GEOG 582

2019/02/27 (Due on 2019/03/06 by 23:59:59, 0 point after due)

Assignment 5: Modules & Midterm Project

Instructions:

1. Create a python file using PyCharm.
   - The file name should be, a05_yourlastname.py (e.g., a05_nara.py)
2. Write your name, RedID, date using comments on the top the python file.
3. Whenever necessary, use comments to annotate your codes as much as you can. Comments will be evaluated. Before writing the answer for a specific question, comment the question number.

   Example)

   ```
   #Q1
    Your python script here
   ```

4. Answer the following **questions**.
   - Try NOT to directly copy and paste python syntaxes from lecture slides or other materials!
   - **Use PyCharm debugging tools!**
5. Save your file and submit it via Blackboard.

Questions: Import Modules

1. Import three built-in modules (e.g., random, math, statistics).
Here you can find python standard (built-in) libraries:

https://docs.python.org/3/library/

2. Write a Python program using the three modules imported in Q1 (e.g., create random 100 numbers, calculate distance between 2-dimensional points, calculate summary statistics on a list). Use your creativity.

3. Create two python scripts, one as a main module (i.e., execute it as a main program) and the other for a sub-module. Define one simple class and one simple function in the sub-module. In the main module, import the sub-module, create one instance of the sub-module class and call the function defined in the sub-module.

GEOG 582

2019/02/27 (Due on 2019/03/06 by 23:59:59, 0 point after due)

Questions: Midterm Project
Assignment 4 (Use codes from Assignment 4!)

1. Define a class **Point**.

a) The **Point** class has attributes to store a coordinate of a 3D point (x, y, z).
If z==0, it is considered as a 2D point.
When initialize the class object, provide a default value 0 for the z-coordinate.
It also has an attribute to store the number of dimension.

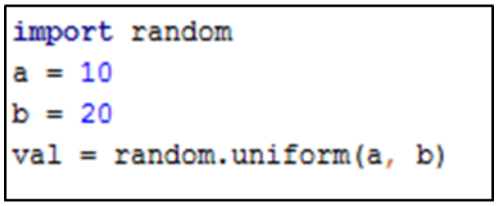
b) The **Point** class has a method ***print_coordinate()*** that prints out its point coordinates.

c) The **Point** class has a method ***calc_distance()*** that calculates the distance between itself (i.e., an instance of **Point**) and another **Point** class object (i.e., another instance of **Point**).
The method returns the distance value in 2D or 3D depending on the number of dimension of 2 points. If both points are 2D then 2D distance, otherwise 3D distance.
The ***calc_distance()*** method also checks if the input point is an instance of **Point** by using a built-in function, ***isinstance()***.

https://docs.python.org/3/library/functions.html#isinstance

d) All **Point** class methods should handle exceptions properly.

2. The following syntax returns a random floating-point value, N,
where a <= N <= b and b <= N <= a when b < a.



```
import random
a = 10
b = 20
val = random.uniform(a, b)
```

Use this syntax and define a function that returns a list of **Point** Class objects with random coordinate values.
This function should take 4 arguments, ***num_point***, ***dimension***, ***lower_bound***, and ***upper_bound***.

3. Generate 100 Point class instances. The coordinates of those 100 points should be floating values between 0 and 100 in the 2-dimensional space.

4. Given a 2D Point Class instance with x=50 and y=50, find the closest point from the list of points created in Q5. Write a program that is based on the following pseudo code.

(Pseudo Code)

```
Create a 2D point class object with x=50 and y=50

Create an empty list to store distance values

Iterate through the point list object

    Calculate a distance between 2 points

    Append the value to the list

Find the minimum distance value in the list and get its list index value

Using the index value obtained from the above, get and print coordinates of a point that is the
closest to the point (50, 50)
```

5. Modify the **Point** Class by adding a property called ***clust_id*** and assign a default value of -1. Modify the ***print_coords()*** method so that the output also prints out the value of ***clust_id***.

GEOG 582

2019/02/27 (Due on 2019/03/06 by 23:59:59, 0 point after due)

Assignment 5 – Midterm portion starts here!

6. Define a class **MyKmeans**

a) The class constructor initializes the following properties.

k: number of k in k-means cluster.

num_points: number of points, which will be randomly generated.

dimension: the dimension of randomly generated points.

lower_bound: the lower bound of randomly generated point.

upper_bound: the upper bound of randomly generated point.

points: a list to store points (initially an empty list).

centroids: a dictionary that store cluster centroids information. Key is a cluster id, and value is a Point instance.

b) Define the following methods for MyKmeans.

set_parameters: ask users to input parameters including **k**, **num_points**, **dimension**, **lower_bound**, **upper_bound**

generate_points: generate random point instances (use the **function** you have created in the previous exercise.)

initialize_centroids: Use one of the following options.

- Option 1: randomly generate **k** points within the defined boundary and add those to the "centroids" dictionary (i.e., ***self.centroids***).
- Option 2: randomly pick **k** points from ***self.points*** and add those to the "centroid" dictionary (i.e., ***self.centroids***).

  Hint: random.sample()
  https://docs.python.org/3/library/random.html#module-random

assign_random_clust_number: assign a random ***cluster_id*** to each point in ***self.points***. ***cluster_id*** should be between 1 to k.

assign_clust_number: use the following syntax or similar. Understand how it works and insert comments to describe this method.

```
for i in self.points:
    #initialize with maximum integer value
    #'sys.maxint 'is Python 2.x only. For 3.x, use 'sys.maxsize'
    min_dist = sys.maxint

    for j in self.centroids:
        dist = i.calc_distance(self.centroids[j])

        if(min_dist > dist):
            min_dist = dist
            i.clust_id = j
```