COEN 166 Artificial Intelligence
Lab Assignment #1 Sample Report

Collin Paiz **1576109**

Question 1: Addition

Implementation

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

Results

Question q1
==========

Passed a = 1 and b = 1, return 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, return 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, return a+b = 7.9
*** PASS: test_cases/q1/addition3.test
*** add(a,b) returns the sum of a and b

Question q1: 1/1

Question 2: buyLotsOfFruit function

Implementation

```
def buyLotsOfFruit(orderList):
    """
        orderList: List of (fruit, numPounds) tuples

    Returns cost of order
    """
```

```
    totalCost = 0.0
    "*** YOUR CODE HERE ***"
    for fruit, pounds in orderList:
        if fruit in fruitPrices:
            totalCost += fruitPrices[fruit] * pounds
        else:
            print("Error. Fruit not in price list")
            return None

    return totalCost
```

Explanation
We first declare totalCost to keep track of the total cost of the fruit order, and set its value to 0.0. Then we iterate through the orderList, grabbing both the fruit and pounds for each element (tuple) in the list. If that fruit is present in the fruitPrices dictionary, we will add the price of that fruit multiplied by the number of pounds the order calls for to the totalCost. If the fruit is not present in the fruitPrices dictionary, we will print an error message and return *None*. After iterating through the orderList and evaluating the totalCost, we will return totalCost.

Results
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

Implementation

```
def shopSmart(orderList, fruitShops):
    """
        orderList: List of (fruit, numPound) tuples
        fruitShops: List of FruitShops
    """
```

```
    "*** YOUR CODE HERE ***"
    minCost = float('inf')
    minShop = None
    for shop in fruitShops:
        temp = shop.getPriceOfOrder(orderList)
        if temp < minCost:
            minCost = temp
            minShop = shop
    return minShop
```

Explanation

First we declare minCost to keep track of the minimum cost of the order at each shop, and set its value to infinity, so any possible price in the orderList will be less than it. We also declare minShop to keep track of which shop is the least expensive for the order. We then iterate through each shop in the fruitShops list. Using the Shop class function "getPriceOfOrder", we can set the price of the order from the current shop to a temp variable. If that temp variable (the price of the order at the current shop) is less than the minCost so far, then it sets the minCost to the new lowest cost (temp) and sets the minShop to the current shop. After iterating through all the shops, minShop should be set to the least expensive shop, and so we will return it (minShop).

Results

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