NAME ________________________________ **ICA-6**

NetID______________________ @email.arizona.edu

Work with your neighbor.

**1**. Consider the following problem specification:

*Write a program that reads a (possibly empty) file containing only numbers (and whitespace) and prints out the difference between the smallest and largest numbers. An empty input file should generate no output.*

Write the following black box tests for this program:

a. two error cases (you should figure out what constitutes error behaviors given the specification above)

b. two edge cases

c. one regular (normal) case

**2.** The following function checks whether two words are anagrams. What white-box tests would you use to test this code?

```
def count(char, string):
    n = 0
    lc_str = string.lower()
    for i in range(len(string)):
        if char == lc_str[i]:
            n = n+1
    return n

def anagrams(str1, str2):
    for ch in "abcdefghijklmnopqrstuvwxyz":
        return count(ch, str1) != count(ch, str2)
    return True
```

| value for `str1` | value for `str2` |
|---|---|
| | |
| | |

**3.** What white-box tests would you use for the following code?

```
# list_has_negatives(arg_list) returns True if arg_list
# contains any negative numbers, False if it does not.

def list_has_negatives(arg_list):
    for i in range(len(arg_list)):
        has_negatives = arg_list[i] < 0

    return has_negatives
```

| value for `arg_list` |
|---|
| |
| |

geolocation.py (for problem 4)

```
from math import *

RADIUS = 3963.1676  # Earth radius in miles

class GeoLocation:

    def __init__(self, latitude, longitude):
        self._latitude = float(latitude)
        self._longitude = float(longitude)

    def get_latitude(self):
        return self._latitude

    def get_longitude(self):
        return self._longitude

    def __str__(self):
        return "latitude: " + str(self._latitude) + ", longitude: " + str(self._longitude)

    def distance_from(self, other):
        lat1 = radians(self._latitude)
        long1 = radians(self._longitude)
        lat2 = radians(other._latitude)
        long2 = radians(other._longitude)
        # apply the spherical law of cosines with a triangle composed of the
        # two locations and the north pole
        the_cos = sin(lat1) * sin(lat2) + cos(lat1) * cos(lat2) * cos(long1 - long2)
        arc_length = acos(the_cos)
        return arc_length * RADIUS
```

**4.** Use geolocation.py to work on the following tasks

(a) Write a main function that creates and uses `GeoLocation` objects in order to satisfy the specification described below.

***Background:*** *In the TV series Breaking Bad, the Walter White character buried millions of dollars at a particular location in the desert outside of Albuquerque, New Mexico. He then bought a lottery ticket to help him remember that his stash was buried at a latitude of 34 degrees, 59 minutes, 20 seconds and a longitude of -106 degrees, -106 degrees, 36 minutes, 52, seconds. Your program will compute the distance between Walter's stash and the local FBI building and a local studio know as ABQ Studios (where Breaking Bad was filmed). Walter's stash was supposedly buried in the desert, but from this program you'll see that the coordinates they gave on the TV show are really the coordinates of the studio.*

Your program is required to produce the following output:

```
The stash is at latitude: 34.988889, longitude: -106.614444
```

```
ABQ studio is at latitude: 34.989978, longitude: -106.614357
FBI building is at latitude: 35.131281, longitude: -106.61263
Distance in miles between:
    stash/studio is 0.07548768123801672
    stash/fbi    is 9.849836190409732
```

(b) Next, you will write a `PlaceInformation` class that represents a place of interest by its name, its address, its category tag, and its physical location given in terms of latitude and longitude. Since information in this class includes latitude and longitude data, you will use a `GeoLocaton` object to hold that information.

Implement the following attributes:

- `_name`: the name of the place
- `_address`: the address of the place
- `_tag`: a tag for the place that describes its category (restaurant, hotel, etc.)
- `_geolocation`: a `GeoLocation` object holding the latitude and longitude of the place

Implement the following methods:

- `__init__(self, name, address, tag, latitude, longitude)`

- `__str__(self)`

- `get_name()`

- `get_address()`

- `get_tag()`

- `distance_from(self, other)`: returns the distance between the `PlaceInformation` object `self` and the `GeoLocation` object `other`.

Note that the `distance_from()` method must calculate the distance between a `PlaceInformation` object and a `GeoLocation` object. However, there is no need to replicate the code from the `GeoLocation` class that computes the distance based on latitute and longitude.

A `PlaceInformation` object has a `_geolocation` attribute which in turn is a `GeoLocation` object. Use that object's `distance_from()` method to compute the distance.