CSE 101: Introduction to Computational and Algorithmic Thinking

Stony Brook University

Lab Assignment #10

Spring 2018

Assignment Due: April 13, 2018 by 11:59 pm

Assignment Objectives

This lab assignment will give you some practice working with classes and objects.

Getting Started

Visit Piazza and download the "bare bones" file `lab10.py` onto your computer. Open `lab10.py` in PyCharm and fill in the following information at the top:

1. your first and last name as they appear in Blackboard
2. your Net ID (e.g., jsmith)
3. your Stony Brook ID # (e.g., 111999999)
4. the course number (CSE 101)
5. the assignment name and number (Lab #10)

Submit your final `lab10.py` file to Blackboard by the due date and time. Late work will not be graded. Code that crashes and cannot be graded will earn no credit.

Preliminaries

Throughout this lab you will be working with the class given below, which defines the characteristics of a package being shipped in the mail:

```
class Package:
    def __init__(self, sender, recipient, cost=0, distance=0):
        self.sender = sender
        self.recipient = recipient
        self.cost = cost
        self.distance = distance
```

`sender` and `recipient` are strings that specify the cities where the sender and recipient of the package live, respectively. `cost` is the cost to ship the package, and `distance` is the distance that the package traveled.

This class definition is available in the file `package.py`. DO NOT CHANGE THE CONTENTS OF THAT FILE.

Part I: Shipping Packages (20 points)

Complete the function `shipping_cost()`, which takes two arguments, in this order:

1. `packages`: a list of `Package` objects for which we need to compute the costs to ship.
2. `cost_schedule`: a list of 4 integers that represent how much it costs to ship a package various distances:

Value	**Meaning**
`cost_schedule[0]`	the cost to ship a package < 100 miles
`cost_schedule[1]`	the cost to ship a package ≥ 100 miles, but < 300 miles
`cost_schedule[2]`	the cost to ship a package ≥ 300 miles, but < 500 miles
`cost_schedule[3]`	the cost to ship a package ≥ 500 miles

The function has two purposes:

1. It computes and returns the total cost to ship all of the packages given in the `packages` list and returns that total.
2. It computes the cost to ship each package in the `packages` list and updates the `cost` attribute of each `Package` accordingly. For example, suppose we compute that it will cost $15 to ship `packages[i]`. The function will set `packages[i].cost = 15`

You may assume that `packages` always contains at least one `Package` object.

Example:

Function call:

```
shipping_cost([Package("Monmouth", "Appleby",       cost=0, distance=144),
               Package("Larkinge", "Ballachulish",  cost=0, distance=65),
               Package("Malrton",  "Auchtermuchty", cost=0, distance=872),
               Package("Monmouth", "Anghor Thom",   cost=0, distance=937)],
               [10, 16, 37, 49])}
```

Return value: `124`

Updated `packages` list:

```
[Package("Monmouth", "Appleby",       cost=16, distance=144),
 Package("Larkinge", "Ballachulish",  cost=10, distance=65),
 Package("Malrton",  "Auchtermuchty", cost=49, distance=872),
 Package("Monmouth", "Anghor Thom",   cost=49, distance=937)]
```

Part II: Tracking Packages (20 points)

Complete the function `package_tracking()`, which takes three arguments, in this order:

1. `packages_info`: a list of tuples containing two values: the name of the city that a package was sent from and the name of the city that the package was sent to.

2. `locations`: a dictionary that maps a city's name to its 2D coordinate position, stored as a tuple.
3. `cost_schedule`: a list of 4 integers that represent how much it costs to ship a package various distances:

Value	**Meaning**
`cost_schedule[0]`	the cost to ship a package < 100 miles
`cost_schedule[1]`	the cost to ship a package ≥ 100 miles, but < 300 miles
`cost_schedule[2]`	the cost to ship a package ≥ 300 miles, but < 500 miles
`cost_schedule[3]`	the cost to ship a package ≥ 500 miles

The function iterates over the `packages` list and incrementally builds a list of `Package` objects that represent packages being sent with the given senders and recipients. For each new `Package` object that the function creates, the function consults the `location` dictionary to help it compute the distance that each package will travel. The function can then set the `distance` field of each `Package` object accordingly. Next, using the `cost_schedule` list, the function sets the `cost` field of each `Package` object. Finally, the function returns the list of `Package` objects it created.

We will use the *2D Euclidean distance* as the distance between two cities. (This isn't accurate because the Earth is round, but it's OK!) Given the coordinate positions (x_1, y_1) and (x_2, y_2) of two points, the distance between the points is given as $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

Example:

Function call:

```
package_tracking(
     [('Appleby',    'Satbury'),
      ('Northpass',  'Berkton'),
      ('Eanverness', 'Satbury')],

     {'Appleby': (58, 189),         'Berkton': (84, 13),
      'Ballachulish': (12, 149),    'Aerilon': (28, 77),
      'Garennton': (173, 68),       'Malrton': (124, 133),
      'Peltragow': (194, 181),      'Paentmarwy': (191, 151),
      'Eanverness': (134, 50),      'Satbury': (25, 181),
      'Bracklewhyte': (47, 124),    'Larkinge': (8, 157),
      'Burnsley': (71, 5),          'Erith': (2, 181),
      'Monmouth': (160, 13),        'Northpass': (84, 45),
      'Jedborourgh': (87, 163),     'Anghor Thom': (109, 10),
      'Auchtermuchty': (140, 189), 'Murrayfield': (164, 83)},

     [18, 21, 24, 48]
)
```

Returned list:

```
[Package("Appleby",    "Satbury", cost=18, distance=33.95585369269929),
 Package("Northpass",  "Berkton", cost=18, distance=32.0),
 Package("Eanverness", "Satbury", cost=21, distance=170.41713528867922)]
```

Notes:

Consider the `Package` class again for a moment:

```
class Package:
    def __init__(self, sender, recipient, cost=0, distance=0):
        self.sender = sender
        self.recipient = recipient
        self.cost = cost
        self.distance = distance
```

We note that the constructor (`__init__()`) has two *default arguments* for the `cost` and `distance` attributes. In practice this means that we can construct a `Package` object with only two attributes and set the `cost` and `distance` attributes later:

```
new_package = Package('New York', 'Los Angeles')
# ... other code here, possibly ... and then:
new_package.cost = 50
new_package.distance = 3000
```

or we can construct the object with all 4 attributes at once, if we like:

```
new_package = Package('New York', 'Los Angeles', 50, 3000)
```

You can use either approach; both are acceptable.

How to Submit Your Work for Grading

To submit your `.py` file for grading:

1. Login to Blackboard and locate the course account for CSE 101.
2. Click on "Assignments" in the left-hand menu and find the link for this assignment.
3. Click on the link for this assignment.
4. Click the "Browse My Computer" button and locate the `.py` file you wish to submit. Submit only that one `.py` file.
5. Click the "Submit" button to submit your work for grading.

Oops, I messed up and I need to resubmit a file!

No worries! Just follow the above directions again. We will grade only your last submission.