COEN 166 Artificial Intelligence
Lab Assignment #1 Report

Name: Marianne Fuyu Yamazaki Dorr **Student ID: 1606975**

Question 1: Addition

Source Code:

```
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
```

Result:

```
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 2: buyLotsOfFruit function

Source Code:

```
def buyLotsOfFruit(orderList):
    """
        orderList: List of (fruit, numPounds) tuples
    Returns cost of order
    """
    totalCost = 0.0
    for x in orderList:
        if (x[0] in fruitPrices):
            totalCost += (x[1] * fruitPrices[x[0]])
        else:
            print("Error: One or more fruits not part of the list")
            return None
    return totalCost
```

Explanation:

First, we iterate through all the fruits in the orderList. We check if the fruit is part of the fruitPrices list. If it is, then we add the price of the fruit multiplied by the given amount to the totalCost using the price given by the tuple in fruitPrices and the amount given in the tuple in orderList for the appropriate fruit. If the fruit is not present in fruitPrices, the code will print an error message and return None. Once the code is done iterating through the orderList, it will return the totalCost of the orderList.

Result:

```
Question q2
 ===========
 *** 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 ###
```

Question 3: shopSmart function

Source Code:

```
def shopSmart(orderList, fruitShops):
    """
        orderList: List of (fruit, numPound) tuples
        fruitShops: List of FruitShops
    """
    "*** YOUR CODE HERE ***"
    current_min = None
    best_shop = None
    for current_shop in fruitShops:
        totalCost = current_shop.getPriceOfOrder(orderList)
        if (current_min == None):
            current_min = totalCost
            best_shop = current_shop
        if (current_min > totalCost):
            current_min = totalCost
            best_shop = current_shop
    return best_shop
```

Explanation:

First initialize two new variables to keep track of the lowest cost order in current_min and the shop associated with the lowest cost order in best_shop to the value None. Iterate through the list of shops in fruitShops. For each shop, get the total price of the orderList using the definition getPriceOfOrder() from the shop.py class. If the current_min is None, set the current_min to the totalCost and the best_shop to the current shop being iterated. Through the next iterations, check if the new totalCost is better (lower) than the current_min. If so, set the new current_min to the new totalCost and the best_shop to the current_shop. Once the shop iteration is done, return the best_shop.

Result:

```
Question q3
===========

Welcome to shop1 fruit shop
Welcome to shop2 fruit shop
*** PASS: test_cases\q3\select_shop1.test
***   shopSmart(order, shops) selects the cheapest shop
Welcome to shop1 fruit shop
Welcome to shop2 fruit shop
*** PASS: test_cases\q3\select_shop2.test
***   shopSmart(order, shops) selects the cheapest shop
Welcome to shop1 fruit shop
Welcome to shop2 fruit shop
Welcome to shop3 fruit shop
*** PASS: test_cases\q3\select_shop3.test
***   shopSmart(order, shops) selects the cheapest shop

### Question q3: 1/1 ###
```