Assignment 2

Name: ________________________________ Date: ____________________ Block: ______

You must work on your own. You may discuss background issues and general solution strategies with others, but the programs you submit to the **A2 Dropbox** must be the work of just you. I assume that you are thoroughly familiar with the discussion of academic integrity that is on the course website and in the school handbook. Any doubts that you have about "crossing the line" should be discussed with me before the deadline.

0 Objectives

More practice with strings and conditional execution. Using `import`. Defining a simple fruitful function and testing it for correctness. Using graphics procedures to produce a design. Representing color with an `rgb` array.

1 Overview

There are two problems and two modules to submit: `GenerateWeather.py` and `ShowSun.py`. Both give you practice defining and using functions. The first problem is numerical and the second is graphical and makes use of `simpleGraphics`.

For the module `GenerateWeather.py` you will write a function `WindChill` that has two string arguments. They specify temperature (in either Celsius or Fahrenheit) and wind speed (in either mph or kph). The function then computes and returns the windchill temperature (in either Celsius or Fahrenheit). You also write an Application Script that is part of the `GenerateWeather.py` module. It will test your implementation of `WindChill`.

The module `ShowSun.py` has you putting together a function `DrawSun` that draws a "sun" and an Application Script that uses `DrawSun` to produce a nested-sun design.

Start by setting up a folder called `A2` onto your Desktop. Into this folder, put the modules
- `Weather.py`
- `GenerateWeather.py`
- `simpleGraphics.py`
- `ShowSun.py`

which can be downloaded from the Assignments webpage. In the following, when we talk about interactions that involve the command shell and Python interactive mode and related matters, IT IS ASSUMED that `A2` is the CURRENT WORKING DIRECTORY.

2 Windchill

2.1 Background

The commands

```
>>> import Weather
>>> help(Weather)
```

will bring up a synopsis of five functions:

```
FUNCTIONS

      WCF(T, W)
            Returns the WindChill value in Fahrenheit as float.

            Precondition: T and W are numbers that represent
            temperature in Fahrenheit and wind speed in miles per
            hour respectively. Must have T<=50 and W>=3.

      to_C(x)
            Returns the Celsius equivalent of x as float.

            Precondition: x is a number that represents a
            Fahrenheit temperature.

      to_F(x)
            Returns the Fahrenheit equivalent of x.

            Precondition: x is a number that represents a Celsius
            temperature.

      to_K(x)
            Returns the kilometer equivalent of x as float.

            Precondition: x is a number that represents a distance in miles.

      to_M(x)
            Returns the mile equivalent of x as float.

            Precondition: x is a number that represents a distance in
            kilometers.
```

(Note: sometimes “>>>help” does different things in Mac and Linux environments.) Because of the import command, we have access to these functions and because of the documentation we know how to use them. For example, here is a sequence that reports the windchill factor in Celsius assuming that the temperature is -3C and the wind is 20kph:

```
>>> Temp = -3
>>> Temp = Weather.to_F(Temp)
>>> Wind = 20
>>> Wind = Weather.to_M(Wind)
>>> TheWindChill = Weather.WCF(Temp,Wind)
>>> TheWindChill = Weather.to_C(TheWindChill)
>>> print TheWindChill
-9.00504160055
```

Alternatively, we could write a function to handle this kind of metric-based windchill computation. Here is a module that contains a function that does just that. It also has an Application Script that can be used to check it out:

```
# Metric.py
""" Contains a function for metric-based windchill and an Application Script
for testing."""

import Weather

def WCC(T,W):
      """ returns the Windhill value in Celsius

            Precondition: T and W are numbers that represent temperature in
            Celsius and wind speed in kilometers per hour respectively. Must
            have T <= 10 and W >= 4.8. """

            Temp = Weather.to_F(T)
            Wind = Weather.to_M(W)
            TheWindChill = Weather.WCF(Temp,Wind)
            TheWindChill = Weather.to_C(TheWindChill)
            return TheWindChill

#Application Script
if __name__ == '__main__':
      """ Checks out WindChill on an example """
      x = WCC(-3,20)
      print x
```

Back in the command shell we can run the Application Script as follows:

```
      > python Metric.py
      > -9.00504160055
```

As expected, this is the same result that we obtained in interactive mode. Notice that in the Application Script we say `WCC(-3,20)` and not `Metric.WCC(-3,20)`. Side note. We could implement `WCC` with a 1-line function body:

```
      return Weather.to_C(Weather.WCF(Weather.to_F(T),Weather.to_M(W)))
```

But this kind of cryptic “showing off ” can obscure what’s going on and it invites screw-ups. It’s reckless coding: “Look Mom, here I am on a busy street riding my bike with no hands.”

2.2 Details About the Function You Are to Write

You are to write a module `GenerateWeather.py` that will house a function `WindChill(T,W)` that can take either metric or English input. It will also include an Application Script that can be used for testing. The module `GenerateWeather.py` that was supplied is a template. Here are the details about the function `WindChill(T,W)`:

The Input Parameters

The arguments `T` and `W` are strings that specify the temperature and wind speed. This table indicates the legal options for the argument `T`:

Input String	What it Means
`'23F'`	twenty three degrees Fahrenheit
`'-10F'`	minus ten degrees Fahrenheit
`'0C'`	zero degrees Celsius
`'18C'`	eighteen degrees Celsius
`'16'`	sixteen degrees Fahrenheit

Formally, `T` is a valid *temperature string* if it encodes an integer or a string that encodes an integer concatenated with an `'F'` or a `'C'`. Note that `T` encodes a Fahrenheit temperature if there is no `'F'` or `'C'` at the end.

Likewise, this table indicates the legal options for the argument `W`:

Input String	What it Means
`'10mph'`	ten miles-per-hour
`'10kph'`	ten kilometers-per-hour
`'10'`	ten miles-per-hour

Formally, `W` is a valid *wind speed* string if it encodes a nonnegative integer or a nonnegative integer concatenated with either `'mph'` or `'kph'`. Note that `W` encodes a miles-per-hour windspeed if there is no `'mph'` or `'kph'` at the end.

Your implementation of `WindChill` is not required to gracefully handle illegal input. `WindChill(20F,30)`, `WindChill('Brrr','Windy')`, `WindChill('I am staying home')` each violate the precondition that says `T` is a temperature string as defined above and `W` is a wind speed string as defined above. It is to be expected that Python will grind to a halt with an error message in these cases. Later in the course, we will learn how design functions that aren't this rude to the careless user!

The Value to be Returned

`WindChill` returns the windchill temperature in Fahrenheit if `T` is in Fahrenheit. If `T` is in Celsius, then `WindChill` returns the windchill temperature in Celsius. However, if the air temperature is greater than 50F or if the wind speed is less than 3mph, then the windchill temperature equals the air temperature. In all cases, the windchill value that is to be returned should be rounded to the nearest integer. Thus, if the computed windchill temperature is -5.3F, then -5 should be returned. If the computed windchill temperature is 16.7C, then 17 should be returned. The built-in function `round` can be used for rounding. In all cases, the value returned should have type float.

The Body

Here are some hints about developing the body of `WindChill`.

1. Decode the temperature string `T` obtaining a float variable that houses the temperature in Fahrenheit. You may have to use `Weather.to_F`.

2. Decode the input windspeed string `W` obtaining a float variable that houses the windspeed in mph. You may have to use `Weather.to_M`.

3. Use `Weather.WCF` to compute the Fahrenheit wind chill temperature. Of course, if the temperature is greater than 50F or the wind speed is less than 3mph, then the Fahrenheit windchill temperature is just the Fahrenheit air temperature.

4. If `T` specifies a Celsius temperature, then you have to convert the computed windchill to Celsius.

5. Regardless of whether the final windchill is in Fahrenheit or Celsius, round to the nearest integer and return the result as a float.

To facilitate debugging, you should insert appropriate `print` statements throughout (1)-(5) to ensure that your code is doing the right thing. Make sure these debugging print statements are removed in your final submission to the A2 Dropbox or points will be deducted.

You are required to make effective use of the functions in `Weather.py`. You will lose points if you say something like `y = 1.8*x+32` instead of `y = Weather.to F(x)`. Of course it is tempting to do such an easy calculation without using a function, but I am forcing certain issues in this assignment.

2.3 The Application Script

The act of testing a function is the act of inspiring confidence that the function performs as advertised. The module `GenerateWeather` is to include an Application Script that checks out `WindChill` on a set of representative test problems. For each test problem, it is to display the value produced by `WindChill` and the “true” value.

For example, earlier we established via interactive mode that the windchill is -9.00504160055 if the temperature is -3C and the wind is 20kph. We can regard -9.00504160055 as the “true” value because it was obtained using correct software: `Weather.py`. If `WindChill('-3C','20kph')` returns -9, then you can be reasonably confident that `WindChill(T,W)` will work with any other “C-kph” pair (provided the `T` is not too warm and `W` is not too calm.) But there are many other situations to check out:

T	W
ends with 'F'	default
default	ends with 'kph'
ends with 'C'	ends with 'mph'
'85F'	ends with 'kph'
ends with 'C'	'2kph'
:	:

Design 10 different “representative” test cases and compute the “actual” windchill using the `Weather` module in interactive mode as I illustrated above. The Application Script should produce ten lines of output. Each should report the input string `T`, the input string `W`, the actual windchill, and the value returned by `WindChill(T,W)`. A handy print function has been supplied for doing this. It is included in the template `GenerateWeather.py`.

Make sure your 10 representative cases are different. `WindChill('-10F','40')` and `WindChill('5F','20')` do NOT shed light on two different representative cases. They both are instances of the case “`T` is a number string with an `F` and `W` is a default mph windspeed string.”

I will check out your implementation on a zillion representative cases so even though you are formally just checking ten different situations, be sure that your `WindChill` can handle all valid input combinations. The theme of testing functions will continue throughout the course. You will learn about more systematic approaches later on. This is just a preliminary exposure to the idea.

2.4 Submission to A2 Dropbox

Submit your finished implementation of `GenerateWeather` to the A2 Dropbox. Your score will depend on style, the number of cases from my testing program that your `WindChill` can handle, and the design of your ten test cases in the Application Script. Regarding style, review how modules and functions are to be documented using structured `docstrings`. Follow the rules! You will lose points for substandard documentation. Is your documentation “good enough’? Type `help(GenerateWeather)` in Python interactive mode. Can someone use your `WindChill` function based on what they see?

3 The Sun

Now for a really “hot” problem that involves using the procedures in `simpleGraphics.py`.

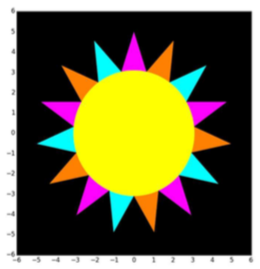
3.1 Background

Playing with the Lecture 14 demos will get you ready for this part of the assignment. Take a look at the template module `ShowSun.py`. Using any editor (Komodo, Idle, etc.), enter your name and date at the top.

You have to complete this module by doing two things. You have to write the function body for `DrawSun` and you have to add code to the Application Script so that it displays a specified nesting design.

3.2 Drawing a Single Sun

The function `DrawSun(x,y,r,c1,c2,c3)` that you are to develop is to produce a sun design like this:



This particular sun is the result of the call `DrawSun(0,0,5,MAGENTA,ORANGE,CYAN)`. Here are some facts about this graphic. A sun has a disk and 15 uniformly spaces “rays”. We’ll index these starting at the top and going around clockwise: Ray 1, Ray 2, …, Ray 15. A sun has a center (*x, y*) and a radius *r*. The radius is the distance from the center to the tip of a ray. The "disk part" of a sun is yellow and its radius is αr where α = .62. Every third ray is colored the same. (Magenta, orange, and cyan in the displayed example above.) This informal specification of `DrawSun(x,y,r,c1,c2,c3)` can be made more precise:

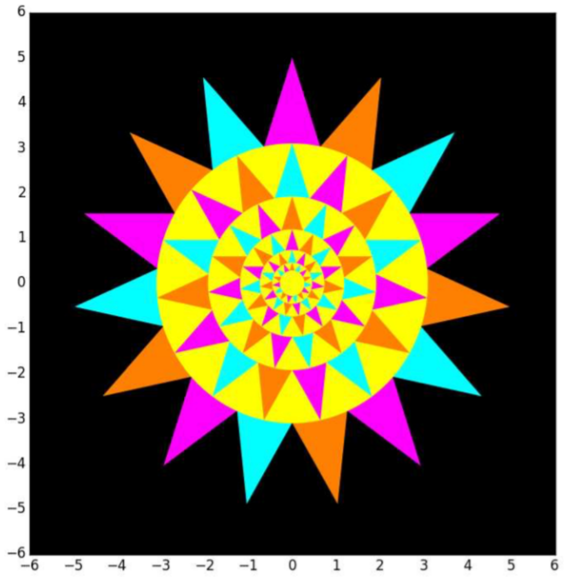
- `x` and `y` are numbers that specify the center (*x, y*) of the sun
- `r` is a positive number that specifies the radius of the sun
- `c1`, `c2`, and `c3` are `rgb` arrays that specify the three "ray colors." In particular, `c1` specifies the color of Rays 1, 4, 7, 10, and 13. `c2` specifies the color of Rays 2, 5, 8, 11, and 14. `c3` specifies the color of Rays 3, 6, 9, 12, and 15.

The implementation of `DrawSun` will require four calls to functions in `simpleGraphics.py`.

You will need three `DrawStars` (with suitable coloring and rotation) and a wrap-up `DrawDisk`.

3.3 The Nested Stars Design

The Application Script must also display a nesting like this:



Look carefully and you will see six stars in the design, each have the same center and the same ray colors. The suns are nested, each one "just fitting" inside of the disk of its larger neighbor. To specify the rules for the nesting we need to index the Suns from largest to smallest, say S_1, S_2, S_3, S_4, S_5, and S_6. Here are the rules:

> *The Radius Rule*. The radius of S_k is α times the radius of S_{k-1} where α = .62. i.e. if *r* is the radius of S_1, then the radius of S_2 is .62*r*.

The Color Rule. The ray colors in S_k are the same as the ray colors in S_{k-1} except they are shifted clockwise one "notch." In the displayed example, the color sequence for S_1 (starting from the noon position) is

```
MOCMOCMOCMOCMOC
```

while for S_2 it is

```
CMOCMOCMOCMOCMO
```

Here M = Magenta, O = orange, and C = cyan. Complete the Application Script so that it draws the nesting with 6 suitable calls to `DrawSun`.

You are free to use either the built-in colors or you can "make up" your own `rgb` arrays. (If you Google "rgb colors" you will discover that there are several websites that display tables of colors and their `rgb` representation. So if you go this route, pick an interesting triplet of colors.)

To summarize, when you run `ShowSun.py` from the command shell, two figures should open up. In Figure 1 you should see a single sun. This will affirm that your implementation of `DrawSun` is correct. Figure 2 will display the nested stars. If it is correct then you successfully used `DrawSun`. Submit your implementation of `ShowSun.py` to the A2 Dropbox.