Abelardo Riojas
ISC 4221C Professor Quaife
Fall 2019. *Python*

This is the output of my code when just the script name is inputted into the command line.
Command: python lab1.py

```
You must input coordinates into the program as follows for it to run.
Example command: python lab1.py 29.1872 'N' 82.1401 'W'

Lake City: 30.1897 'N' 82.6393 'W'
Ocala: 29.1872 'N' 82.1401 'W'
```

This is the output of my code when latitude and longitude are properly entered into the command line along with the script name.
Command: `python lab1.py 29.1872 'N' 82.1401 'W'`

```
Bubble Sort, n = 25
 Unsorted Array: [56, 98, 8, 99, 6, 32, 73, 24, 31, 18, 34, 33, 14, 15, 66,
82, 10, 76, 17, 45, 92, 27, 35, 71, 8]
 Index Array:    [4, 2, 24, 16, 12, 13, 18, 9, 7, 21, 8, 5, 11, 10, 22, 19, 0,
14, 23, 6, 17, 15, 20, 1, 3]
 Sorted Array:   [6, 8, 8, 10, 14, 15, 17, 18, 24, 27, 31, 32, 33, 34, 35, 45,
56, 66, 71, 73, 76, 82, 92, 98, 99]

Selection Sort, n = 25
 Unsorted Array: [2, 29, 83, 58, 23, 77, 8, 65, 56, 23, 1, 81, 53, 15, 30, 68,
68, 88, 49, 34, 25, 88, 94, 63, 14]
 Index Array:    [10, 0, 6, 24, 13, 9, 4, 20, 1, 14, 19, 18, 12, 8, 3, 23, 7,
16, 15, 5, 11, 2, 17, 21, 22]
 Sorted Array:   [1, 2, 8, 14, 15, 23, 23, 25, 29, 30, 34, 49, 53, 56, 58, 63,
65, 68, 68, 77, 81, 83, 88, 88, 94]

Testing for sortedness using sorted():
True
True

Finding closest stores for 29.1872, -82.1401
store#2 Gainesville 33.99 miles
store#6 Orlando 73.72 miles
store#4 Jacksonville 84.47 miles
store#5 Tampa 86.67 miles
store#1 Tallahassee 154.01 miles
store#7 Hialeah 256.16 miles
store#3 Miami 263.78 miles
```

My code explained

```
import random
import math
import sys
try:
      float(sys.argv[1])
except IndexError:
      print("\nYou must input coordinates into the program as follows for it
to run. \nExample command: python lab1.py 29.1872 'N' 82.1401 'W'\n")
      print("Lake City: 30.1897 'N' 82.6393 'W' \nOcala: 29.1872 'N' 82.1401
'W'\n")
      sys.exit()
```

The header of my code imports a couple of packages that will be used and checks if the user put in any command line arguments. If nothing was submitted along with the script name, then the code tells the user what to do and exits.

```
def randListGen(length):
      randlist = [];
      for i in range(length):
            randlist.append(random.randint(1,101))
      return randlist
```

This function generates a random list of length n of integers between 1 and 100 using the random package -- mainly for testing sorting functions.

```
def selectionSort(array, display):
      n = len(array)
      if (display == 1):
            print ('\nSelection Sort, n = ' + str(n) + '\n Unsorted Array: ' +
str(array))
      smallestInd = 0
      indexArray = list(range(n))
      for i in range(n):
            temp = array[smallestInd]
            indexTemp = indexArray[smallestInd]
            array[smallestInd] = array[i-1]
            indexArray[smallestInd] = indexArray[i-1]
            array[i-1] = temp
            indexArray[i-1] = indexTemp
            smallestInd = i
            for j in range(i,n):
                  if (array[smallestInd] > array[j]):
                        smallestInd = j
      return([indexArray, array])
```

This function is the selection sort algorithm. It looks for the smallest number in a list and swaps it with the nth entry in a list to sort it. This process is repeated until the list is sorted. The ‘display’ argument is so it prints out nicely once we get to part 3 of the lab. The index array is also printed out to the user for a better idea of how the sorting happened.

```
def bubbleSort(array):
      n = len(array)
      print('\nBubble Sort, n = ' + str(n) + '\n Unsorted Array: ' +
str(array))

      indexArray = list(range(n))
      flag = 0
      while(flag < n-1):
            flag = 0
            for i in range(n-1):
                  if(array[i] > array[i+1]):
                              temp = array[i]
                              indexTemp = indexArray[i]
                              array[i] = array[i+1]
                              indexArray[i] = indexArray[i+1]
                              array[i+1] = temp
                              indexArray[i+1] = indexTemp
                  else:
                        flag = flag + 1
      return([indexArray, array])
```

This is my bubble sort function. It sweeps through a list swapping numbers that don’t increase in value when compared left to right. This is done until no swaps are needed to sort the list anymore by checking if the flag counter reaches the list length. Flags are accumulated by not swapping a pair of two numbers.

```
bs = bubbleSort(randListGen(25))
print(' Index Array:    ' + str(bs[0]) + '\n Sorted Array:   ' + str(bs[1]))
ss = selectionSort(randListGen(25),1)
print(' Index Array:    ' + str(ss[0]) + '\n Sorted Array:   ' + str(ss[1]))

print('\nTesting for sortedness using sorted():')
print(sorted(bs[1]) == bs[1])
print(sorted(ss[1]) == ss[1])
```

This part of my code checks if two random lists are actually sorted using the built-in `sorted()` command. This is done by printing the result of the boolean operation `sorted(bs[1]) == bs[1])` which will either print true or false depending on if the list is actually sorted.

```
def haversine(lat1, dir_lat1, lon1, dir_lon1, lat2, dir_lat2, lon2, dir_lon2):

    R = 6371000

    phi_1 = math.radians(lat1)
    phi_2 = math.radians(lat2)

    delta_phi = math.radians(lat2 - lat1)
    delta_lambda = math.radians(lon2 - lon1)

    a = math.sin(delta_phi / 2.0) ** 2 + math.cos(phi_1) * math.cos(phi_2) *
math.sin(delta_lambda / 2.0) ** 2

    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))

    meters = R * c
    km = meters / 1000
    return(round(km,3))
```

This is the haversine function that I found online that I had to augment slightly. I changed the function's arguments to include the direction of the latitude and longitudinal arguments with the variables dir_lat and dir_lon. It returns the straight line distance between two points on a sphere.

```
class store:
      storelist = []
      def __init__(self, storeName, city, lat, dir_lat, lon, dir_lon):
            self.storeName = storeName
            self.city = city
            if (dir_lat == 'S'):
                  lat = float(lat) * -1.0
            self.lat = float(lat)
            if (dir_lon == 'W'):
                  lon = float(lon) * -1.0
            self.lon = float(lon)
            self.dir_lat = dir_lat
            self.dir_lon = dir_lon

      @classmethod
      def makeStoreList(cls,storeList):
            cls.storelist.append(store(storeList[0], storeList[1],
storeList[2], storeList[3], storeList[4], storeList[5]))
```

This class makes a lot more sense if you read the following block of code first.

```
with open('stores_location.dat', 'r') as f:
      for line in f:
            store.makeStoreList(line.split())
```

The store class makes a store object from the read-in store_location.dat file using the `split()` command. A store list is then made using the class method `makestorelist()` to make a list of stores whose properties can be accessed as an object, simplifying things immensely.

```
def distance():
      distances = []
      lat = float(sys.argv[1])
      dir_lat = sys.argv[2]
      lon = float(sys.argv[3])
      dir_lon = sys.argv[4]
      if (dir_lon == 'W'):
                  lon = float(lon) * -1.0
      if (dir_lat == 'S'):
                  lat = float(lat) * -1.0
      n = len(store.storelist)
      print('\nFinding closest stores for ' + str(lat) + ', ' +  str(lon))
      for i in range(n):
            distance = haversine(lat, dir_lat, lon, dir_lon,
store.storelist[i].lat, store.storelist[i].dir_lat, store.storelist[i].lon,
store.storelist[i].dir_lon)
            distances.append(distance)
      sortedstores = selectionSort(distances,0)
      index = sortedstores[0]
      j = 0
      for i in index:
            print(str(store.storelist[i].storeName) + ' ' +
str(store.storelist[i].city) + ' ' +  str(round(sortedstores[1][j] * 0.621371,
2)) + ' miles')
            j = j + 1
distance()
```

This final function creates a list, parses the command line arguments, and then calculates the haversine distance between the inputted coordinates and each store in the `store` class store list. This distances are appended to the list and then sorted using `selectionSort()`. Distances are matched up with the correct store name and city (using the index arrays produced by the sorting algorithm) to produce a useful GUI for our user. Finally the `distance()` function is ran.