0.760

What to do

Write a ***properly documented*** script file that divides up the interval between 0 and 1 into a large number of equal-size intervals and approximates the area of the unit circle (in other words, π) by 4 times the sum of trapezoidal areas. Your program should ask the user how many intervals to use, and how many decimal places are needed. Then starting with the requested number of intervals, keep doubling the number of intervals until the numerical estimate of π matches the built-in value `math.pi` to the requested number of decimal places.

Details:

- The sample run uses 7 decimal places; use 11 decimal places in your output.
- The number of decimal places shown should automatically adjust to be one greater than the number requested; in the sample there are 8 decimal places (hint: use `round`).
- Keep the π estimates and the final value of π shown lined up at the decimal point.
- After completing your script, in the shell call `areaUnderCurve(1.,100.,np.log,10000)` and display the result. That evaluates the area under the log function from 1 to 100. Notice that once your script is run, functions defined in the script are still available in the shell.

Right after that, evaluate the actual integral of $\log(x)$ from 1 to 100, which is $g(100) - g(1)$, where $g(x) = x \log x - x$ (see the example). You do not have to define a function $g(x)$.

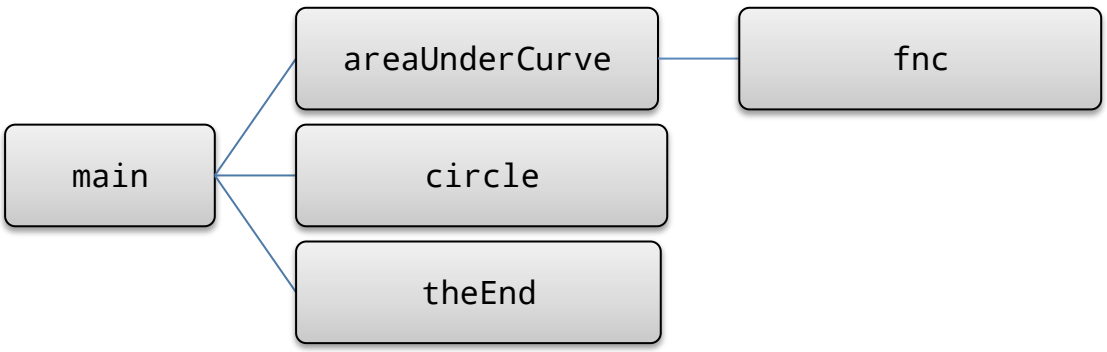
Sample Output

The following output shows what you are to do, but it converges to only 7 decimal places, whereas your output should converge to 11 decimal places. Your output should start with 1000 intervals as shown. The two shell commands shown after the script output should be similar, except that you will use 10000 for the number of intervals instead of 1000 shown. The result will be a little different.

```
Enter the number of decimal places for pi: 7

Enter the number of intervals to use to start: 1000

Using    1000 intervals, the estimate of pi is 3.14155547
Using    2000 intervals, the estimate of pi is 3.14157951
Using    4000 intervals, the estimate of pi is 3.14158801
Using    8000 intervals, the estimate of pi is 3.14159101
Using   16000 intervals, the estimate of pi is 3.14159207
Using   32000 intervals, the estimate of pi is 3.14159245
Using   64000 intervals, the estimate of pi is 3.14159258
                     The value of pi is about 3.14159265

Programmed by the Instructors
Date: Tue Oct 22 15:57:26 2013
End of processing

In [2]: areaUnderCurve(1.0, 100.0, np.log, 1000)
Out[2]: 361.51621028239975

In [3]: 100. * (math.log(100.) - 1.) + 1.
Out[3]: 361.51701859880916
```

Functions

Your program should have a structure like this:



As in question 2, a `main` function does the work, instead of coding shell commands right into the script file. Here is what each function does:

- `main()`: prompts the user for the number of decimals and the the number of intervals to use. Assume the inputs are a valid integers. It then repeatedly evaluates the area of the quarter unit circle using trapezoidal areas by calling `areaUnderCurve`., continuing until the result matches `math.pi` to the required number of decimal places. Each time it prints a π

estimate, as shown in the sample output above. Finally it calls `theEnd` (the same one as in Question 2).

- `areaUnderCurve(left, right, fnc, numberOfIntervals)`: this function finds the area under the curve defined by the function `fnc(x)` (that is, the function `fnc` must take have a single parameter, which should be converted to an array, and it returns an array of numerical function values, like `np.sin` or `np.sqrt`.) The function `areaUnderCurve` calculates the positions of the *x*-values and evaluates `fnc` at those *x* positions to determine the *y*-values, then combines them to find the trapezoidal areas which it totals, all as described earlier in Equation (1). The *x*-values should be evenly distributed between the two endpoints, `left` and `right`. Note that `numberOfIntervals` is one less than the number of *x*- and *y*-values to be evaluated. The value returned should be a float. You are NOT allowed to use a loop in `areaUnderCurve`. Use array calculations instead.
- `circle(xpos)`: evaluates $\sqrt{1 - xpos^2}$, the function that defines the top curve of a circle. This function has one argument, which should be converted to a numpy array, and returns a numpy array of function values. It should be appropriate to use as a `fnc` argument in a call to `areaUnderCurve`. You are NOT allowed to use a loop in `circle`.
- `fnc(xpos):` `fnc` is a parameter of `areaUnderCurve`. Which function it is depends on the call to `areaUnderCurve`. You do not have to code `fnc`, since the name is just an alias for some other function, such as `circle`.
- `theEnd( )`: This function simply produces the final output that terminates the program. It has no parameters, so the brackets are empty. It does not return a value. It should be the same function as in Assignment 2, and there are no marks for providing it, but you can lose marks if it is missing.

Hand-in

Hand in the results of running the code with 1000 starting intervals, and a target of 11 decimal places. You should also include the output from two shell commands as shown, using 10000 intervals for the first one instead of the 1000 intervals shown in the sample output.