CSCI A201/A597 - Introduction to Programming I

Fall 2017

Homework 12

Due Date

- Thursday, November 09, 12:59pm on IU Canvas.

Work Policy

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

Homework Tasks [total: 100 points]

Topics: this week's Homework continues with OOP topics. Please *refer* to this week's Lecture 12 notes and Reading Assignment 12. For example, you'll need to read carefully the Python3 Tutorial (`www.python-course.eu`) sections about Magic Methods and Operator Overloading to understand how to implement the `__init__()` and `__repr__()` methods for the classes you'll be writing for Homework 12 (...and about how `__repr__()` differs from the `__str__()` method we've seen at lecture time...).

For this assignment, you'll be creating a class called `Point` for representing (... you guessed it!) a point on a 2D surface (a 2D *plane*). The class `Point` you will implement has to store information about the point's:

- 2D position: x- and y- coordinates, both integers
- color: for now this is a string.

You may choose the names of your class's ***attributes***, and you have to document the attributes properly in the class's *docstring*.
Every function definition, including ***method*** definitions, must have a *docstring* (with both a purpose statement and signature).
When you've completed and run your programming tasks on IDLE, save a transcript of your IDLE session to a file named `hw12-transcript.py`, and include that with your submission. Make sure that your IDLE transcript includes interactions with running *all* of your homework scripts. As for previous homework, `hw12-transcript.py` is considered integral part of your homework when grading.

Put all of the code for this assignment into a script called `hw12-point.py`

A. [40 points]
Write a class definition for `Point` :
   1. write a *docstring* for the class:
      the class's *docstring* has to include a *purpose statement*, and explain all the instance *attributes* (and the types of those attributes) that are going to be used by your class instances.
   2. write an `.__init__()` method for the class.
      This constructor method needs to take three arguments: x- and y- coordinates, and color; the

constructor needs to properly assign the appropriate *values* to the appropriate instance *attributes*.
For example, the command `Point(20,30,"purple")` should instantiate a `Point` object with coordinates (20,30) and the color "purple".

3. write a `.__repr__()` method (see references*) for creating a string to represent the given `Point`.
Note: the `.__repr__()` should *never* print anything on the screen; to get full points for this part, your __repre__() method should *return* a string that represents the `Point`, in a way that could be directly interpreted by the Python interactive shell in IDLE. So if `p1` refers to a purple `Point` with coordinates (20,30), then its representation would be the string `"Point(20,30,'purple')"`.
4. write methods `.vertical_move()` and `.horizontal_move()` for your `Point` class.
For each one of these two methods, take one integer, and *modify* the `Point` instance accordingly, i.e.: the `Point` instance needs to be modified by *adding* the given number to its y-coordinate (for the `.vertical_move()` method) or the x-coordinate (for the `.horizontal_move()` method).
For example, if `point1` is a `Point` with x- and y- coordinates (3,25), calling `point1.horizontal_move(9)` would change the x- coordinate of `point1` to 12 (while keeping the y-coordinate and the color values the same as before).
And another example: `point1.vertical_move(-6)` would change the y- coordinate of `point1` to 19 (and keep the x- coordinate and the color the same).
5. write a method named `.distant_from()` that takes a single `Point` instance as argument, and returns the distance from that `Point` passed as argument, to the current point.
Note: you may need to use the `sqrt()` function to calculate square roots. That'll require importing the `sqrt()` function from the `math` module at the beginning of your program: put the command `from math import sqrt` at the top of your script. (such `import` statements should be placed at the top of your program, *before* any class or function definitions)

B. [20 points]
Test your `Point` class, by writing some `main` program code after your class definition, in the same `hw12-point.py` file.
Some first requirements for this task:

- *before* every step that either *makes something* or *changes something*, write a `print()` statement to explain what is about to happen.
For example, `print("Instantiating a Point object, in a variable named point1, at coordinates (20,30) ...")` or `print("The point2 is being moved by 6 units to the left...")`).
By using these `print` statements, if any part of your code crashes, we'll know where it happened.
- *after* each step that makes or changes something, write a `print()` statement to explain the results of the change that was just completed.
For example, `print("point1 has now become: " + repr(point1))`), so that we can make sure that the changes happened properly (as well as test the `.__repr__()` method).

Your `main` program has to perform at the very least the following steps:

1. Instantiate at least two `Point` objects, each having different positions and colors. (this will also help checking that your `.__init__()` method, as well as the structure of the class definition, work)
2. Move at least one of your `Point` objects vertically, using `.vertical_move()`.
3. Move at least one of your `Point` objects horizontally, using `.horizontal_move()`.
4. Change the color of at least one of your `Point` objects, by assigning a new value to the color attribute directly (without calling any method).
5. Compute (and print) the distance between the two points, by using `.distant_from()`.

For all the above steps, you need to print what happens before and after the step is executed.

C. [40 points]
For this task, you are going to have Python actually *draw* the `Point` objects, using a graphics module that is part of the standard Python distribution: the module is called `turtle`. This simple graphics `turtle` module `"simulates"` a little *turtle* (it doesn't actually draws a turtle on the screen, only a stylized arrow)

that carries a pen. By invoking the appropriate functions, you can write Python code to command the turtle to walk (e.g. `forward(10)`), to turn (e.g. `right(30)`), to pick up the pen (e.g. `penup()`), and many other actions, albeit we won't use most of those actions for this homework.

If you are interested, you may read more about the `turtle` module in Python at the official Python documentation web site.

For this homework, you will *need to know* these three `turtle` functions:

- goto(): takes x- and y- coordinates (two integers), and the turtle will walk to that point as a result
- dot(): takes the diameter and the color (an integer and a string), and the turtle will draw a dot wherever the it happens to be at that moment, where the dot is of the specified size (diameter) and color.
- penup(): instructs the turtle to 'pick up the pen', so that it doesn't draw while walking to a new location (for this homework, you will need to use this function just once, at the beginning of your program).

Parts you need to complete for this task:

1. include the command `from turtle import *`. The 'asterisk character' * will cause *all* of the functions to be imported from the `turtle` module, so that you can use them anywhere in the script. Place this command at the beginning of your program, before any of your class or function definitions, as well as before any testing code.

2. Invoke the function `penup()` after the `import` statement(s). This will prevent drawing lines along the way, when the turtle moves to a new location to draw a new point.

   You may also call the `hideturtle()` function, if you don't like seeing the "turtle" arrow on the screen. Likewise, you may call `speed(0)` to speed up the turtle.

3. Write a `.draw()` method in to your `Point` class, taking no extra arguments, to draws an appropriately colored point at the `Point` object's coordinates, with a default 4-pixel diameter.
   Hint: use the functions `goto()` and `dot()` from the `turtle` module.

   Note that the turtle function `dot()` is different from any object instantiated from your constructor `Point()` for your `Point` class: your class is to represent a point in your program code, the dot() function is to draw a point on a 2D surface on the screen.

4. As completion of this task, in the main program part of your script, write some testing code for drawing at least two `Point`s on the screen. As for previous tasks, from your program you have to print some explanation, i.e. that you are about to draw a `Point`, etc.

Homework 12 Submission Instructions:

1. Your submission has to contain the files as listed above:
   `hw12-transcript.py`, and
   `hw12-point.py`.
   1. At the top of your files, include "*A201 / Fall 2017*" (or "*A597 / Fall 2017*"), "*Homework 12*", your *full name* and your *IU username* (i.e. your IU account) *in* the homework text. Note: in your Python files (file extension .py), kindly include this information at the top of the file, within Python `# comments`.
   2. Make sure that your plain-text documents (including all your `.py` files) are all clearly readable by anyone; 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.

2. Turn in your Homework 12 files by 12:59PM on Thursday, November 09 2017, on the IU Canvas *A201/A597 Fall 2017* page.
3. When you turn in your homework on IU Canvas, it is your responsibility to verify that your files have been uploaded, and that their content is visible on the IU Canvas server.
4. Please do not omit your IU username from the content.

Last updated: November 08, 2017
mitja@indiana·edu