Department of Computer Science and Engineering

Islamic University of Technology (IUT)

A subsidiary organ of OIC

Lab Report: Python/Autograder Tutorial

CSE 4618: Artificial Intelligence

Name	**: Mashrur Ahsan**
Student ID	**: 200042115**
Section	**: 1 (SWE)**
Semester	**: Winter (6th)**
Academic Year	**: 2023-24**
Date of Submission	**: 19/01/2024**
Lab No.	**: 0**

Introduction:

This lab is a mini Linux and a mini Python tutorial. This lab contains instructions on how to set up the right python version for the tasks.

There are three problem statements presented in this lab: “addition.py”, “buyLotsOfFruit.py” and “shopSmart.py”. These are the files that needs to be edited to solve the problem statements. There’s a file named “autograder.py” which will evaluate the correctness of the code that was written.

Problem Statements:

1. Addition

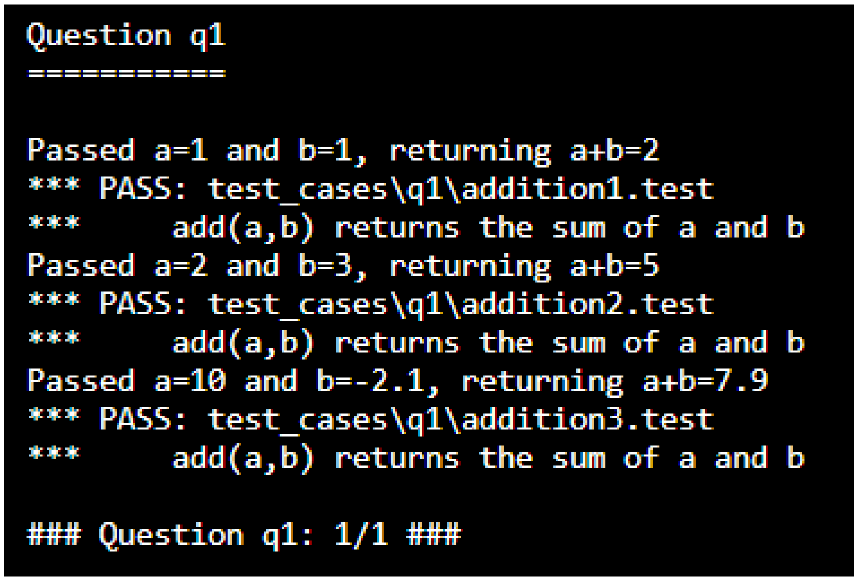
Open `addition.py` and look at the definition of `add`:

```
def add(a, b):
    "Return the sum of a and b"
    "*** YOUR CODE HERE ***"
    return 0
```

The tests called this with `a` and `b` set to different values, but the code always returned zero. Modify this definition to read:

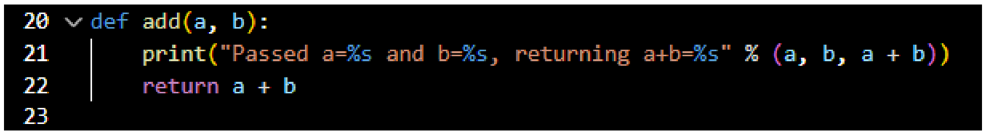
```
def add(a, b):
    "Return the sum of a and b"
    print("Passed a=%s and b=%s, returning a+b=%s" % (a,b,a+b))
    return a+b
```

Now if the autograger.py is run again then it will show us that all of the test cases have passed.



```
Question q1
===========

Passed a=1 and b=1, returning a+b=2
*** PASS: test_cases\q1\addition1.test
***     add(a,b) returns the sum of a and b
Passed a=2 and b=3, returning a+b=5
*** PASS: test_cases\q1\addition2.test
***     add(a,b) returns the sum of a and b
Passed a=10 and b=-2.1, returning a+b=7.9
*** PASS: test_cases\q1\addition3.test
***     add(a,b) returns the sum of a and b

### Question q1: 1/1 ###
```

In the editor the function would like this:



```python
def add(a, b):
    print("Passed a=%s and b=%s, returning a+b=%s" % (a, b, a + b))
    return a + b

```

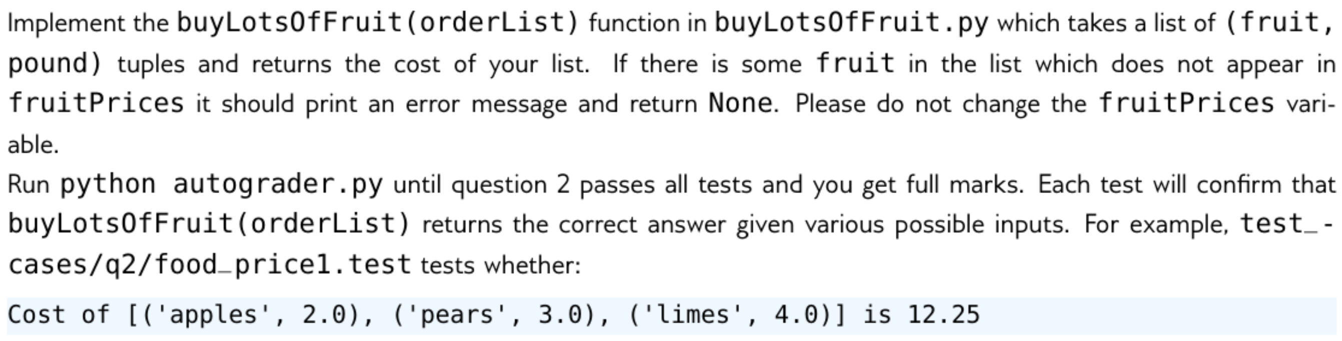
Analysis & Explanation:

Here, “%” being used as a format specifier. Other examples are: %d, %f, %x. Other than that, here we are performing a basic addition operation.

Challenges & Findings:

This problem statement was quite straightforward, no challenges faced while doing it. As for the findings, one could say that we can see a basic structure of a function in python. As we can see from the test cases, this function would also work for float values.

2. buyLotsOfFruit Function:

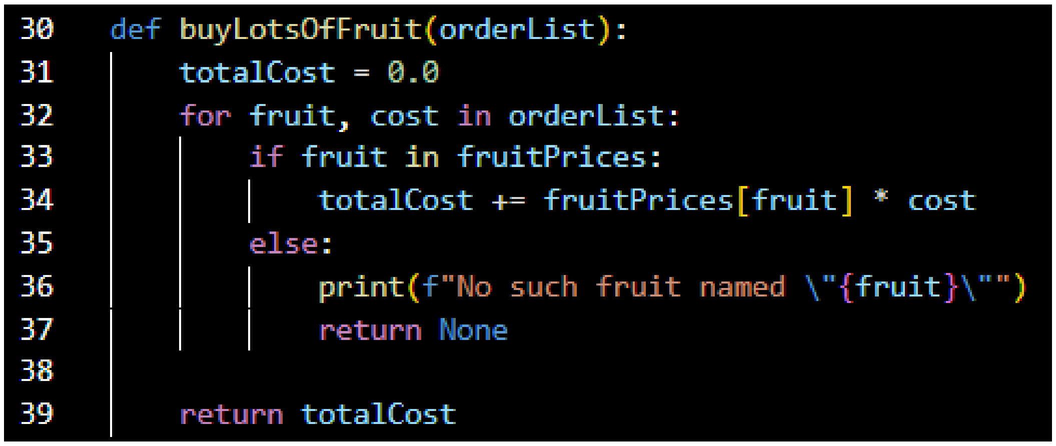


Implement the `buyLotsOfFruit(orderList)` function in `buyLotsOfFruit.py` which takes a list of `(fruit, pound)` tuples and returns the cost of your list. If there is some `fruit` in the list which does not appear in `fruitPrices` it should print an error message and return `None`. Please do not change the `fruitPrices` variable.

Run `python autograder.py` until question 2 passes all tests and you get full marks. Each test will confirm that `buyLotsOfFruit(orderList)` returns the correct answer given various possible inputs. For example, `test_cases/q2/food_price1.test` tests whether:

```
Cost of [('apples', 2.0), ('pears', 3.0), ('limes', 4.0)] is 12.25
```

The solution of the code:



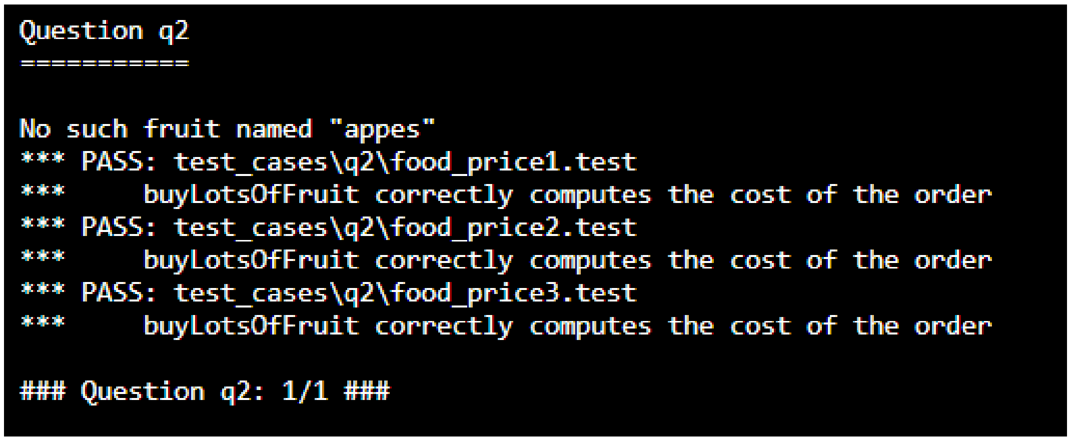
```python
def buyLotsOfFruit(orderList):
    totalCost = 0.0
    for fruit, cost in orderList:
        if fruit in fruitPrices:
            totalCost += fruitPrices[fruit] * cost
        else:
            print(f"No such fruit named \"{fruit}\"")
            return None

    return totalCost
```

Analysis & Explanation:

The task is to implement a function that will calculate total cost from a list of tuples based on a predefined dictionary. Handle missing fruits with error message and return None. Task completion is measured by passing of the test cases.

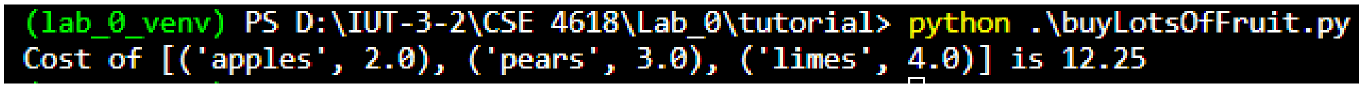
The “buyLotsOfFruit” function iterates through each tuple in the “orderList” and checks whether a “fruit” – the key, is present in the “fruitPrices” dictionary. Then it calculates the total cost if a match is found. On the other hand, if a fruit is not present in the dictionary, it prints out that there is no such fruit and returns None. Otherwise, it returns the total cost.

Now, if we run the autograder.py again. We’ll see that it passes all the test cases:



```
Question q2
===========

No such fruit named "appes"
*** PASS: test_cases\q2\food_price1.test
***     buyLotsOfFruit correctly computes the cost of the order
*** PASS: test_cases\q2\food_price2.test
***     buyLotsOfFruit correctly computes the cost of the order
*** PASS: test_cases\q2\food_price3.test
***     buyLotsOfFruit correctly computes the cost of the order

### Question q2: 1/1 ###
```

If we run the code from the command line then the output would look like this:



```
(lab_0_venv) PS D:\IUT-3-2\CSE 4618\Lab_0\tutorial> python .\buyLotsOfFruit.py
Cost of [('apples', 2.0), ('pears', 3.0), ('limes', 4.0)] is 12.25
```

Challenges:

This problem statement was quite straightforward, so no significant challenges was faced while doing it.

Findings:

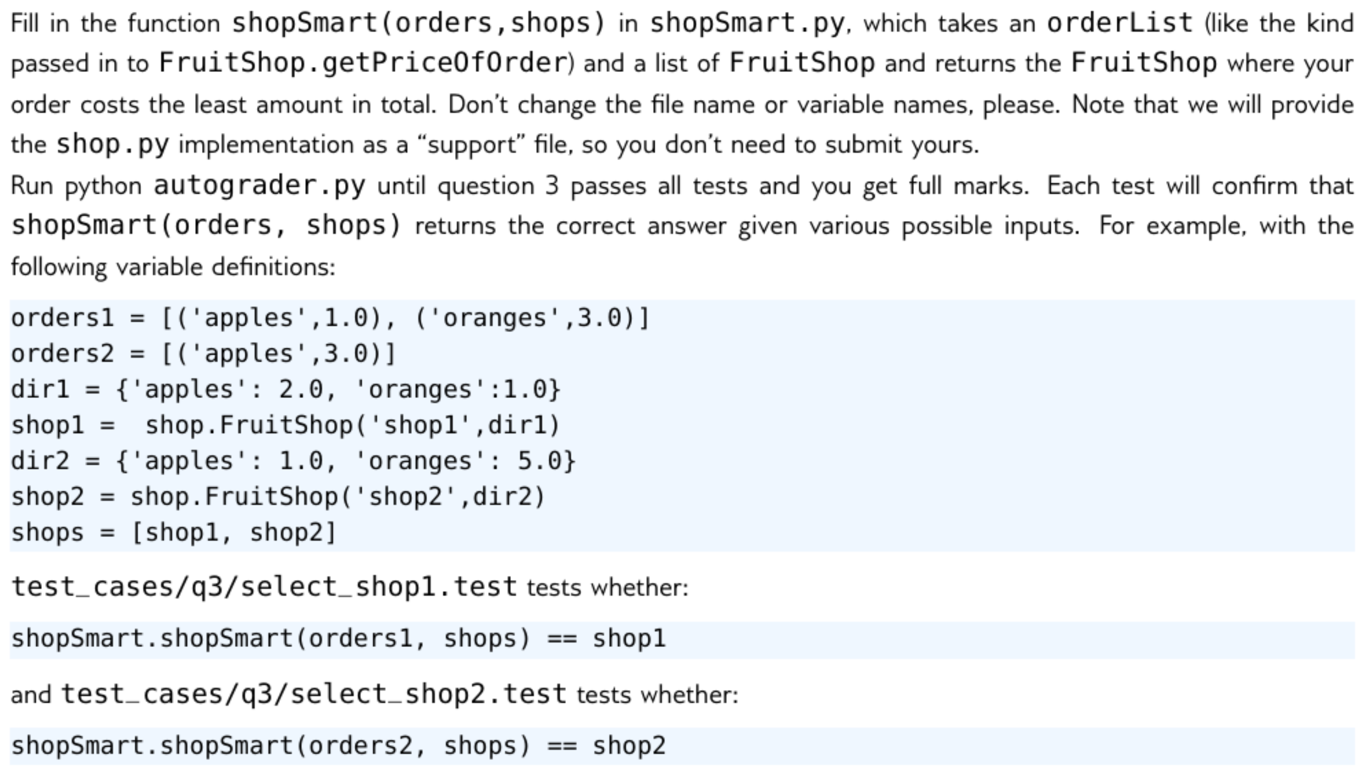
In this code, dictionaries, tuples, lists are being used. Dictionaries, tuples, and lists are three different data structures, each has its own characteristics.

Lists are mutable. Can be modified, added or removed. Dictionaries are mutable too, can add, remove or modify key-value pairs. But tuples are not mutable, can not change the values of its elements.

Examples:

Lists: [a, b, c]
Tuples: (1, 2, 3)
Dictionaries: {‘a’: ‘1’, ‘b’: ‘2’, ‘c’: ‘3’}
List of Tuples = [('apple', 2), ('orange', 3), ('pear', 1)]
Dictionary of Lists = {'fruits': ['apple', 'orange', 'pear'], 'quantities': [2, 3, 1]}
Dictionary of Tuples = {'apple': (2, 1.5), 'orange': (3, 2.0), 'pear': (1, 1.75)}
List of Dictionaries = [{'fruit': 'apple', 'qty': 2}, {'fruit': 'orange', 'qty': 3}]

3. <u>shopSmart Function:</u>



Fill in the function `shopSmart(orders,shops)` in `shopSmart.py`, which takes an `orderList` (like the kind passed in to `FruitShop.getPriceOfOrder`) and a list of `FruitShop` and returns the `FruitShop` where your order costs the least amount in total. Don’t change the file name or variable names, please. Note that we will provide the `shop.py` implementation as a “support” file, so you don’t need to submit yours.
Run python `autograder.py` until question 3 passes all tests and you get full marks. Each test will confirm that `shopSmart(orders, shops)` returns the correct answer given various possible inputs. For example, with the following variable definitions:

```
orders1 = [('apples',1.0), ('oranges',3.0)]
orders2 = [('apples',3.0)]
dir1 = {'apples': 2.0, 'oranges':1.0}
shop1 =  shop.FruitShop('shop1',dir1)
dir2 = {'apples': 1.0, 'oranges': 5.0}
shop2 = shop.FruitShop('shop2',dir2)
shops = [shop1, shop2]
```

`test_cases/q3/select_shop1.test` tests whether:

```
shopSmart.shopSmart(orders1, shops) == shop1
```

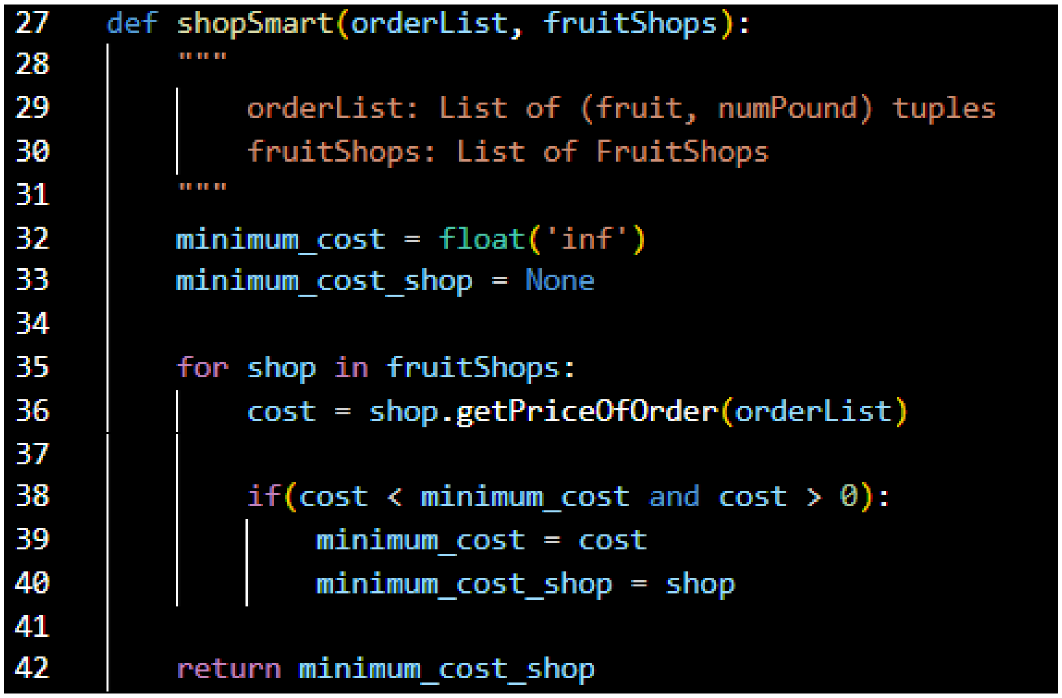
and `test_cases/q3/select_shop2.test` tests whether:

```
shopSmart.shopSmart(orders2, shops) == shop2
```

Analysis of the Problem Statement:

Write a function that will take a list (list of fruit orders) and a list of fruit shops. Then it will return the fruit shop with least amount of total cost for all the fruits ordered to be purchased. There is a helper class named "FruitShop" which has some helper functions. The correctness of the code will depend on all the test cases being passed

The Solution:



```python
def shopSmart(orderList, fruitShops):
    """
        orderList: List of (fruit, numPound) tuples
        fruitShops: List of FruitShops
    """
    minimum_cost = float('inf')
    minimum_cost_shop = None

    for shop in fruitShops:
        cost = shop.getPriceOfOrder(orderList)

        if(cost < minimum_cost and cost > 0):
            minimum_cost = cost
            minimum_cost_shop = shop

    return minimum_cost_shop
```

Explanation:

The solution takes a list of orders and a list of fruit shops. Here "minimum_cost" was set to positive infinity which ensures that the first valid cost encountered in the loop will always be smaller than the initial value of "minimum_cost".

"minimum_cost_shop" is set to None, because none of the shops have been considered to be of minimum cost yet. This allows the first shop with a valid cost to be automatically assigned as the minimum cost shop.

The code iterates through each shop, calculates the cost of the orders with the help of helper class functions – "getPriceOfOrder", and updates "minimum_cost_shop".

Finally, it returns the Fruit Shop with the minimum cost for the given order. If all costs are invalid, it returns None.

Challenges:

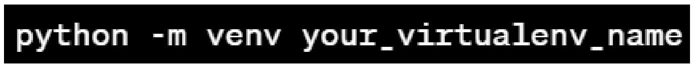
Had to look at the underlying implementations of the helper class and functions to fully understand how the things worked.

Findings:

How we can work with objects or classes. How modularity helps the code to look clean. No significant findings found, since most of the significant findings were found while doing the second task.

Overall Findings:

I was able to do the tasks without creating a conda environment. I was able to find an alternate solution – which is: using a virtual environment.

The command looks like this: 

```
python -m venv your_virtualenv_name
```

After creating the virtual environment, we need to go the scripts folder in the terminal and write: ".\activate" – to activate the virtual environment. After that we can proceed to do the tasks in the virtual environment.