CSCI A201/A597 - Introduction to Programming I

Fall 2017

Programming Assignment 12

[100 points total]

Due Date

- at 4:00PM, on Monday November 13, 2017.

Work Policy

Programming Assignment 12 is to be done individually - no group solutions or cooperative work.
You may discuss the programming assignment with other A201/A597 students, but you have to write the solutions and implement programming answers by yourself. If you discuss the programming assignment with other A201/A597 students, you must acknowledge them by including their names in a comment at the top of the Python files you turn in for this programming assignment. Do not share any source code or programming assignment answers with any students.

Notes

Keep a written log of your work for Programming Assignment 12, in a separate text document named **`pa12-log.txt`**, in which you annotate your work on these tasks.

At the top of the **`pa12-log.txt`** file, write:

- "A201/A597 Programming Assignment 12"
- your first and last name
- your IU username (that's the same as your IU email, *not* your numerical ID)

For Programming Assignment 12, you'll be using turtlegraphics, specifically the following functions/methods/classes:

- `.Screen()` # to create a "canvas" window for drawing in it
- `.Turtle()` # to create a "turtle" that can draw on the "canvas"

- `.up()` and `.down()` # to stop/start drawing

  relative movements of the turtle:
- `.forward(units)` # to move the turtle from wherever it is, in the direction it's facing, by a certain number of units (pixels)
- `.left(degrees)` and `.right(degrees)` # to turn the turtle's current direction by a certain number of degrees

  absolute movements of the turtle:
- `.goto(x,y)` # to place the turtle at a certain position (x,y) on the canvas (the direction of the turtle remains unchanged)
- `.setheading(degrees)` # to set the turtle's current direction to a certain direction on the canvas (the position of the turtle remains unchanged)

When you've completed and run your programming tasks on IDLE, save a transcript of your IDLE session to a file named `pa12-transcript.py`, and include that with your submission. Make sure that your IDLE transcript includes interactions with running *all* of your programming assignment scripts. As for previous programming assignments, `pa12-transcript.py` is considered an essential part of your programming assignment when grading.

For Programming Assignment 12, it is ***required*** to include properly formatted `docstrings` with purpose statements and signatures for all functions you define.

If you don't include properly formatted `docstrings` for all functions you define for Programming Assignment 12, a total of 50% will be *deducted* from your Programming Assignment 12 grade *after* implementations are graded. (for *all functions*: include docstrings, purpose statements, and signatures)

Tasks

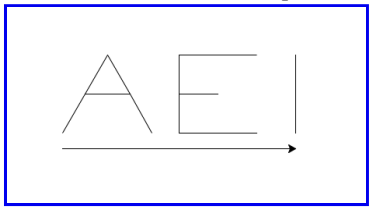
A. Write the following three functions, all in the same file named **`pa12-letters.py`**, which will also contain your main program code.

1. `drawLetter?(pTurtle, pSize)`:
   First, ***rename*** this function to the letter you choose to draw: for example, if you choose to draw the letter "Q" (not recommended, since it's tricky to reproduce its shape with what we know so far about turtlegraphics!) then the function needs to be named drawLetterQ.
   This function needs to draw a letter of your choosing, using `pTurtle` as the turtlegraphics turtle that's going to do the drawing. The letter you choose needs to be `pSize` pixels tall.

2. `drawLetter?(pTurtle, pSize)`:
   First, ***rename*** this function to another letter you choose to draw: for example, if you choose to draw the letter "B" (also not recommended for the moment, since it's tricky to reproduce its shape with what we know so far about turtlegraphics!) then the function needs to be named drawLetterB.
   This function needs to draw a letter of your choosing, using `pTurtle` as the turtlegraphics turtle that's going to do the drawing. The letter you choose needs to be `pSize` pixels tall.

3. `drawLetter?(pTurtle, pSize)`:
   First, ***rename*** this function to a third letter you choose to draw: for example, if you choose to draw the letter "J" (still not recommended, since it's tricky to reproduce its shape with what we know so far about turtlegraphics!) then the function needs to be named drawLetterJ.
   This function needs to draw a letter of your choosing, using `pTurtle` as the turtlegraphics turtle that's going to do the drawing. The letter you choose needs to be `pSize` pixels tall.

Once you have defined the three functions, you need to use them in your main program, to "write" the three letters using turtlegraphics. The three letters also need to be *underlined* by another line drawn by turtlegraphics.
For example, if you chose "A" "E" and "I", your output may look similar to this, where the second parameter for each called function is the number 100, and as a consequence each letter is *100 pixels tall*:



As starting code, we are providing you these Python programs:

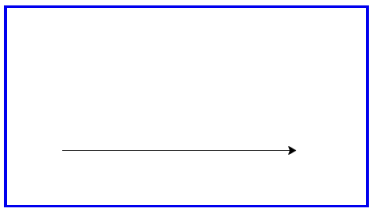
```
import turtle             # import turtle graphics
import math

# global variables are defined here:

gWindow = turtle.Screen() # create a window for drawing in it
gPencil = turtle.Turtle() # create a "turtle" that can draw


# TODO: functions to be used for drawing need to be defined here:

# def drawSomething(....)
#     ....
#     ....

# def drawSomethingElse(....)
#     ....
#     ....



# the main program starts here:


# TODO: here write the code for drawing three letters,
#     for example "A E I" as in the image above
#     each letter needs to be drawn by
#     its own Python function that you defined above!



# conclude the drawing by underlining the three letters...

gPencil.up()              # lift the pen (no drawing until pen back down)
gPencil.goto( -200, -20 ) # go to first point position
gPencil.down()            # put the pen down on canvas (start drawing)

gPencil.setheading( 0 )   # set the mouse heading (drawing direction) to 0
gPencil.forward(300)      # draw 300 pixels
```

and an alternate version, with a different (but just as correct) way of importing:

```
from turtle import *            # import all from turtle graphics
import math

# Programming Assignment 12 starting code


# global variables are defined here:

gWindow = Screen() # create a window for drawing in it
gPencil = Turtle() # create a "turtle" that can draw


# TODO: functions to be used for drawing need to be defined here:

# def drawSomething(....)
#     ....
#     ....

# def drawSomethingElse(....)
#     ....
#     ....



# the main program starts here:


# TODO: here write the code for drawing three letters,
#     for example "A E I" as in the image above.
#     Note: each letter needs to be drawn by
#     its own Python function that you defined above!



# conclude the drawing by underlining the three letters...

gPencil.up()              # lift the pen (no drawing until pen back down)
gPencil.goto( -200, -20 ) # go to first point position
gPencil.down()            # put the pen down on canvas (start drawing)

gPencil.setheading( 0 )   # set the mouse heading (drawing direction) to 0
gPencil.forward(300)      # draw 300 pixels
```

(click to download the actual python files)

The above code doesn't do much, but has lots of potential... for the moment, it draws just this:



B. **Build histogram data of all characters contained in a string**:

Implement the following, in a file named **`pa12-histogram.py`**, which will also contain your main program code.

For this Task, you'll be going back to the code you wrote for Programming Assignment 07. You may ***reuse*** any of the solution code *you wrote* for Programming Assignment 07, i.e. reuse your PA07 code in the functions you're writing for **`pa12-histogram.py`** (however, make sure that you don't include the code as *main program code* for this Programming Assignment 12 -- here it needs to be implemented as a function!)

- Define the function `string_to_hist()`, which takes a string, and returns a dictionary containing ***histogram*** data, based on the letter frequencies in the given string.

  So for example, if the string s had the following contents:

  ```
  s = "I have never seen a purple cow, And I never hope to see one. But I can tell you anyhow, I'd rather see than be one."
  ```

  Then `string_to_hist(s)` would return a dictionary containing the following ***histogram*** data:
  ```
  {'I': 4, ' ': 25, 'h': 5, 'a': 6, 'v': 3, 'e': 18, 'n': 9, 'r': 5, 's': 3, 'p': 3, 'u': 3, 'l': 3, 'c': 2, 'o': 7, 'w': 2, ',': 2, 'A': 1, 'd': 2, 't': 5, '.': 2, 'B': 1, 'y': 2, "'": 1, 'b': 1}
  ```

  (note that a dictionary is never sorted according to any specific order, so the same histogram may also be represented like this:
  ```
  {' ': 25, "'": 1, ',': 2, '.': 2, 'A': 1, 'B': 1, 'I': 4, 'a': 6, 'b': 1, 'c': 2, 'd': 2, 'e': 18, 'h': 5, 'l': 3, 'n': 9, 'o': 7, 'p': 3, 'r': 5, 's': 3, 't': 5, 'u': 3, 'v': 3, 'w': 2, 'y': 2}
  ```
  or in any other order)

  For your own testing, you may want to print the output obtained from `string_to_hist()` (or simply call `string_to_hist()` in the Python IDLE shell) on a few different string inputs, to make sure that it produces the desired output.

C. **Draw a histogram of all characters contained in a string**:

Implement the following, in a file named **`pa12-histogram.py`** (the same file as for Task B above), which will also contain your main program code.

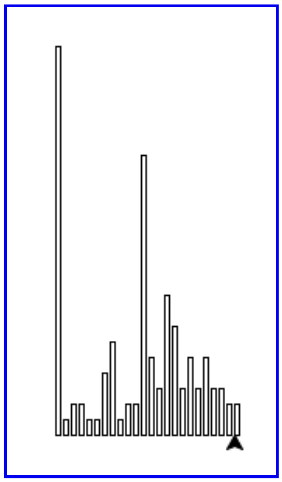
For this Task, you'll be implementing *drawing* the histogram using turtlegraphics.

You may ***reuse*** the function you wrote for Task B, as well as what you wrote for Task A, i.e. include those functions (or modify them to fit this task needs) in your file named **`pa12-histogram.py`**.

1. `drawBlock(pTurtle, pWidth, pHeight)`:
   Write a *helper* functions, i.e. a function that you will be using several times through your program, to help you with a specific task: drawing a single block (i.e. a rectangle) from the current position, using the turtle, with the width and height sizes passed as parameters. Name this function `drawBlock()`.

2. Once you have defined the `drawBlock()` function, you need to use `drawBlock()` in your main program, to draw a graphical representation of the ***histogram*** of all letters contained in the string, using turtlegraphics!

A sample drawing, showing a graphical representation of the same histogram given as example in Task B above:



PS: you may use the `.speed("fastest")` method for your turtle to draw a bit faster ... please read the relevant section in the official Python turtlegraphics documentation for more information.

Programming Assignment 12 Submission Instructions:

1. When you submit your Programming Assignment 12, also include a text file named **`pa12-log.txt`**, with your annotations as explained above.

2. When you submit your assignment on IU Canvas, it should consist of all the files detailed above..
    1. At the top of each one of your files, include "*A201 / Fall 2017*" (or "*A597 / Fall 2017*"), "*Programming Assignment 12*", your *full name* and your *IU username* (i.e. your IU account).
    Note: in your Python files (file extension .py), kindly include the same information at the top of the file, within Python `# comments`.
    2. Make sure that all your Python and plain-text documents are clearly readable by anyone; Python `.py` files, as all submitted plain-text documents, need to use either:
        - ASCII encoding (7-bit, allows no accents/smart quotes/etc.), ← preferred for Python source code submissions that only contain English-alphabet characters & symbols.
        - UTF-8 encoding (capable of encoding all sorts of characters) ← only for any submissions that *require* non-English-alphabet characters & symbols.

    (...otherwise your output may end up garbled...)
    3. Do not use any other document file formats: submitted files *other* than plain text (file formats such as .pdf, .doc, .docx, .html, .xml, etc...) will be *rejected*. In other words, all submitted `.txt` and `.py` files need to be plain text.
3. Turn in your Programming Assignment 12 files by 4:00PM on Monday, November 13, 2017, on IU Canvas.
4. When you turn in your programming assignment on IU Canvas, it is your responsibility to verify that your file(s) have been uploaded, and their content on the IU Canvas server.
5. Please do not omit your IU username from the content.

Rejection Warning

Please double-check the specific requirements for the tasks above, so that your submission doesn't get rejected. Also, please remember course policies about turning in assignment files, in particular about Rejected Assignments: if your Python script crashes, if you submit any part of this assignment in the incorrect format, etc. then your assignment will be rejected and you will be given 0 points for the assignment until you fix the problems and resubmit it. If you are in doubt about whether you're using the correct file formats, ask one of the instructors.

Getting Help

If you need help with Programming Assignment 12, or any other A201/A597-related topic:

- attend **office hours**, as listed on the Syllabus page
- ask the AIs, UIs, and Instructor:
  the recommended way to contact A201/A597 instructors is with a **Canvas message**.
- consult the textbook
- consult lecture notes
- consult the references provided in reading assignments

Last updated: 2017-11-09
mitja@indiana.edu