



Department of Computer Science and Engineering

Islamic University of Technology (IUT)

A subsidiary organ of OIC

Laboratory Report

CSE 4618: Artificial Intelligence Lab

Name	: Mirza Mohammad Azwad
Student ID	: 200042121
Section	: BSc in SWE
Semester	: 6th Semester
Academic Year	: 2022-23
Date	: 22/01/2024
Lab No	: 0

Contents

Section	Page Number
1. Introduction	2
2. Analysis of Problem	3
3. Explanation of Solution	6
4. Problems Faced and Interesting Findings	10

Introduction

In this lab we essentially had to do some problem solving in python involving basic python programming and understanding the basics of python syntax. The 3 problems outlined were essentially simple python code snippets that we had to modify to solve these problems and find the desired solution.

The lab was evaluated initially with the autograd.py script running which essentially checks if the modified python code snippets passed the test cases outlined.

We were also introduced to creating virtual environments for python using conda so that we could run the code in the desired environment for this course which is the python 3.6 version that doesn't match my Operating System's python version.

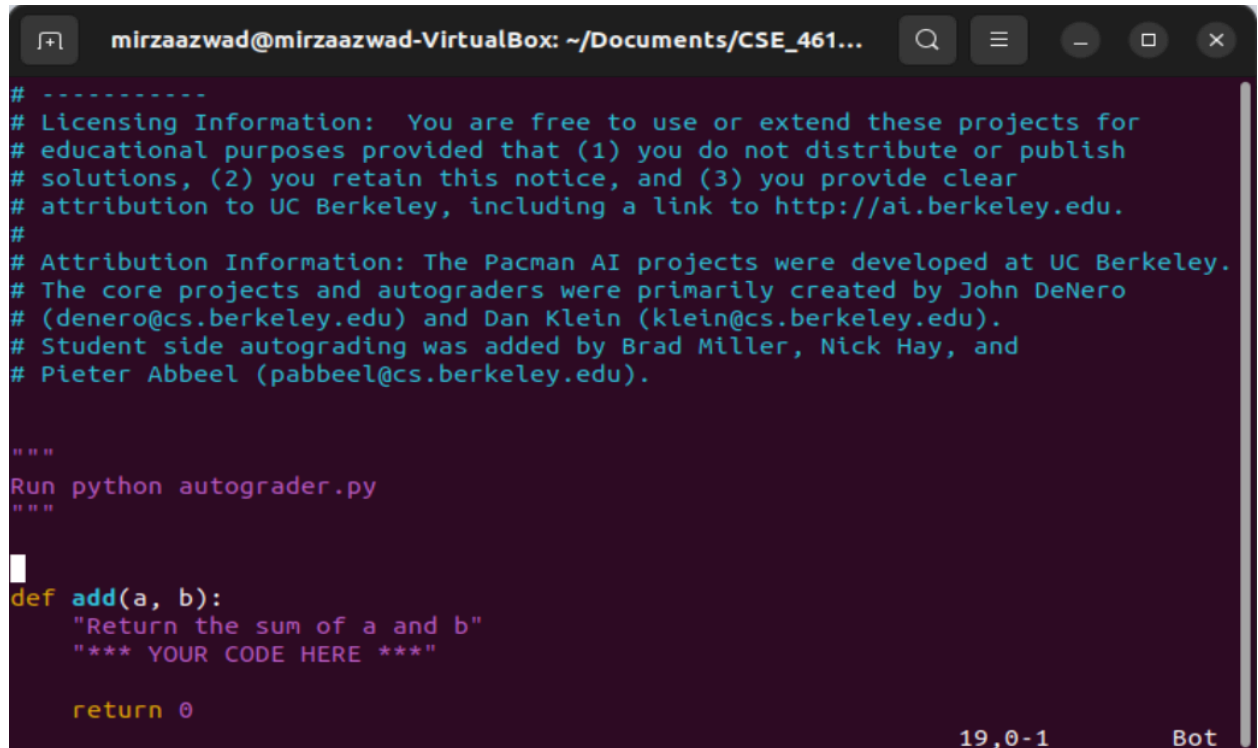
I carried out this lab in a Virtual Machine(VirtualBox) loaded with Ubuntu 22.04LTS as basic linux was a requirements per the taskbook and the was recommended as per the lab instructions

Analysis of the Problems

There were 3 main problems:

1. **addition.py**

This problem involved this python function that is supposed to return the result of addition by modifying the **add** function

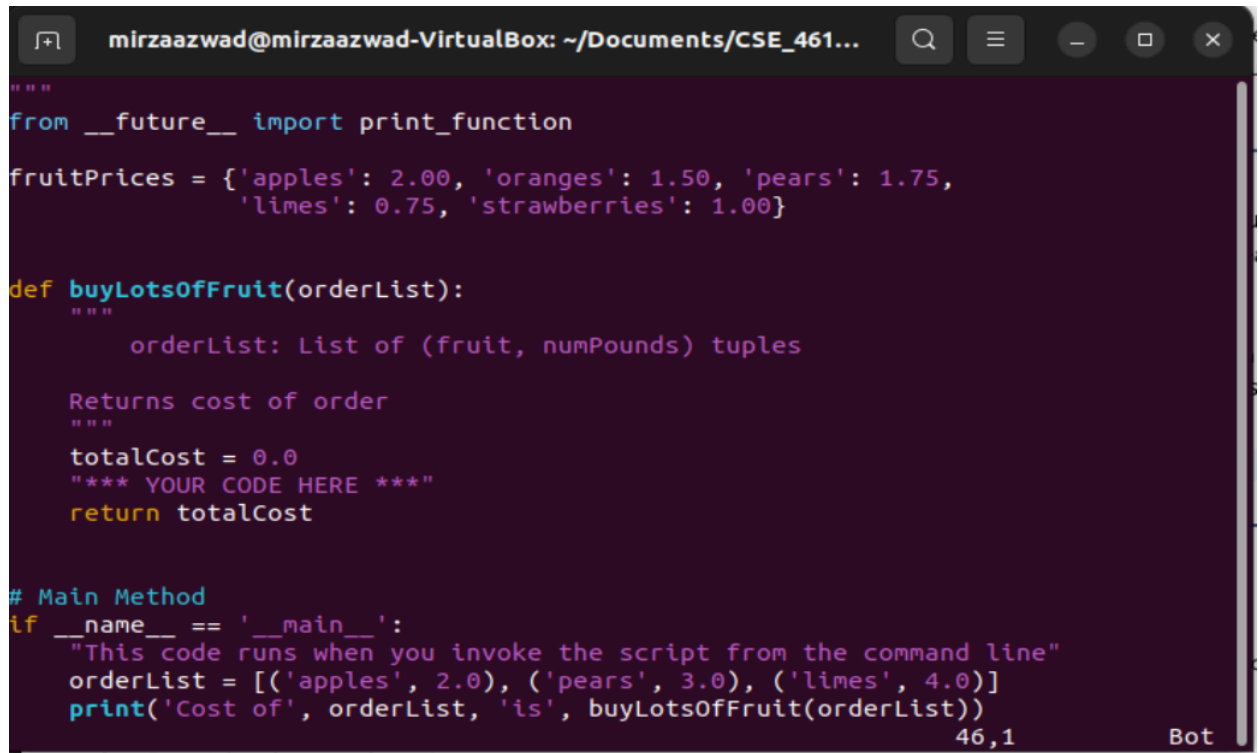


```
mirzaazwad@mirzaazwad-VirtualBox: ~/Documents/CSE_461...  
# -----  
# Licensing Information: You are free to use or extend these projects for  
# educational purposes provided that (1) you do not distribute or publish  
# solutions, (2) you retain this notice, and (3) you provide clear  
# attribution to UC Berkeley, including a link to http://ai.berkeley.edu.  
#  
# Attribution Information: The Pacman AI projects were developed at UC Berkeley.  
# The core projects and autograders were primarily created by John DeNero  
# (denero@cs.berkeley.edu) and Dan Klein (klein@cs.berkeley.edu).  
# Student side autograding was added by Brad Miller, Nick Hay, and  
# Pieter Abbeel (pabbeel@cs.berkeley.edu).  
  
"""  
Run python autograder.py  
"""  
  
def add(a, b):  
    "Return the sum of a and b"  
    """ YOUR CODE HERE """  
  
    return 0
```

19,0-1 Bot

2. buyLotsOfFruits.py

This problem involved a function called *buyLotsOfFruits* and in this function we had to find the total cost calculated for all the fruits and then return the **total computed cost**. The fruits are present as *fruitPrices* in a dictionary with key value pairs having the fruit name as an **str** and the value or price of each pound as a **float**. The *orderList* is present as a list of tuples denoting the fruit name as **str** with the amount in pounds given as **float**



```
mirzaazwad@mirzaazwad-VirtualBox: ~/Documents/CSE_461...  
"""  
from __future__ import print_function  
  
fruitPrices = {'apples': 2.00, 'oranges': 1.50, 'pears': 1.75,  
               'limes': 0.75, 'strawberries': 1.00}  
  
def buyLotsOfFruit(orderList):  
    """  
        orderList: List of (fruit, numPounds) tuples  
  
        Returns cost of order  
    """  
    totalCost = 0.0  
    """ YOUR CODE HERE """  
    return totalCost  
  
# Main Method  
if __name__ == '__main__':  
    "This code runs when you invoke the script from the command line"  
    orderList = [('apples', 2.0), ('pears', 3.0), ('limes', 4.0)]  
    print('Cost of', orderList, 'is', buyLotsOfFruit(orderList))  
46,1 Bot
```

3. shopSmart.py

In this problem we simply had to find the optimal shop or in other words, the shop with the least total cost. We are provided with 3 shops and we compare their total prices for the mentioned order and then print the title of the shop with the least total cost. We have the *orders* as a list of tuples like the problem above. And we are using another module *shop* which we imported and use the *FruitShop* instantiation.

```

mirzaazwad@mirzaazwad-VirtualBox: ~/Documents/CSE_461...
Welcome to shop1 fruit shop
Welcome to shop2 fruit shop
For orders: [('apples', 1.0), ('oranges', 3.0)] best shop is shop1
For orders: [('apples', 3.0)] best shop is shop2
"""
from __future__ import print_function
import shop

def shopSmart(orderList, fruitShops):
    """
    orderList: List of (fruit, numPound) tuples
    fruitShops: List of FruitShops
    """
    """ YOUR CODE HERE """

    return None

if __name__ == '__main__':
    "This code runs when you invoke the script from the command line"
    orders = [('apples', 1.0), ('oranges', 3.0)]
    dir1 = {'apples': 2.0, 'oranges': 1.0}

```

23,1 70%

```

mirzaazwad@mirzaazwad-VirtualBox: ~/Documents/CSE_461...
def shopSmart(orderList, fruitShops):
    """
    orderList: List of (fruit, numPound) tuples
    fruitShops: List of FruitShops
    """
    """ YOUR CODE HERE """

    return None

if __name__ == '__main__':
    "This code runs when you invoke the script from the command line"
    orders = [('apples', 1.0), ('oranges', 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]
    print("For orders ", orders, ", the best shop is", shopSmart(orders, shops).
getName())
    orders = [('apples', 3.0)]
    print("For orders: ", orders, ", the best shop is", shopSmart(orders, shops)
getName())

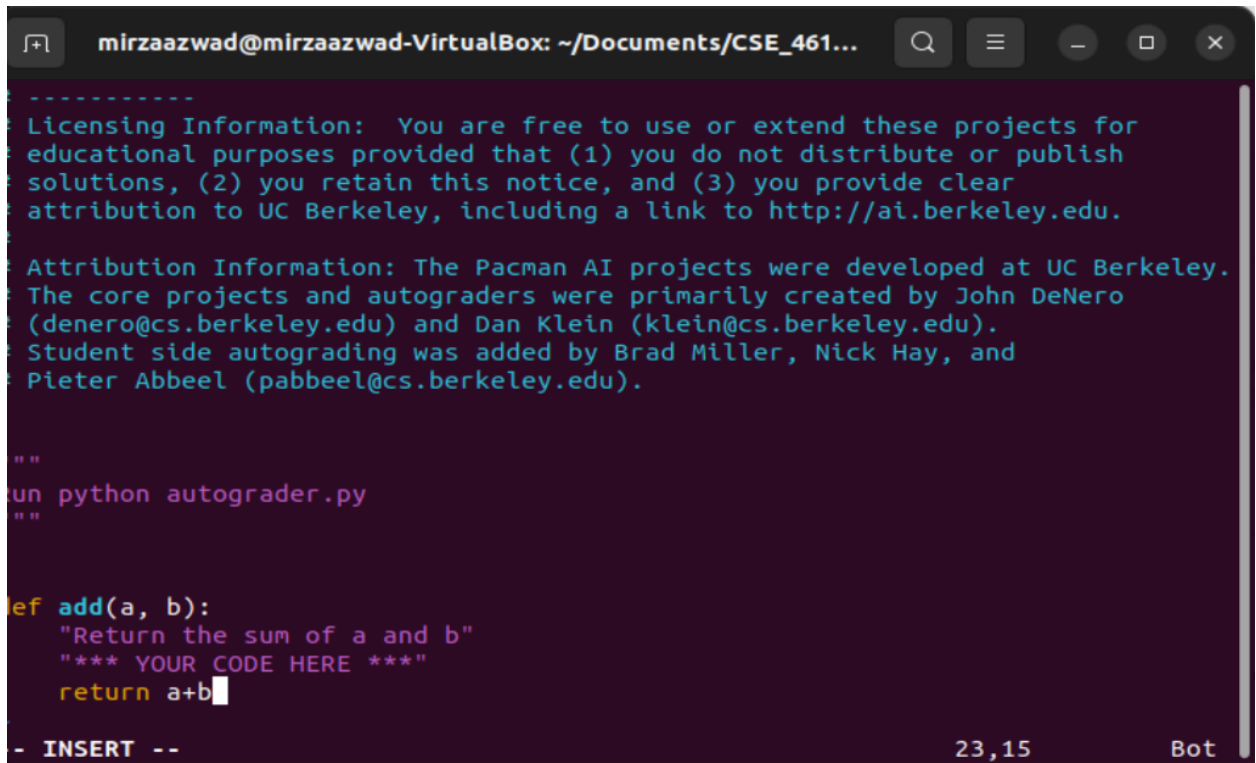
```

47,1 Bot

Explanation of Solution

1. addition.py

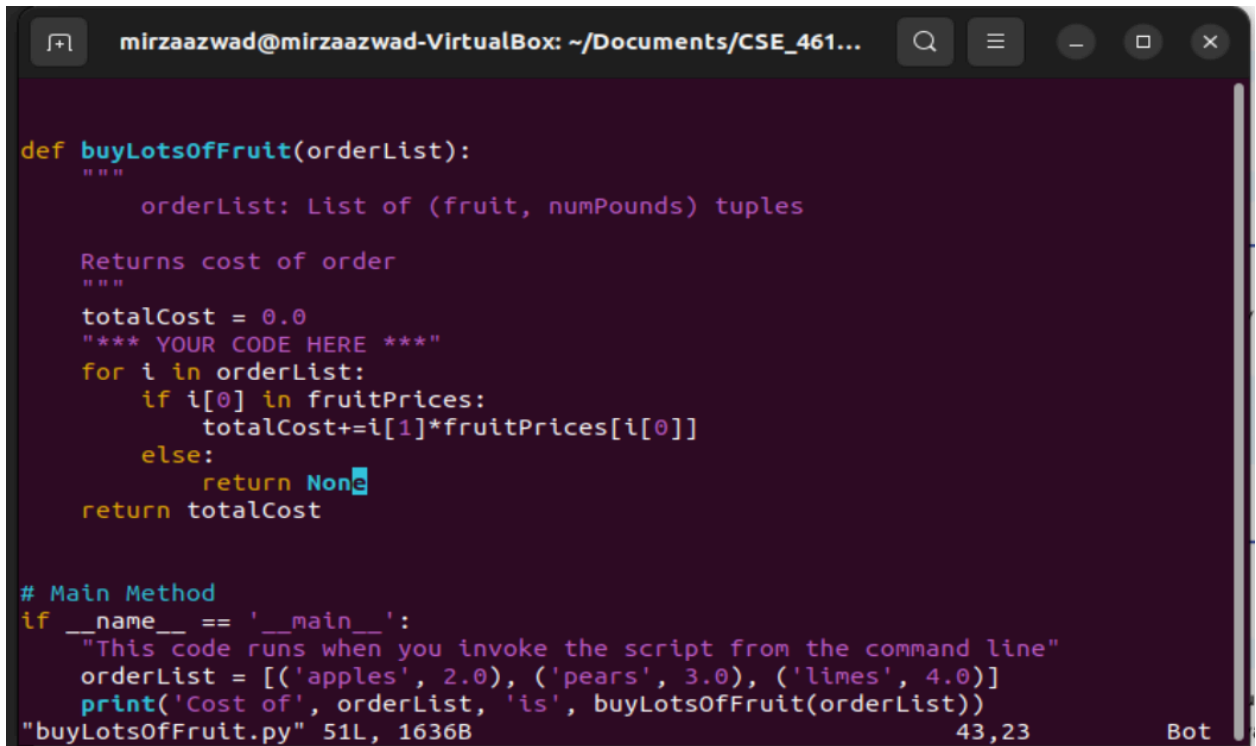
We can simply return the sum of the arguments *a* and *b* created using the '+' arithmetic operator.



```
mirzaazwad@mirzaazwad-VirtualBox: ~/Documents/CSE_461...  
-----  
Licensing Information: You are free to use or extend these projects for  
educational purposes provided that (1) you do not distribute or publish  
solutions, (2) you retain this notice, and (3) you provide clear  
attribution to UC Berkeley, including a link to http://ai.berkeley.edu.  
  
Attribution Information: The Pacman AI projects were developed at UC Berkeley.  
The core projects and autograders were primarily created by John DeNero  
(denero@cs.berkeley.edu) and Dan Klein (klein@cs.berkeley.edu).  
Student side autograding was added by Brad Miller, Nick Hay, and  
Pieter Abbeel (pabbeel@cs.berkeley.edu).  
  
""  
Run python autograder.py  
""  
  
def add(a, b):  
    "Return the sum of a and b"  
    """ YOUR CODE HERE """  
    return a+b  
  
- INSERT -- 23,15 Bot
```

2. buyLotsOfFruit.py

In this problem, I simply checked if the fruit mentioned in the *orderList* is present in the *fruitPrices* dictionary. And as the problem stated to return None if the fruit is not present, if it is present in that case we add the price to the totalCost as the product of the per pound price in *fruitPrices*, and the pounds mentioned in the *orderList* gives the cost of the fruit order mentioned in the *orderList*.



```
def buyLotsOfFruit(orderList):  
    """  
        orderList: List of (fruit, numPounds) tuples  
  
        Returns cost of order  
    """  
    totalCost = 0.0  
    """ YOUR CODE HERE """  
    for i in orderList:  
        if i[0] in fruitPrices:  
            totalCost+=i[1]*fruitPrices[i[0]]  
        else:  
            return None  
    return totalCost  
  
# Main Method  
if __name__ == '__main__':  
    "This code runs when you invoke the script from the command line"  
    orderList = [('apples', 2.0), ('pears', 3.0), ('limes', 4.0)]  
    print('Cost of', orderList, 'is', buyLotsOfFruit(orderList))  
"buyLotsOfFruit.py" 51L, 1636B 43,23 Bot
```


3. shopSmart.py

First we look at the *shop* module where we see the *FruitShop* class which has 2 methods that we can use that is *getPriceOfOrder* and the magic method *__str__* which can be used to get the name of the shop in question.

```

class FruitShop:
    def __init__(self, name, fruitPrices):
        """
        name: Name of the fruit shop

        fruitPrices: Dictionary with keys as fruit
        strings and prices for values e.g.
        {'apples':2.00, 'oranges': 1.50, 'pears': 1.75}
        """
        self.fruitPrices = fruitPrices
        self.name = name
        print('Welcome to %s fruit shop' % (name))

    def getCostPerPound(self, fruit):
        """
        fruit: Fruit string
        Returns cost of 'fruit', assuming 'fruit'
        is in our inventory or None otherwise
        """
        if fruit not in self.fruitPrices:
            return None

```

31,1 35%

```

        if fruit not in self.fruitPrices:
            return None
        return self.fruitPrices[fruit]

    def getPriceOfOrder(self, orderList):
        """
        orderList: List of (fruit, numPounds) tuples

        Returns cost of orderList, only including the values of
        fruits that this fruit shop has.
        """
        totalCost = 0.0
        for fruit, numPounds in orderList:
            costPerPound = self.getCostPerPound(fruit)
            if costPerPound != None:
                totalCost += numPounds * costPerPound
        return totalCost

    def getName(self):
        return self.name

    def __str__(self):
        return "<FruitShop: %s>" % self.getName()

```

52,0-1 91%

Using *shop.py*, in the next step we compute the total cost of the order using *getPriceOfOrder* and we compare with a very large value which we change as per the cost obtained for that shop, the idea is to update the large value *minPrice* to get the current minimum price when compared with each of the existing shops. We iterate over the list using an implicit iterator using the python for loop that iterates over the *fruitShops* list. This helps us get the price for each shop and compare with the *minPrice*. This helps us narrow down the shop and store the cheapest shop for each iteration in the *bestShop*

```

def shopSmart(orderList, fruitShops):
    """
    orderList: List of (fruit, numPound) tuples
    fruitShops: List of FruitShops
    """
    """ YOUR CODE HERE """
    minPrice=float('inf')
    bestShop=None
    for i in fruitShops:
        price=i.getPriceOfOrder(orderList)
        if minPrice>price:
            minPrice=price
            bestShop=i
    return bestShop

if __name__ == '__main__':
    "This code runs when you invoke the script from the command line"
    orders = [('apples', 1.0), ('oranges', 3.0)]
    dir1 = {'apples': 2.0, 'oranges': 1.0}
    shop1 = shop.FruitShop('shop1', dir1)
    dir2 = {'apples': 1.0, 'oranges': 5.0}
    "shopSmart.py" 53L, 1897B

```

Problems Faced and Interesting Findings

I faced a problem in the last problem which helped me discover how to set the largest possible float value in python3 using `float('inf')` which essentially sets the largest possible value, initially I used to set the `minPrice` to a large value denoted by `1e12` which I observed is exceeded as the *auograd.py* clearly shows that the third task does not pass all the test cases.

```

mirzaazwad@mirzaazwad-VirtualBox: ~/Documents/CSE_461...
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
*** FAIL: test_cases/q3/select_shop3.test
*** shopSmart(order, shops) must select the cheapest shop
*** student result: "None"
*** correct result: "<FruitShop: shop3>"
*** Tests failed.

### Question q3: 0/1 ###

```

```

mirzaazwad@mirzaazwad-VirtualBox: ~/Documents/CSE_461...
"""
from __future__ import print_function
import shop

def shopSmart(orderList, fruitShops):
    """
    orderList: List of (fruit, numPound) tuples
    fruitShops: List of FruitShops
    """
    """ YOUR CODE HERE """
    minPrice=1e12
    bestShop=None
    for i in fruitShops:
        price=i.getPriceOfOrder(orderList)
        if minPrice>price:
            minPrice=price
            bestShop=i
    return bestShop

if __name__ == '__main__':
    "This code runs when you invoke the script from the command line"
    39,17 70%

```

When I did update to setting the `minPrice` as a typecast of `'inf'`, this worked perfectly and passed all the test cases with the interesting finding being how to set a large value in python.

```
mirzaazwad@mirzaazwad-VirtualBox: ~/Documents/CSE_461...  
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 ###  
  
Finished at 3:15:37  
  
Provisional grades  
=====br/>Question q1: 1/1  
Question q2: 1/1  
Question q3: 1/1  
-----  
Total: 3/3  
  
Your grades are NOT yet registered. To register your grades, make sure  
to follow your instructor's guidelines to receive credit on your project.  
  
(cse4618) mirzaazwad@mirzaazwad-VirtualBox:~/Documents/CSE_4618_ArtificialIntell  
gence/Lab6$
```