CS1110 Spring 2015

Assignment 3: Due Thursday Feb 26 at 11pm

You must work either on your own or with one partner. If you work with a partner, you and your partner must first register as a group in CMS and then submit your work as a group.

You may discuss background issues and general solution strategies with others, but the programs you submit must be the work of just you (and your partner). We assume that you are thoroughly familiar with the discussion of academic integrity that is on the course website. Any doubts that you have about "crossing the line" should be discussed with a member of the teaching staff before the deadline.

Objectives. String Methods, functions, iteration with `for`. The assignment is based on Lectures 6, 7, and 8 and Lab 4.

1 Overview

There are two problems and two modules to submit: `Password.py` and `Spiral.py`. The first problem is rich in for-loop iteration over strings and involves computing the "strength" of a password. The second problem is graphical and also requires a facility with for-loops. The idea is to draw a "discrete" logarithmic spiral that is defined by a structured sequence of line segments that are joined end-to-end.

Start by setting up a folder called `A3`. Into this folder put the modules `Password.py` (a template) and `simpleGraphicsE.py`. Both can be downloaded from the Assignments webpage. The latter module is an extended and improved version of `simpleGraphics.py` that was used in earlier assignments and labs. In particular, it includes a procedure that can be used to draw a colored line segment. House the modules that you are to develop (`Password.py` and `Spiral.py`) in your `A3` folder and make sure that it is the CURRENT WORKING DIRECTORY whenever you are working on the assignment.

2 Password Strength

This problem is about computing a number that measures the "strength" of a password. For us, a password is a string made up of characters from these constants that are part of the `string` module:

```
string.digits:      '0123456789'
string.letters:     'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
string.punctuation: '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

To be clear on jargon, we say a character is a *digit* if it is a character in `string.digits`, a *letter* if it is a character in `string.letters`, and a *special character* if it is a character in `string.punctuation`. The strength of a password is the sum of bonus scores minus the sum of the deduction scores. We describe the bonuses and deductions in the next two subsections.

The ideas behind this assignment are inspired by the website `http://www.passwordmeter.com` and you may want to play around with what you see there. We have modified some of their formulas to make the computations doable given where we are in the course and to produce (what we think) is a better metric for assessing password strength. For example, `passwordmeter.com` regards `'A!!!!!!!!!!!!!!!!!!!!!!!!!'` as a very strong password.

Word of warning before going one step further. **KEEP YOUR ACTIVE PASSWORDS OUT OF THIS ASSIGNMENT.** Do not debug your code using them as examples. Do not pass them along to `passwordmeter.com`.

2.1 The Bonus Scores

The bonus scores reward certain "good behaviors" associated with password design. Here they are:

B1. *Is the password long but not too long?*

Let N equal the number of characters in the password. The B1 score is the smaller of the two numbers $3N$ and 45. For example, the B1 score for `'ABCDEFGHIJKLMNOPQRSTUVWXYZ'` is 45. It is good to have a long password but there is no extra reward if $N > 15$.

B2. *What is the mix of upper and lower case letters?*

Let N be the number of letters in the password, L the number of lower case letters in the password, and U the number of upper case letters in the password. The B2 score is zero if there are no letters in the password. Otherwise, it is given by the largest integer less than or equal to $40(1-(L/N))(1-(U/N))$. Thus, the B2 score for `'ABcdef123'` is the largest integer less than or equal to 40(1-(4/6))(1-(2/6)) = 80/9, i.e., 8. Remember, "case" only applies to the characters in `string.letter`. And take a look at `math.floor`.

B3. *How many digit characters are there in the password?*

Let N be the number of digits in the password. The B3 score equals $4N$.

B4. *How many special characters are there in the password?*

Let N be the number of special characters in the password. The B4 score equals $6N$.

B5. *How good is the middle of the password?*

Excluding the zeroth and last character in the password, let N be the number of special characters or digits in the password. The B5 score equals $3N$. Thus, the B5 score for the length-8 string `'01*ab82@'` is $3 \times 4 = 12$.

B6. *Overall Mix of Characters*

If the password contains a lower case letter, an upper case letter, a special character, and a digit, then the B6 score is 10. Otherwise, the B6 score is zero.

2.2 The Deduction Scores

The deductions penalize bad behavior when designing a password. Here they are:

D1. *Just letters?*

Let N be the length of the password. If it is comprised only of letters, then the D1 deduction is N. Otherwise, the D1 deduction is zero.

D2. *Just digits?*

Let N be the length of the password. If it is comprised only of digits, then the D2 deduction is N. Otherwise the D2 deduction is zero.

D3. *Repeat characters?*

Let N be the length of the password and M the number of characters in the password that appear exactly once. The D3 deduction is $3(N - M)$ Thus the D3 deduction for `'abcadca'` is 15.

D4. *Consecutive Look-a-Likes?*

A two-character substring of the password is a consecutive look-a-like if consists of either (i) two upper case letters, or (ii) two lower case letters, or (iii) two digits, or (iv) two special characters. The D4 deduction is two times the total number of consecutive look-a-likes. Thus, the D4 score for `'abc4(*Dvx'` is 8 because there are four consecutive look-a-like substrings: `'ab'`, `'bc'`, `'(*'`, and `'vx'`.

D5. *Row-1 Triplets?*

If a length-3 substring of the password is also a substring of `'1234567890'`, then it is called a row-1 triplet. The D5 deduction is three times the total number of row-1 triplets. Thus the D5 score for `'01245672345x7890'` is 18 because there are six row-1 triplets: `'456'`, `'567'`, `'234'`, `'345'`, `'789'`, and `'890'`. The idea behind this deduction is that the lazy password designer will type in three adjacent digits from the top row of the keyboard just to get some digits into the string.

D6. *Row-2 Triplets?*

If a length-3 substring of the password is also a substring of `'qwertyuiop'`, then it is called a row-2 triplet. The D6 deduction is three times the total number of row-2 triplets.

D7. *Row-3 Triplets?*

If a length-3 substring of the password is also a substring of `'asdfghjkl'`, then it is called a row-3 triplet. The D7 deduction is three times the total number of row-3 triplets.

D8. *Row-4 Triplets?*

If a length-3 substring of the password is also a substring of `'zxcvbnm'`, then it is called a row-4 triplet. The D8 deduction is three times the total number of row-4 triplets.

2.3 Computing Password Strength

The recipe for password strength σ is the sum of the bonuses minus the sum of the deductions:

$$\sigma \;=\; (B1 + B2 + B3 + B4 + B5 + B6) \;-\; (D1 + D2 + D3 + D4 + D5 + D6 + D7 + D8)$$

You are to develop a module `Password` that can be used to compute password strength. It must have these features:

Import

You will need to import the `math` module and `string` module. Regarding the former, you will need the `floor` function in your implementation of the B2 score.

Global Variables

It should have four global variables `LOWER`, `UPPER`, `DIGITS`, and `SPECIAL` that respectively house the string of all lower case letters, the string of all the upper case letters, the string of all the digits, and string of all the special characters.

The Application Script

It should use `raw_input` to solicit a string. If the string is not a valid password then it should print a message saying just that. (A string is not a valid password if it includes a character that is is not a character in `string.digits`, `string.letters`, or `string.punctuation`. For example, a string that includes a blank is not a valid password.) If it is a valid password, then it should display the password (on a single line), the password strength (on a single line), the six bonus scores (on a single line), and the eight deduction scores (on a single line). Something like this is good enough:

```
  Password:  sjY789^abcFG
  Strength:  70
-----------------------------------------
   Bonuses:  36   9  12   6  12  10
Deductions:   0   0   0  12   3   0   0   0
```

Function Definitions

There must be one function for each bonus score computation and one function for each deduction score computation. For example,

```
def B1(s):
    """ Returns an int that is a score associated
    with password length.

    PreC: s is a valid password string."""

    return 3*min(len(s),15)
```

Each of these functions should return an `int` value.

To simplify the organization, you must also include a pair of "helper" functions. Here is one:

```
def HowMany(s1,s2):
    """ Returns an int that is the number of
    characters in S1 that are also characters in S2.

    PreC: s1 and s2 are strings. """
```

Having this function "on tap" makes it very easy to implement a number of the bonus score and deduction score functions.

The other helper function is for you to design upon consideration of D5, D6, D7, and D8. Notice that these four deduction score computations are identical except for the "keyboard row strings." Thus, it makes sense to have a helper function of the form `HowManyTriplets(s1,s2)` that can be used to compute the number of length-3 substrings in `s1` that are also substrings of `s2`.

Lastly, you are to include in `Password.py` a function of the form `PWS(s)`. It should return an `int` that is the password strength of the valid password string `s`.

2.4 A Given Template

We provide a skeleton version of `Password.py` that introduces the notion of a *function stub*. Here is an example:

```
def B2(s):
    pass
```

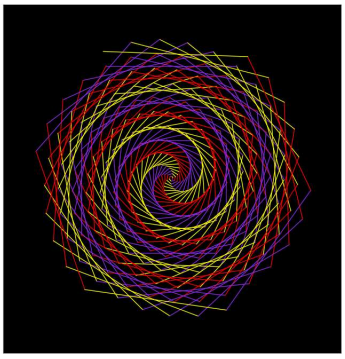
Notice that we have provided stubs for all the functions that you are to write. Think of a stub as a placeholder. The `pass` statement acts like a do-nothing function body. For `B2`, start by writing its specification. Remove the `pass` statement and develop the function body. Augment the Application Script so that it also prints out `B2(s)`. That way you can check for `B2`'s correctness. When you are done with `B2` move on to `B3` (or any of the other stubs).

2.5 CMS and Grading

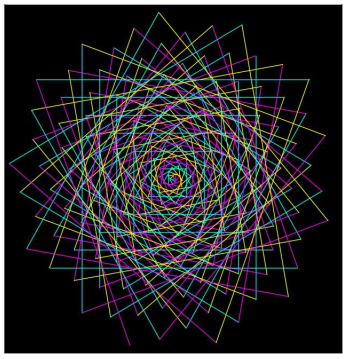
Submit you implementation of `Password.py` to CMS. You will be graded on the effective use of global variables (2 points), the Application Script (3 points) the correctness of the six bonus score functions (5 points) the correctness of the eight deduction score functions (8 points), the correctness of the two helper functions (6 points), the effective use of the two helper functions (6 points), the correctness of `PWS` (2 points) and style (8 points). **No credit for solutions that use lists or tuples.**

3 Spiral

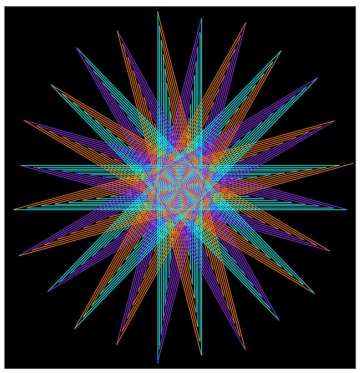
A *discrete logarithnic spiral* is defined by two numbers N and t. The number of edges is given by N and the "turn angle" is given by t. Here are some interesting examples:



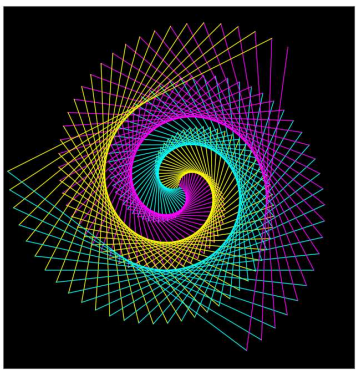
$N = 300,\ t = 62$



$N = 200,\ t = 110$



$N = 300,\ t = 165$

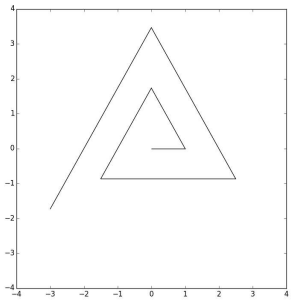


$N = 200,\ t = 118$

The spiral "starts" at (0,0) and consists of N edges that are connected end-to-end. Indexing from 0, the kth edge has length $k+1$ and "compass heading" equal to kt degrees. This means that if (x_c, y_c) is the "starting endpoint" of the kth edge, then

$$\left(x_c + (k+1)\cos\left(\frac{kt\pi}{180}\right), y_c + (k+1)\sin\left(\frac{kt\pi}{180}\right)\right)$$

is the "ending endpoint" of the kth edge. Another way to think about it is to imagine walking along the spiral. When you come to the end of an edge, you turn t degrees to your left before continuing along the next edge. Consider the example with $N = 6$ and $t = 120°$:



Here are the details about its six edges:

Edge	Length	Heading
0	1	$0 \cdot 120$
1	2	$1 \cdot 120$
2	3	$2 \cdot 120$
3	4	$3 \cdot 120$
4	5	$4 \cdot 120$
5	6	$5 \cdot 120$

3.1 Drawing a Spiral

You are to develop a module `Spiral.py` that can be used to display discrete logarithmic spirals. It should import the module `simpleGraphicsE` which includes a procedure

```
DrawLineSeg(a,b,L,theta,linecolor=BLACK)
```

that can be used to draw a colored line segment with starting endpoint (a, b), length L, and heading θ (in degrees).

Begin by implementing a function

```
def DrawSpiral(N,t,c1,c2,c3):
    """ Draws a spiral with N edges, turn angle, and
    colors c1, c2, and c3.

    Beginning with edge 0, every third edge has color c1.
    Beginning with edge 1, every third edge has color c2.
    Beginning with edge 2, every third edge has color c3.

    PreC: N is a positive int, t is float that specifies the turn angle
    in degrees,  and c1, c2, and c3 are rgb arrays. """
```

The module `Spiral.py` should also include an Application Script that can be used to check out your implementation of `DrawSpiral`:

```
MakeWindow(350,labels=False,bgcolor=BLACK)
DrawSpiral(300,62,PURPLE,RED,YELLOW)
ShowWindow()
```

3.2 The Radius of a Spiral

If we are to use the `simpleGraphicsE` environment to display a particular logarithmic spiral, then a problem arises: How big should we make the display window? In the above example, we happen to know that the spiral with $N = 300$ and $t = 62$ fits nicely inside `MakeWindow(350)`. One way to properly size the display window if to compute the *radius* of the spiral. We define the radius of a spiral that has N edges to be the largest of the numbers $d_0, \ldots, d_{N-1}$ where d_k is the distance from the origin to the ending endpoint of the kth edge. Write a function `SpiralRadius(N,t)` that computes this value and include it in `Spiral.py`. Augment the Application Script so that it draws a second spiral of your own design:

```
# First Spiral
MakeWindow(350,labels=False,bgcolor=BLACK)
DrawSpiral(300,62,PURPLE,RED,YELLOW)
# Second Spiral
N =
t =
r = SpiralRadius(N,t)
MakeWindow(r,labels=False,bgcolor=BLACK)
c1 =
c2 =
c3 =
DrawSpiral(N,t,c1,c2,c3)
ShowWindow()
```

3.3 CMS and Grading

Submit `Spiral.py` to CMS. Ten points for `DrawSpiral`, five points for the Application Script, five points for `SpiralRadius` and five points for style. **No credit for solutions that use lists or tuples.**