Bahria University
Discovering Knowledge

Artificial Intelligence Practical Lab File (Solution)

TARWAN KUMAR

Code+Course: CSL - 411

INDEX

Lab #	DATE	Task #	Objective	Remarks	Initials
1		10	Python Programming 1		
2		7	Python Programming 2		
3		3	Python Programming 3		
4		2	BFS and DFS		
5		2	Greedy and Astar		
6		2	Hill Climbing Simulated Annealing		
7		3	Genetic Algorithm		
8		2	ID3 Linear Regression		
9		2	Expert Systems		
10		1	Neural Network		
11		3	Prolog-I		
12		3	Prolog-II		

CSL 411 –Artificial Intelligence Lab

FALL 2017

Lab # 01

Python Programming 1

COURSE INSTRUCTOR: TARIQ SIDDIQUE

LAB ENGINEER: TARWAN KUMAR

DEPARTMENT OF COMPUTER SCIENCE

BAHRIA UNIVERSITY, KARACHI CAMPUS

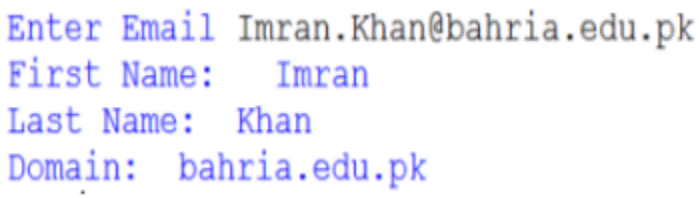
Task # 1

Write a program that needs to ask the user for her or his email address in the format firstname.lastname@bahria.edu.pk OR firstname.lastname@gmail.com. The application takes as input this email address, parses the email and replies to the user with first name, last name and host name

Code

```
email = input("Enter Email")

ind = email.index(".")

at = email.index("@")

fname = email[0:ind]

lname = email[ind+1:at]

domain = email[at+1:]

print ("First Name: ",fname)

print ("Last Name: ",lname)

print ("Host Name: ",domain)
```

Output



Task # 2

Write a program that converts a positive integer into the Roman number system. The Roman number system has digits I (1), V (5), X (10), L (50), C(100), D(500) and M(1000). Numbers up to 3999 are formed according to the following rules: a) As in the decimal system, the thousands, hundreds, tens and ones are expressed separately. b) The numbers 1 to 9 are expressed as: 1 I 6 VI 2 II 7 VII 3 III 8 VIII 4 IV 9 IX 5 V (An I preceding a V or X is subtracted from the value, and there cannot be more than three I's in a row.) c) Tens and hundreds are done the same way, except that the letters X, L, C, and C, D, M are used instead of I, V, X respectively. Your program should take an input, such as 1978, and convert it to Roman numerals, MCMLXXVIII.

Code

```
num_map=[(1000,"M"), (900,'CM'),(500,'D'),(400,'CD'), (100,'C'), (90,'XC'),(50,'L'),(40,'XL'),
(10,'X'),(9,'IX'),(5,'V'),(4,'IV'),  (1,'I')]

def numToroman(num):
    roman=''
    while num>0:
        for i,r in num_map:
            while num >=i:
                roman +=r
                num -=i
    return roman
num = int(input("input a number \n"))
if(num > 1 and num < 3999):
    print("The Roman Numeral of ",num," is "+numToroman(num))
else:
    print("Num should between 1-3999")
```

Output

```
input a number
1978
The Roman Numeral of  1978  is MCMLXXVIII
```

Task # 3

Write a program that calculates the user's body mass index (BMI) and categorizes it as underweight, normal, overweight, or obese, based on the table from the United States Centers for Disease Control.

Code

```
weight = int(input("Enter Weight in Pounds: "))
hei = int(input("Enter Height in Inches: "))
hei = hei**2
bmi=weight/hei
bmi=bmi*703
if bmi < 18.5:
 print("You have a BMI of ",bmi," and your weight status is Underwight")
elif bmi >= 18.5 and bmi <= 24.9:
 print("You have a BMI of ",bmi," and your weight status is Normal")
elif bmi >= 25 and bmi <= 29.9:
 print("You have a BMI of ",bmi," and your weight status is Overweight")
elif bmi > 30:
 print("You have a BMI of ",bmi," and your weight status is Obese")
```

Output

```
Enter Weight in Pounds: 110
Enter Height in Inches: 60
You have a BMI of  21.480555555555554  and your weight status is Normal
```

Task # 4

Write a program to compute quotient and remainder of a number without using division ('/') operator and modulo ('%') operator.

Code

```
deno = 4

nume = 18

i = 0

while nume>=deno:

 nume=nume-deno

 i+=1

print("Quotient ",i," Reminder ",nume)
```

Output

```
Quotient  4  Reminder  2
```

Task # 5

Blood types are important for blood transfusion. The blood types must be matched since if not matched properly, the recipient's blood can form clots and these can lead to heart attacks, embolisms and strokes.

Code

```
x = "y"
while x == "y":
    blood = input("Enter Blood type: ")
    if blood == "O-":
        print("Can Receive from: O-")
        print("Can Donate to: O-, O+, A-, A+, B-, B+, AB-, AB+")
        x = input("Continue? ")
    elif blood == "O+":
        print("Can Receive from: O-, O+")
        print("Can Donate to: O+, A+, B+, AB+")
        x = input("Continue? ")
    elif blood == "A-":
        print("Can Receive from: O-, A-")
        print("Can Donate to: A-, A+, AB-, AB+")
        x = input("Continue? ")
    elif blood == "A+":
        print("Can Receive from: O-, O+, A-, A+")
        print("Can Donate to: A+, AB+")
        x = input("Continue? ")
    elif blood == "B-":
        print("Can Receive from: O-, B-")
        print("Can Donate to: B-, B+, AB-, AB+")
        x = input("Continue? ")
```

```
    elif blood == "B+":
        print("Can Receive from: O-, O+, B-, B+")
        print("Can Donate to: B+, AB+")
        x = input("Continue? ")
    elif blood == "AB-":
        print("Can Receive from: O-, A-, B-, AB-")
        print("Can Donate to: AB-, AB+")
        x = input("Continue? ")
    elif blood == "AB+":
        print("Can Receive from: O-, O+, A-, A+, B-, B+, AB-, AB+")
        print("Can Donate to: AB+")
        x = input("Do you want to Continue (y/n)? ")
```

Output

```
Enter Blood type: O+
Can Receive from: O-, O+
Can Donate to: O+, A+, B+, AB+
Continue? y
Enter Blood type: A+
Can Receive from: O-, O+, A-, A+
Can Donate to: A+, AB+
Continue? n
```

Task # 6

Write a program to take an array and find the number of occurrences each number had.

Code

```
from collections import Counter

l = [1,1,3,6,9,6,9,3,6,8,5]

print(Counter(l))
```

Output

```
Counter({6: 3, 1: 2, 3: 2, 9: 2, 5: 1, 8: 1})
```

Task # 7

Given the following array, display its data graphically by plotting each numeric value as a bar of asterisks (*) as shown in the diagram. [Hint: use string repetition operator] array = [10, 19, 5, 1, 7, 14, 0, 7, 5]

Code

```
list = [10,19,5,1,7,14,0,7,5]

print("Element  Value     Histogram")

for i in range(len(list)):

 print(i,"        ",list[i],"       ",list[i]*"*")
```

Output

```
Element  Value      Histogram
0          10          **********
1          19          ********************
2          5         *****
3          1         *
4          7         *******
5          14          **************
6          0
7          7         *******
8          5         *****
```

Task # 8

Suppose you have the following matrix: 2 0 4 2 6 3 9 9 1 0 4 1 7 1 2 3 7 4 2 2 2 7 1 6 1 5 8 7 4 1 CS Department, BUKC 12/12 Semester Fall 2017 CSL-411: Aritificial Intelligence Lab Lab01: Python Programming 1 Design then implement a Python program that will produce its transpose and print it along with the original one.

Code

```
X = [[2,0,4,2,6,3], [9,9,1,0,4,1], [7,1,2,3,7,4], [2,2,2,7,1,6], [1,5,8,7,4,1]]

result = [[X[j][i] for j in range(len(X))] for i in range(len(X[0]))]

print("Original Matrix")

for i in X:

 print(i)

print("Transposed Matrix")

for r in result:

 print(r)
```

OUTPUT

```
Original Matrix
[2, 0, 4, 2, 6, 3]
[9, 9, 1, 0, 4, 1]
[7, 1, 2, 3, 7, 4]
[2, 2, 2, 7, 1, 6]
[1, 5, 8, 7, 4, 1]
Transposed Matrix
[2, 9, 7, 2, 1]
[0, 9, 1, 2, 5]
[4, 1, 2, 2, 8]
[2, 0, 3, 7, 7]
[6, 4, 7, 1, 4]
[3, 1, 4, 6, 1]
```

Task # 9

Suppose you have the following matrices: Write a Python program that will calculate its product and print the resultant matrix.

CODE

```
m1 = [[1,2,3,4],
 [5,6,7,8]]
m2 = [[1,2,3],
 [4,5,6],
 [7,8,9],
 [10,11,12]]
result = [[0,0,0],
 [0,0,0]]
for i in range(len(m1)):
  for j in range(len(m2[0])):
    for k in range(len(m2)):
       result[i][j] += m1[i][k] * m2[k][j]


for i in result:
   print(i)
```

OUTPUT

```
[70, 80, 90]
[158, 184, 210]
```

Task # 10

Write a program that calculates the total score for students in a class. Suppose the scores are stored in a three-dimensional array named scores. The first index in scores refers to a student, the second refers to an exam, and the third refers to the part of the exam. Suppose there are 7 students, 5 exams, and each exam has two parts--the multiple-choice part and the programming part. So, scores[i][j][0] represents the score on the multiple-choice part for the i's student on the j's exam. Your program displays the total score for each student. [[[7.5, 20.5], [12, 22.5], [22, 33.5], [43, 21.5], [15, 2.5]], [[4.5, 21.5], [12, 22.5], [12, 34.5], [12, 20.5], [14, 9.5]], [[5.5, 30.5], [9.4, 2.5], [13, 33.5], [11, 23.5], [16, 2.5]], [[6.5, 23.5], [9.4, 32.5],[13, 34.5], [11, 20.5], [16, 7.5]], [[8.5, 25.5], [9.4, 52.5],[13, 36.5], [13, 24.5], [16, 2.5]], [[9.5, 20.5], [9.4, 42.5],[13, 31.5], [12, 20.5], [16, 6.5]], [[1.5, 29.5], [9.4, 22.5],[19, 30.5], [10, 30.5], [19, 5.0]]]

Code

```
num_array = [[[7.5, 20.5], [12, 22.5], [22, 33.5], [43, 21.5], [15, 2.5]], [[4.5, 21.5], [12, 22.5], [12, 34.5], [12, 20.5], [14, 9.5]], [[5.5, 30.5], [9.4, 2.5], [13, 33.5], [11, 23.5], [16, 2.5]], [[6.5, 23.5], [9.4, 32.5], [13, 34.5], [11, 20.5], [16, 7.5]], [[8.5, 25.5], [9.4, 52.5], [13, 36.5], [13, 24.5], [16, 2.5]], [[9.5, 20.5], [9.4, 42.5], [13, 31.5], [12, 20.5], [16, 6.5]], [[1.5, 29.5], [9.4, 22.5], [19, 30.5], [10, 30.5], [19, 5.0]] ]

for i in range(len(num_array)):

 totalscore = 0

 for j in range(len(num_array[i])):

    for k in range(len(num_array[i][j])):

        totalscore = totalscore + num_array[i][j][k]

 print("Student ", i+1, "Score is", totalscore)
```

Output

```
Student  1 Score is 200.0
Student  2 Score is 163.0
Student  3 Score is 147.4
Student  4 Score is 174.4
Student  5 Score is 201.4
Student  6 Score is 181.4
Student  7 Score is 176.9
```

CSL 411 –Artificial Intelligence Lab

FALL 2017

Lab # 02

Python Programming 2

COURSE INSTRUCTOR: TARIQ SIDDIQUE

LAB ENGINEER: TARWAN KUMAR

DEPARTMENT OF COMPUTER SCIENCE

BAHRIA UNIVERSITY, KARACHI CAMPUS

Task # 1

Write a python application with the following prototypes that returns the user's body mass index (BMI) **def computeBMI(weight, height):** To calculate BMI based on weight in pounds (lb) and height in inches (in), use this formula:

$$BMI = \frac{mass(lb)}{(height(in))^2} \times 703$$

and **def findStatus(bmi):**

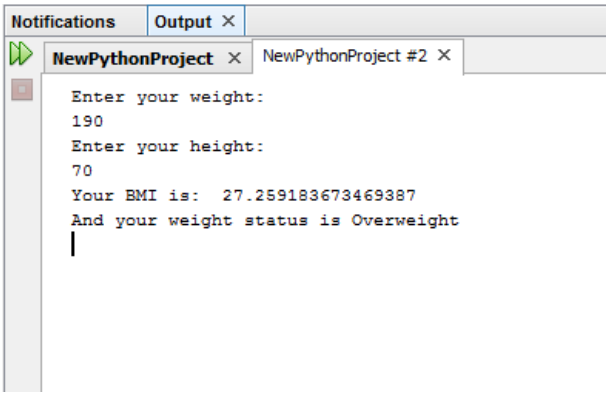
BMI	Weight Status
Below 18.5	Underweight
18.5 – 24.9	Normal
25.0-29.9	Overweight
30.0 and above	Obese

Categorizes it as underweight, normal, overweight, or obese, based on the table from the United States Centers for Disease Control:

CODE

```
def computeBMI(weight,height):
    bmi=(weight/height**2)*703
    return bmi
def findStatus(bmi):
    if bmi < 18.5:
     print('And your weight status is Underweight')
    elif bmi > 18.5 and bmi < 24.9:
     print('And your weight status is normal')
    elif bmi > 25 and bmi <29.9:
     print('And your weight status is Overweight')
    else:
     print('And your weigt status is obese')
pounds=int(input('Enter your weight in whole pounds:\n'))
inches=int(input('Enter your height in whole inches:\n'))
c=computeBMI(pounds,inches)
print('Your BMI is: ',c)
findStatus(c)
```

Output



Task # 2

Write the following 2 functions:

def ComputeOddSum(num):def ComputeEvenSum(num):

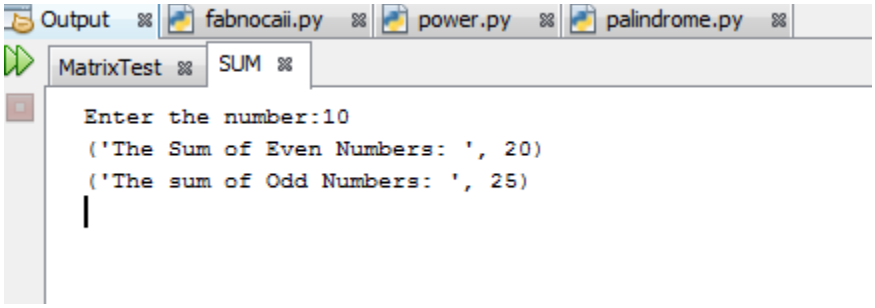
The function **ComputeOddSum** find the sum of all odd numbers less than num.

The function **ComputeEvenSum** find the sum of all even numbers less than num.

Code

```
def ComputeEvenSum(input):
    esum=0
    for i in range(1,input):
        if(i%2==0):
            esum=esum+i
    return esum

def ComputeOddSum(input):
    osum=0
    for i in range(1,input):
     if(i%2!=0):
      osum=osum+i
    return osum
number=int(input('Enter the number:'))
e=ComputeEvenSum(number)
print('The Sum of Even Numbers: ',e)
o=ComputeOddSum(number)
print('The sum of Odd Numbers: ',o)
```

Output



Task # 3

Write a python appliction MatrixTest to call the function with the following two matrices:

a) **Sum** that accepts two two-dimensional arrays (matrices) as arguments and returns a two-dimensional array representing their sum.
b) **Product** that accepts two two-dimensional arrays (matrices) as arguments and returns a two-dimensional array representing their product.

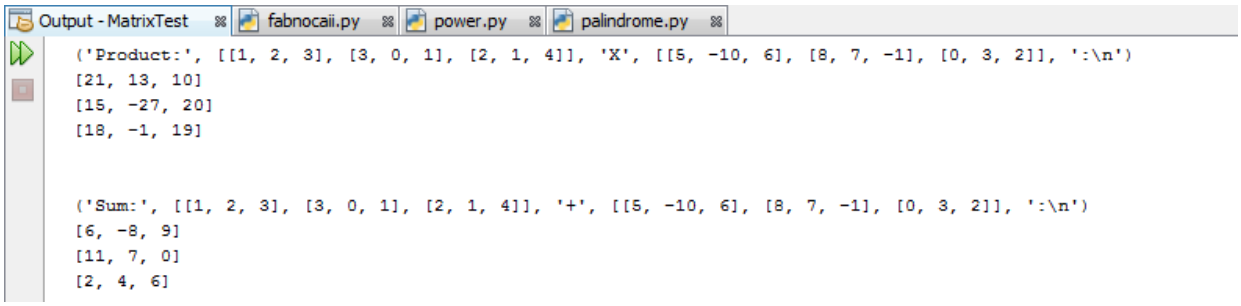
$$M1=\begin{pmatrix}1 & 2 & 3\\ 3 & 0 & 1\\ 2 & 1 & 4\end{pmatrix} \text{ and } M2=\begin{pmatrix}5 & -10 & 6\\ 8 & 7 & -1\\ 0 & 3 & 2\end{pmatrix} \text{ Note that: } M1+M2=\begin{pmatrix}6 & -8 & 9\\ 11 & 7 & 0\\ 2 & 4 & 6\end{pmatrix} \text{ and } M1\times M2=\begin{pmatrix}21 & 13 & 10\\ 15 & -27 & 20\\ 18 & -1 & 19\end{pmatrix}$$

Code

```
def ProductMatrix (A, B):
   rows_A = len(A)
   cols_A = len(A[0])
   rows_B = len(B)
   cols_B = len(B[0])
   if cols_A != rows_B:
     print ("Cannot multiply the two matrices. Incorrect dimensions.")
     return
   C = [[0 for row in range(cols_B)] for col in range(rows_A)]
   for i in range(rows_A):
       for j in range(cols_B):
           for k in range(cols_A):
               C[i][j] += A[i][k] * B[k][j]
   for k in C:
    print(k)
def SumMatrix(A,B):
    c = [[0 for row in range(len(A))] for col in range(len(B))]
    for i in range(len(A)):
        for j in range(len(B)):
            c[i][j]=A[i][j]+B[i][j]
    for k in c:
      print(k)

X = [[1,2,3],
   [3,0,1],
   [2,1,4]]
Y = [[5,-10,6],
   [8,7,-1],
   [0,3,2,]]
print('Product:',X,'X',Y,':\n')
ProductMatrix(X,Y)
print('\n')
```

```
print('Sum:',X,'+',Y,':\n')
SumMatrix(X,Y)
```

Output



Task # 4

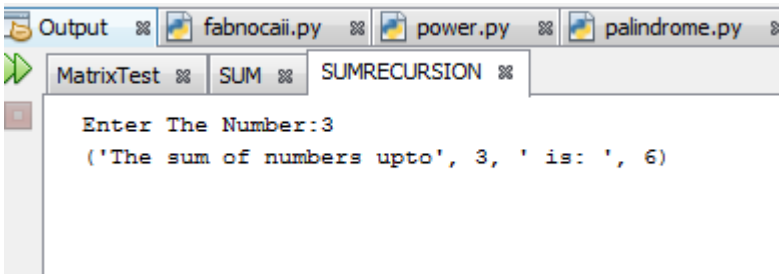
Write a recursive function to get sum of all number from 1 up to given number.

Example N = 5 Result must be sum (1+2+3+4+5) =15.

Code

```
def calSum(num):
    if num > 0:
        return num+calSum(num-1)
    else:
        return 0

input=int(input('Enter The Number:'))
S=calSum(input)
print('The sum of numbers upto',input,' is: ',S)
```

Output



Task # 5

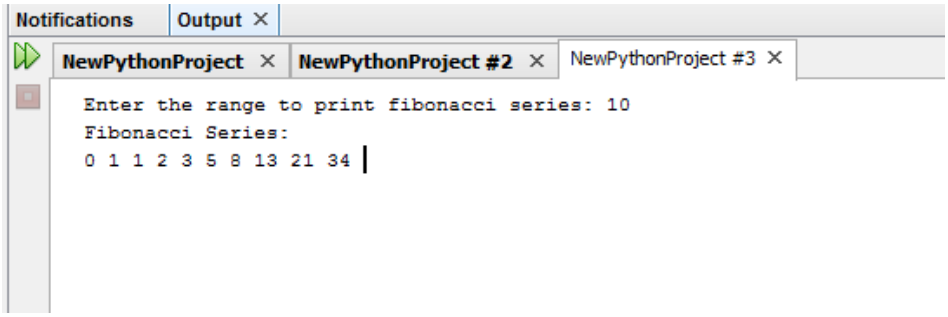
Write a recursive function to compute N Fibonacci number. Test and trace for N = 6 is 8. We remember that a Fibonacci number can be recursively defined as:

$$= \quad -1 + \quad -2 \text{ for} \geq 2, \text{ where} \quad 0 = 0, \quad 1 = 1.$$

Code

```
def fibonacci(range):
  if range<2:
     return range
  else:
     return fibonacci(range-1)+fibonacci(range-2)


f=int(input('Enter the range to print fibonacci series: '))
print('Fibonacci Series: ')
for i in range(0,f):
  c=fibonacci(i)
  print(c,end=' ')
```

Output



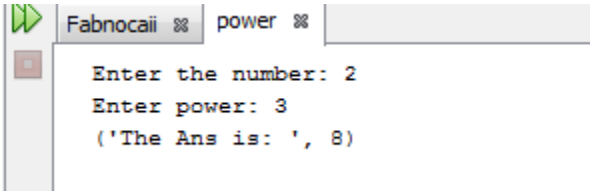
Task # 6

Write a recursive function to compute power of a number (X^p). Test and trace for 4^5? Hint: $4^5 = 4 * 4^4$; $4^0 = 1$.

Code

```
def CalcPower(base,exp):
    if exp == 0:
        return 1;
    elif exp == 1:
        return base;
    else:
        return base * CalcPower(base,exp-1)
number=int(input('Enter the number: '))
power=int(input('Enter power: '))
result=CalcPower(number,power)
print('The Ans is: ',result)
```

OUTPUT



Task # 7

Write a recursive function isPalindrome that takes a string and returns true if it is read forwards or backwards. For example, isPalindrome("mom") →true isPalindrome("cat")→false isPalindrome("level")→true

The prototype for the function should be as follows:

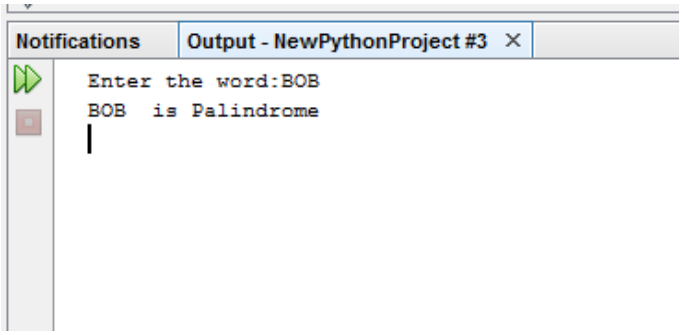
def isPalindrome(string):

Code

```
def CheckPalindrome(string):
    if string == '':
        return True
    else:
        if (ord(string[0]) - ord(string[len(string)-1])) == 0:

            return CheckPalindrome(string[1:len(string)-1])
```

```
        else:
            return False

word=input('Enter the word:')

c=CheckPalindrome(word)
if c==True:
    print(word,' is Palindrome')
else:
    print(word,' is not Palindrome')
```

Output



CSL 411 –Artificial Intelligence Lab

FALL 2017

Lab # 03

Python Programming 3

COURSE INSTRUCTOR: TARIQ SIDDIQUE

LAB ENGINEER: TARWAN KUMAR

DEPARTMENT OF COMPUTER SCIENCE

BAHRIA UNIVERSITY, KARACHI CAMPUS

Task # 1

Write a python application with the following prototypes that returns the user's body mass index
Design then implement a class to represent a **Flight**. A Flight has a *flight number*, a *source*, a *destination* and a *number of available seats*. The class should have:

a. A **constructor** to initialize the 4 instance variables. You have to shorten the name of the source and the destination to 3 characters only if it is longer than 3 characters by a call to the method in the 'j' part.

b. An **overloaded constructor** to initialize the *flight number* and the *number of available seats* instance variables only.

(**NOTE:** Initialize the *source* and the *destination* instance variables to empty string, i.e." ")

c. An **overloaded constructor** to initialize the *flight number* instance variable only.

(**NOTE:** Initialize the *source* and the *destination* instance variables to empty string; and the *number of available seats* to zero)

d. One **accessor** method for each one of the 4 instance variables.

e. One **mutator** method for each one of the 4 instance variables **except** the *flight number* instance variable.

f. A **method** **public void reserve(int numberOfSeats)** to reserve seats on the flight.

(**NOTE:** You have to check that there is enough number of seats to reserve)

g. A **method** **public void cancel(int numberOfSeats)** to cancel one or more reservations

h. A **toString** method to easily return the flight information as follows:

i. An **equals** method to compare 2 flights.

(**NOTE:** 2 Flights considered being equal if they have the same flight number)

j. The following method:

Create objects of the *Flight* class you wrote and try to use all the methods you wrote.

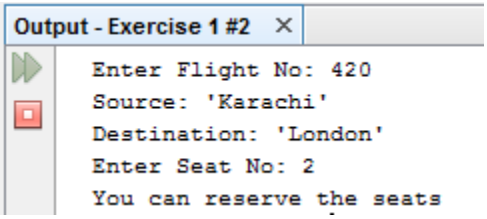
Code

```
class Flight:
    name = ''
    seats = 0

    def __init__(self, fno, source, destination, nseats):
        self.fno = fno
        self.source = source
```

```
    self.destination = destination
    self.nseats = nseats

def mutators(self, a, b, c, d):
    self.fno = a;
    self.source = b;
    self.destination = c;
    self.nseats = d;

def shortAndCapital(self, name):

    if len(name) <= 3:
        return self.name.upper()
    else:
        return self.name[0:3].upper()

def getfno(self):
    self.__fno = input('Enter Flight No: ')
    return self.__fno

def getnseats(self):
    self.__nseats = int(input('Enter Seat No: '))
    self.reserve(self.__nseats)
    return self.__nseats

def getsource(self):
    self.__source = input('Source: ')
    self.__source = self.shortAndCapital(self.__source)
    return self.__source

def getdestination(self):
    self.__destination = input('Destination: ')
    self.__destination = self.shortAndCapital(self.__destination)
    return self.__destination

def reserve(self, seats):
    if seats < self.nseats:
        print('You can reserve the seats')
        self.nseats = self.seats - seats;
    else:
        print('Seats are not available')

def __cancel__(self, seats):
    nseats = seats + seats

def equal(self, flight1, flight2):
    if (self.flight1 == self.flight2):
        print('Both Flights are equal')
    else:
        print('Both flights are not equal')

def ToString(self):
    print('\nFlight No: ', self.getfno(), '\nFrom: ', self.getsource(), '\nTo: ', self.getdestination(),
```

```
        '\nAvailable Seats: ', self.getnseats())


f1 = Flight(1, 'KARACHI', 'LAHORE', 6)

f1.getfno()
f1.getsource()
f1.getdestination()
f1.getnseats()
f1.ToString()
```

Output



Task # 2

MyPython Coffee Outlet runs a catalog business. It sells only one type of coffee beans. The company sells the coffee in 2-lb bags only and the price of a single 2-lb bag is $5.50. when a customer places an order, the company ships the order in boxes. The boxes come in 3 sizes with 3 different costs:

Large box Medium box Small box

capacity 20 bags 10 bags 5 bags

cost $1.80 $1.00 $0.60

The order is shipped using the least number boxes. For example, the order of 52 bags will be shipped in 2 boxes: 2 large boxes, 1medium and 1 small.

Develop an application by using the above that computes the total cost of an order.

Sample out put:

Number of Bags Ordered: 52

The Cost of Order: $ 286.00

Boxes Used:

2 Large - $3.60

1 Medium - $1.00

1 Small - $0.60

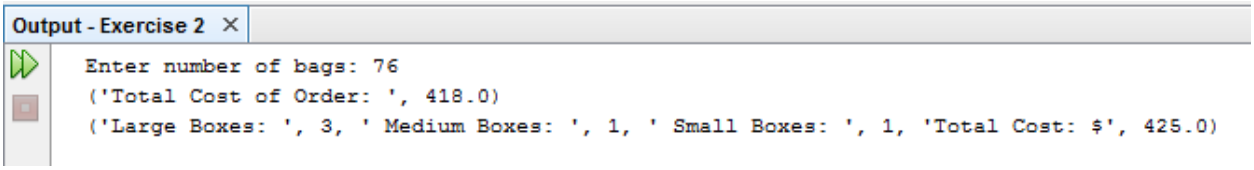
Your total cost is: $ 291.20

Code

```
class Boxes:

    def __init__(self,boxes):
        self.boxes=boxes

    def price(self):
        price=5.50
        total = self.boxes*price
        return total
    def box(self,total):
        c=self.boxes%20
        big=self.boxes/20
        x=big
        d=c%10
        medium=c/10
        big*=1.80
        y=medium
```

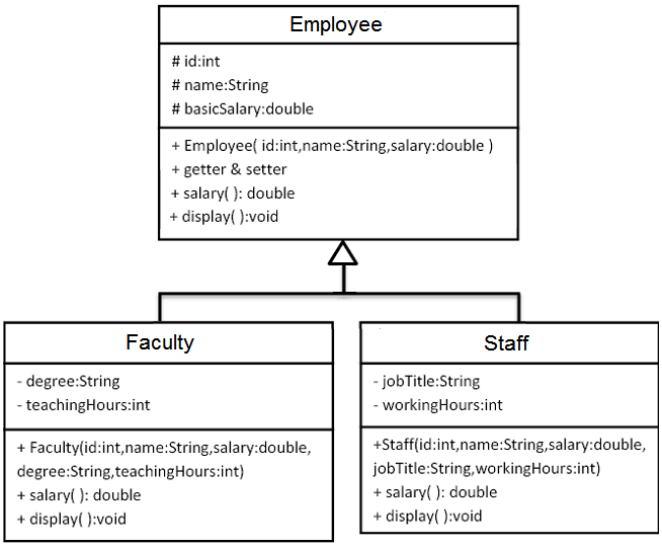
```
        medium*=1.00
        e=d%5
        small=d/5
        z=small
        small*=0.60
        total=total+small+medium+big
        print('Large Boxes: ',int(x),' Medium Boxes: ',int(y),' Small Boxes: ',int(z),'Total Cost: $',total)


a=int(input('Enter number of bags: '))
b1=Boxes(a)
tprice=b1.price()
print('Total Cost of Order: ',tprice)
b1.box(tprice)
```

Output



Task # 3

The class hierarchy of this exercise looks like the following.



Class Employee

Employee: Parameterized constructor.

getter & setter: Create the get and set method to each attribute.

display: Displays Employee ID and Name in the following format:

ID: 1234 – Name: XYZ - Salary: 70000.00

Class Faculty

Faculty: Parameterized constructor.

getter & setter: Create the get and set method to each attribute.

display: Shows the Faculty information in the following format:

ID: 1234 – Name: XYZ – Degree: PhD - Salary: 70000.00

salary: Calculate salary based on the following formula :

Salary= basicSalary + teachingHours * 1000

Class Staff

Staff: Parameterized constructor.

getter & setter: Create the get and set method to each attribute.

display: Shows the Staff information in the following format:

ID: 1234 – Name: XYZ – JobTitle: Registrar - Salar: 70000.00

salary: Calculate salary based on the following formula :

Salary= basicSalary

Unless workingHours > 8, Then

Salary=basicSalary+ (basicSalary *25 /100)

Once you are done implementing the classes, you need to test your work using the class with main method to do the following:

1. Create an instance of class **Employee**, an instance of class **Faculty** and an instance of class **Staff** with proper values for the attributes.

2. Display the information for each instance using display() method.

3. Show the salary for Faculty and Staff.

Run example:

ID: 43221 - Name: Ali - Salary: 70000.0 PKR

ID: 71245 - Name: Majed - Degree: PhD - Salary:182000.0 PKR

ID: 81234 - Name: Nasser - JobTitle: Registrar - Salary:125000.0 PKR

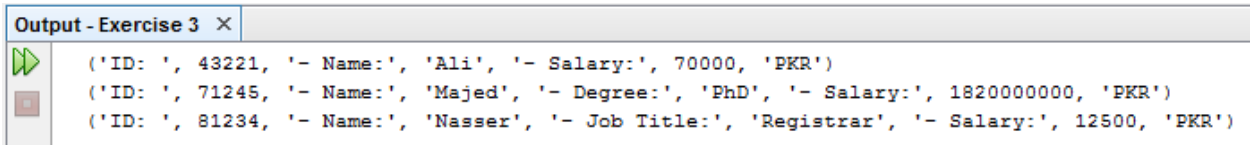
Code

```
class Employee:
    def __init__(self,id,name,bsalary):
        self.id=id
        self.name=name
        self.bsalary=bsalary
    def setid(self,i):
        self.id=i
    def setname(self,n):
        self.name=n
    def setsal(self,s):
        self.bsalary=s
    def getid(self):
        return self.id
    def getname(self):
        return self.name
    def getsal(self):
        return self.bsalary
    def display(self):
        print('ID: ',self.id,'- Name:',self.name,'- Salary:',self.bsalary,'PKR')

class Faculty(Employee):
    def __init__(self,id,name,bsalary,degree,thours):
        Employee.__init__(self,id,name,bsalary)
        self.degree=degree
        self.thours=thours
    def setid(self,i):
        self.id=i
    def setname(self,n):
        self.name=n
```

```
    def setsal(self,s):
        self.bsalary=s
    def setthours(self,h):
        self.thours=h
    def setdegree(self,d):
        self.degree=d
    def getid(self):

        return self.id
    def getname(self):
     return self.name
    def getsal(self):
     return self.bsalary
    def getthours(self):
     return self.thours
    def getdegree(self):
     return self.degree
    def calcSal(self):
        return self.bsalary*self.thours*1000
    def display(self):
       print('ID: ',self.id,'- Name:',self.name,'- Degree:',self.degree,'- Salary:',self.calcSal(),'PKR')

class Staff(Employee):
    def __init__(self,id,name,bsalary,jtitle,whours):
        Employee.__init__(self,id,name,bsalary)
        self.jtitle=jtitle
        self.whours=whours
    def setid(self,i):
        self.id=i
    def setname(self,n):
        self.name=n
    def setsal(self,s):
        self.bsalary=s
    def setwhours(self,h):
        self.whours=h
    def setjtitle(self,t):
        self.jtitle=t
    def getid(self):

        return self.id
    def getname(self):
     return self.name
    def getsal(self):
     return self.bsalary
    def getwhours(self):
     return self.whours
    def getjtitle(self):
     return self.jtitle
    def calcSal(self):
        if self.whours>8:
            return self.bsalary+(self.bsalary*25/100)
        else:
          return self.bsalary
```

```
    def display(self):
        print('ID: ',self.id,'- Name:',self.name,'- Job Title:',self.jtitle,'- Salary:',self.calcSal(),'PKR')

e1=Employee(43221,'Ali',70000)
e1.display()
f1=Faculty(71245,'Majed',10,'PhD',182000)
f1.display()
s1=Staff(81234,'Nasser',10000,'Registrar',125000)
s1.display()
```

Output



CSL 411 –Artificial Intelligence Lab

FALL 2017

Lab # 04

BFS and DFS

COURSE INSTRUCTOR: TARIQ SIDDIQUE

LAB ENGINEER: TARWAN KUMAR

DEPARTMENT OF COMPUTER SCIENCE

BAHRIA UNIVERSITY, KARACHI CAMPUS

Task # 1

Write Python programs to implement 8 puzzle

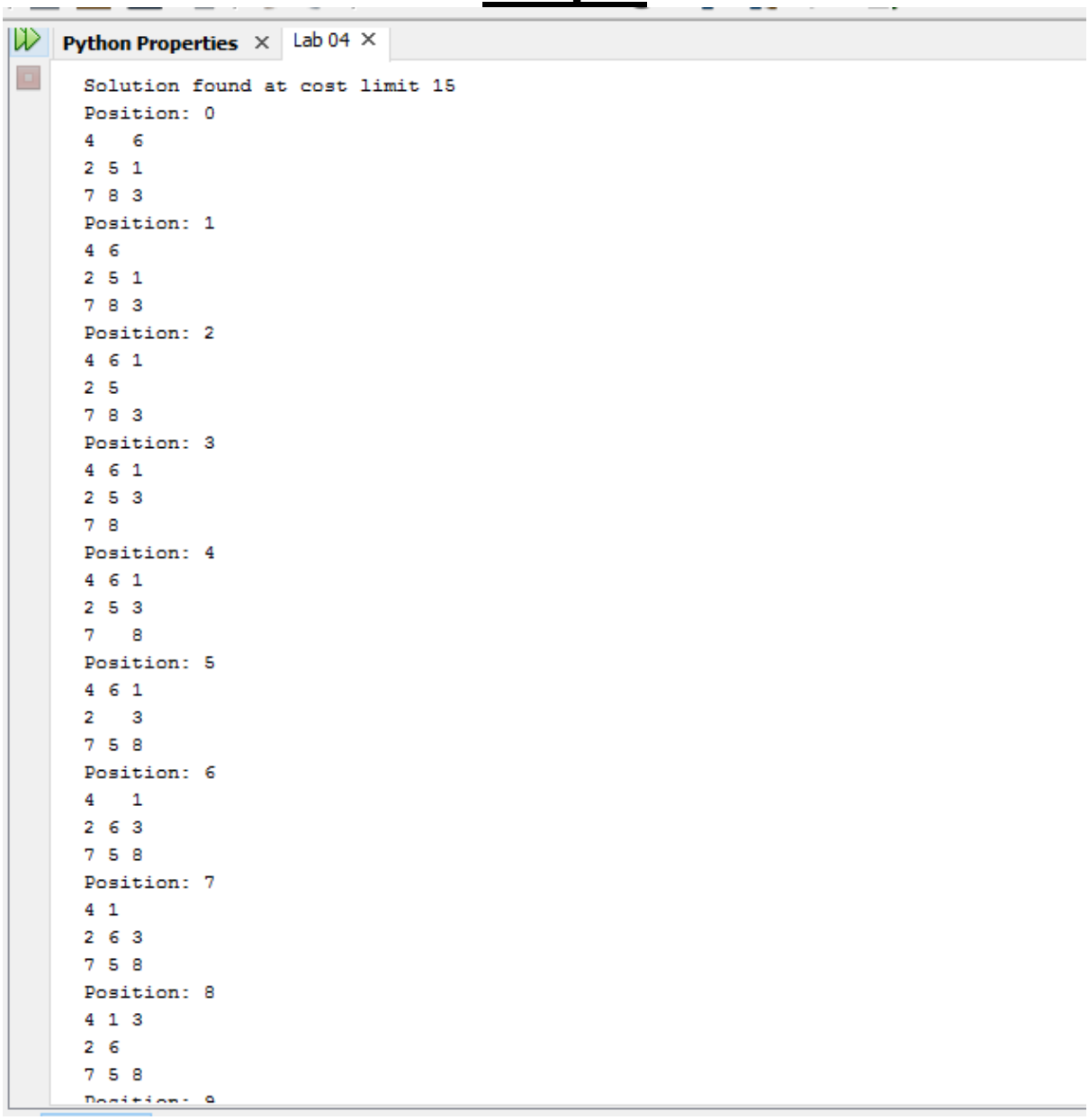
Code

```
#!/usr/bin/python
from copy import deepcopy
from time import sleep
from pprint import pprint
import sys
import math
PUZZLE_SIZE = 3
MAX_COST_CUTOFF = 31
DEBUG_MODE = False
SIMULATION_MODE = False
Infinity = float("inf")
total_nodes_generated = 0
get_2dindex = lambda state, n: (state.index(n) / PUZZLE_SIZE, state.index(n) % PUZZLE_SIZE)
def get_ordered(node):
	ordered = []
	for i in range(0, PUZZLE_SIZE):
		temp = deepcopy(node[i*PUZZLE_SIZE: (i+1)*PUZZLE_SIZE])
		if i%2 == 1: temp.reverse()
		ordered += temp
	ordered.remove(0)
	return ordered

def get_cycle(startnode, endnode, key, cycle):
	if key in cycle: return cycle
	cycle.append(key)
	if startnode.index(key) == endnode.index(key): return cycle
	cycle = get_cycle(startnode, endnode, endnode[startnode.index(key)], cycle)
	return cycle

def check_solvability(startnode, endnode):
	startnode = get_ordered(startnode)
	endnode = get_ordered(endnode)
	cycles = []
	actual_count = 0
	check = dict(zip(startnode, [False for i in range(0, PUZZLE_SIZE * PUZZLE_SIZE)]))
	for item in check.keys():
		if check[item] == False:
			cycle = get_cycle(startnode, endnode, item, [])
			if cycle != []:
				cycles.append(cycle)
				for x in cycle: check[x] = True
	if DEBUG_MODE == True:
		print 'DEBUG: Permutation cycles: ', filter(lambda x: len(x) != 1, cycles)
	for cycle in cycles:
		if len(cycle) == 2: actual_count += 1
		if len(cycle) > 2: actual_count += len(cycle) - 1
	if actual_count % 2 == 0: return True
	return False
def get_next_states(state):
	next_states = []
```

```
        x, y = get_2dindex(state, 0)
        next = filter(lambda x: x[0] >= 0 and x[1] >= 0 and x[0] < PUZZLE_SIZE and x[1] < PUZZLE_SIZE,
                        ((x, y + 1), (x, y - 1), (x + 1, y), (x - 1, y)))

        for x, y in next:
                newindex = x * PUZZLE_SIZE + y
                oldindex = state.index(0)

                newstate = deepcopy(state)
                newstate[oldindex] = newstate[newindex]
                newstate[newindex] = 0
                next_states.append(newstate)

        return next_states

def get_manhattan_distance(current, next):
        total = 0
        for point in current:
                if point == 0: continue
                x1, y1 = get_2dindex(current, point)
                x2, y2 = get_2dindex(next, point)
                total += abs(x1 - x2) + abs(y1 - y2)
        return total

def pretty_print(solution):
        for nodeindex, node in enumerate(solution):
                print 'Position:', nodeindex,
                for index, item in enumerate(node):
                        if index % PUZZLE_SIZE == 0: print
                        if item == 0: print ' ',
                        else: print item,
                print


def ida_star(startnode, endnode):
        cost_cutoff = get_manhattan_distance(startnode, endnode)

        while cost_cutoff < MAX_COST_CUTOFF:
                solution, updated_cost_cutoff = dfs(startnode, 0, cost_cutoff, [startnode])
                if solution != None: return solution, updated_cost_cutoff
                if updated_cost_cutoff == Infinity: return None
                if DEBUG_MODE == True:
                        print 'DEBUG: No solution for f(n) =', cost_cutoff
                        print 'DEBUG: Trying for f(n) = ', updated_cost_cutoff
                cost_cutoff = updated_cost_cutoff

def dfs(node, cost_from_root, cost_cutoff, path):
        global total_nodes_generated
        total_nodes_generated += 1
        minimum_cost = cost_from_root + get_manhattan_distance(node, endnode)

        if DEBUG_MODE == True:
                print 'DEBUG: DFS for node: ', node
                print 'DEBUG: Cost (depth) from root: ', cost_from_root
                print 'DEBUG: Parent\'s cost limit:', cost_cutoff
                print 'DEBUG: Curent cost limit i.e. f(n): ', minimum_cost
                print 'DEBUG: Path so far:'; pprint(path)
        if SIMULATION_MODE == True: sleep(1)
```

```
        if minimum_cost > cost_cutoff: return None, minimum_cost
        if node == endnode: return path, cost_cutoff

        next_cost_cutoff = Infinity
        next_states = get_next_states(node)
        if DEBUG_MODE == True: print 'DEBUG: Node', node, 'has following child nodes: ', next_states
        for next_node in next_states:
                solution, new_cost_cutoff = dfs(next_node, cost_from_root + 1, cost_cutoff, path + [next_node])
                if solution != None: return solution, new_cost_cutoff
                next_cost_cutoff = min(next_cost_cutoff, new_cost_cutoff)

        return None, next_cost_cutoff

if __name__ == '__main__':
        # 8 puzzle
#       startnode = [7, 2, 4, 5, 0, 6, 8, 3, 1]
#       endnode = [7, 4, 6, 5, 2, 0, 8, 3, 1]
#       endnode = [7, 4, 6, 8, 5, 3, 0, 1, 2]
#       endnode = [7, 4, 6, 5, 2, 1, 0, 8, 3]
        # takes a lot of time
#       endnode = [0, 1, 2, 3, 4, 5, 6, 7, 8]
        # this node is unsolvable
#       endnode = [1, 2, 4, 5, 0, 6, 8, 3, 7]

        # the supposed worst (!) case scenario, but actually its not :P
#       startnode = [8, 6, 7, 2, 5, 4, 3, 0, 1]
#       endnode = [6, 4, 7, 8, 5, 0, 3, 2, 1]

        startnode = [4, 0, 6, 2, 5, 1, 7, 8, 3]
        endnode = [1, 2, 3, 4, 5, 6, 7, 8, 0]

        # 15 puzzle
#       startnode = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0]
#       endnode = [1, 2, 3, 4, 5, 6, 7, 8, 0, 15, 12, 14, 13, 9, 11, 10]

#       PUZZLE_SIZE = int(math.sqrt(len(startnode)))
        if check_solvability(startnode, endnode) == False:
                print 'The goal is not reachable'
                sys.exit(1)
        answer = ida_star(startnode, endnode)
        if answer != None:
                print 'Solution found at cost limit', answer[1]
                pretty_print(answer[0])
        else: print 'No solution found within maximum cost limit', MAX_COST_CUTOFF
        print 'Total nodes generated:', total_nodes_generated
```

Output


Python Properties
Lab 04
Solution found at cost limit 15
Position: 0
4 6
2 5 1
7 8 3
Position: 1
4 6
2 5 1
7 8 3
Position: 2
4 6 1
2 5
7 8 3
Position: 3
4 6 1
2 5 3
7 8
Position: 4
4 6 1
2 5 3
7 8
Position: 5
4 6 1
2 3
7 5 8
Position: 6
4 1
2 6 3
7 5 8
Position: 7
4 1
2 6 3
7 5 8
Position: 8
4 1 3
2 6
7 5 8

: Output

Python Properties × Lab 04 ×

```
Position: 9
4 1 3
2   6
7 5 8
Position: 10
4 1 3
  2 6
7 5 8
Position: 11
  1 3
4 2 6
7 5 8
Position: 12
1   3
4 2 6
7 5 8
Position: 13
1 2 3
4   6
7 5 8
Position: 14
1 2 3
4 5 6
7   8
Position: 15
1 2 3
4 5 6
7 8
Total nodes generated: 996
```

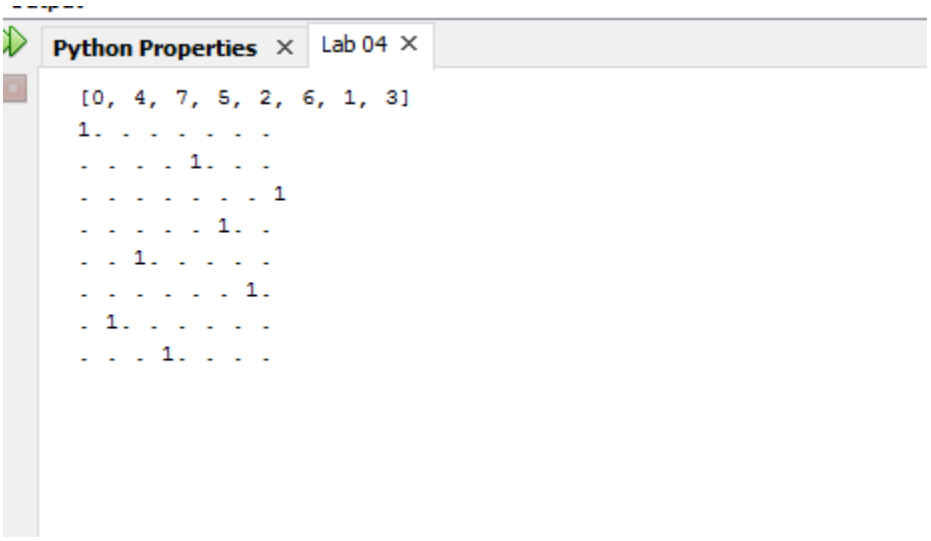
Task # 2

Write Python programs to implement 8 queen problem

Code

```
BOARD_SIZE = 8
class BailOut(Exception):
    pass
def validate(queens):
    left = right = col = queens[-1]
    for r in reversed(queens[:-1]):
        left, right = left-1, right+1
        if r in (left, col, right):
            raise BailOut

def add_queen(queens):
    for i in range(BOARD_SIZE):
        test_queens = queens + [i]
        try:
            validate(test_queens)
            if len(test_queens) == BOARD_SIZE:
                return test_queens
            else:
                return add_queen(test_queens)
        except BailOut:
            pass
    raise BailOut

queens = add_queen([])
print (queens)
print ("\n".join(". "*q + "1" + ". "*(BOARD_SIZE-q-1) for q in queens))
```

Output



CSL 411 –Artificial Intelligence Lab

FALL 2017

Lab # 05

Greedy A-star

COURSE INSTRUCTOR: TARIQ SIDDIQUE

LAB ENGINEER: TARWAN KUMAR

DEPARTMENT OF COMPUTER SCIENCE

BAHRIA UNIVERSITY, KARACHI CAMPUS

Task # 1

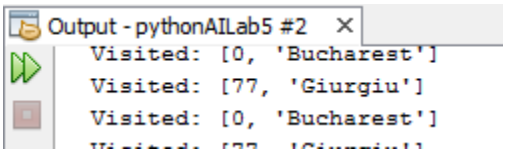
Write Python program to implement Greedy Best First Algorithm for following rout finding problem from PULA to BUZET.

Code

```
import queue

weight = {
'Arad':set(['Sibiu', 'Zerind', 'Timisoara']),
'Bucharest':set(['Urziceni', 'Fagaras', 'Pitesti', 'Giurgiu']),
'Craiova':set(['Dobreta', 'Rimnicu Vilcea', 'Pitesti']),
'Dobreta':set(['Craiova', 'Mehadia']),
'Eforie':set(['Hirsova']),
'Fagaras':set(['Sibiu', 'Bucharest']),
'Giurgiu':set(['Bucharest']),
'Hirsova':set(['Urziceni', 'Eforie']),
'Lasi':set(['Neamt', 'Vaslui']),
'Lugoj':set(['Mehadia', 'Timisoara']),
'Mehadia':set(['Lugoj', 'Dobreta']),
'Neamt':set(['Lasi']),
'Oradea':set(['Zerind', 'Sibiu']),
'Pitesti':set(['Bucharest', 'Rimnicu Vilcea', 'Craiova']),
'Rimnicu Vilcea':set(['Sibiu', 'Pitesti', 'Craiova']),
'Sibiu':set(['Fagaras', 'Rimnicu Vilcea', 'Arad', 'Oradea']),
'Timisoara':set(['Lugoj', 'Arad']),
'Urziceni':set(['Bucharest', 'Hirsova', 'Vaslui']),
'Vaslui':set(['Lasi', 'Urziceni']),
'Zerind':set(['Oradea', 'Arad'])
}

distance = {
'Arad': 366,
'Bucharest': 0,
'Craiova': 160,
'Dobreta': 242,
'Eforie': 161,
'Fagaras': 178,
'Giurgiu': 77,
'Hirsova': 151,
'Lasi': 226,
'Lugoj': 244,
'Mehadia': 241,
'Neamt': 234,
'Oradea': 380,
'Pitesti': 98,
'Rimnicu Vilcea': 193,
'Sibiu': 253,
'Timisoara': 329,
'Urziceni': 80,
'Vaslui': 199,
'Zerind': 374
}

distanceVal = {
```

```
366:'Arad',
0:'Bucharest',
160:'Craiova',
242:'Dobreta',
161:'Eforie',
178:'Fagaras',
77:'Giurgiu',
151:'Hirsova',
226:'Lasi',
244:'Lugoj',
241:'Mehadia',
234:'Neamt',
380:'Oradea',
98:'Pitesti',
193:'Rimnicu Vilcea',
253:'Sibiu',
329:'Timisoara',
80:'Urziceni',
199:'Vaslui',
374:'Zerind',
}

#Greedy BFS Search
def shortPath(graph, start, goal):
    Queue = queue.PriorityQueue()
    Queue.put([distance[start], start])

    while not Queue.empty():
        item = Queue.get();
        print("Visited: " + str(item))
        for paths in weight[item[1]]:
            Queue.put([distance[paths], paths])

print(list(shortPath (weight, 'Arad', 'Bucharest')))
```

Output



Task # 2

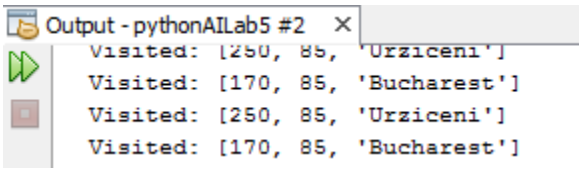
Write Python program to implement A* Search Algorithm for following rout finding problem from PULA to BUZET.

Code

```
import queue

graph = {
'Arad':[['Sibiu', 140], ['Zerind', 75], ['Timisoara', 118]],
'Bucharest':[['Urziceni', 85], ['Fagaras', 211], ['Pitesti', 101], ['Giurgiu', 90]],
'Craiova':[['Dobreta', 120], ['Rimnicu Vilcea', 146], ['Pitesti', 138]],
'Dobreta':[['Craiova', 120], ['Mehadia', 75]],
'Eforie':[['Hirsova', 86]],
'Fagaras':[['Sibiu', 99], ['Bucharest', 211]],
'Giurgiu':[['Bucharest', 90]],
'Hirsova':[['Urzeceni', 98], ['Eforie', 86]],
'Lasi':[['Neamt', 87], ['Vaslui', 92]],
'Lugoj':[['Mehadia', 70], ['Timisoara', 111]],
'Mehadia':[['Lugoj', 70], ['Dobreta', 75]],
'Neamt':[['Lasi', 87]],
'Oradea':[['Zerind', 71], ['Sibiu', 151]],
'Pitesti':[['Bucharest', 101], ['Rimnicu Vilcea', 97], ['Craiova', 138]],
'Rimnicu Vilcea':[['Sibiu', 80], ['Pitesti', 97], ['Craiova', 146]],
'Sibiu':[['Fagaras', 99], ['Rimnicu Vilcea', 80], ['Arad', 140], ['Oradea', 151]],
'Timisoara':[['Lugoj', 111], ['Arad', 118]],
'Urziceni':[['Bucharest', 85], ['Hirsova', 98], ['Vaslui', 142]],
'Vaslui':[['Lasi', 92], ['Urziceni', 142]],
'Zerind':[['Oradea', 71], ['Arad', 75]]
}

#Heuristic Values
distance = {
'Arad': 366,
'Bucharest': 0,
'Craiova': 160,
'Dobreta': 242,
'Eforie': 161,
'Fagaras': 178,
'Giurgiu': 77,
'Hirsova': 151,
'Lasi': 226,
'Lugoj': 244,
'Mehadia': 241,
'Neamt': 234,
'Oradea': 380,
'Pitesti': 98,
'Rimnicu Vilcea': 193,
'Sibiu': 253,
'Timisoara': 329,
'Urziceni': 80,
'Vaslui': 199,
'Zerind': 374
}
```

```
distanceValue = {
366:'Arad',
0:'Bucharest',
160:'Craiova',
242:'Dobreta',
161:'Eforie',
178:'Fagaras',
77:'Giurgiu',
151:'Hirsova',
226:'Lasi',
244:'Lugoj',
241:'Mehadia',
234:'Neamt',
380:'Oradea',
98:'Pitesti',
193:'Rimnicu Vilcea',
253:'Sibiu',
329:'Timisoara',
80:'Urziceni',
199:'Vaslui',
374:'Zerind',
}

def asterikShortPath(graph, start, goal):
    Queue = queue.PriorityQueue()
    Queue.put([0 , distance[start], start])
    while not Queue.empty():
        item = Queue.get();
        print("Visited: " + str(item))
        for paths in graph[distanceValue[distance[item[2]]]]:
            if paths == goal:
                yield paths
            else:
                Queue.put([(distance[paths[0]] + paths[1] + item[1]), paths[1], paths[0]])

print(list(asterikShortPath(graph, 'Arad', 'Bucharest')))
```

Output



CSL 411 –Artificial Intelligence Lab

FALL 2017

Lab # 06

Hill Climbing and Simulated Annealing Algorithms

COURSE INSTRUCTOR: TARIQ SIDDIQUE

LAB ENGINEER: TARWAN KUMAR

DEPARTMENT OF COMPUTER SCIENCE

BAHRIA UNIVERSITY, KARACHI CAMPUS

Task # 1

Write Python programs to implement Hill Climbing and Simulated Annealing Algorithms. Apply these algorithms to the traveling salesman problem below.

Code

```
from __future__ import print_function
import math
import random
from simanneal import Annealer
def distance(a, b):
"""Calculates distance between two latitude-longitude coordinates."""
R = 3963 # radius of Earth (miles)
lat1, lon1 = math.radians(a[0]), math.radians(a[1])
lat2, lon2 = math.radians(b[0]), math.radians(b[1])
return math.acos(math.sin(lat1) * math.sin(lat2) +
math.cos(lat1) * math.cos(lat2) * math.cos(lon1 - lon2)) * R
class TravellingSalesmanProblem(Annealer):
"""Test annealer with a travelling salesman problem.
"""
# pass extra data (the distance matrix) into the constructor
def __init__(self, state, distance_matrix):
self.distance_matrix = distance_matrix
super(TravellingSalesmanProblem, self).__init__(state) # important!
def move(self):
"""Swaps two cities in the route."""
a = random.randint(0, len(self.state) - 1)
b = random.randint(0, len(self.state) - 1)
self.state[a], self.state[b] = self.state[b], self.state[a]
def energy(self):
"""Calculates the length of the route."""
e = 0
for i in range(len(self.state)):
e += self.distance_matrix[self.state[i-1]][self.state[i]]
return e
if __name__ == '__main__':
# latitude and longitude for the twenty largest Pakistani cities
cities = {
'Karachi': (40.72, 74.00),
```

```
'Lahore': (34.05, 118.25),
'Multan': (41.88, 87.63),
'Faisalabad': (29.77, 95.38),
'Peshawar': (33.45, 112.07),
'Hyderabad': (39.95, 75.17),
'Islamabad': (29.53, 98.47),
'Quetta': (32.78, 96.80),
'Sukkur': (32.78, 117.15),
'Murree': (37.30, 121.87),
'Dera Ismail Khan': (42.33, 83.05),
'Larkana': (37.78, 122.42),
'Jacobabad': (30.32, 81.70),
'Gujranwala': (39.78, 86.15),
'Sialkot': (30.27, 97.77),
'Bahwalpur': (39.98, 82.98),
'Jhang': (32.75, 97.33),
'Chitral': (35.23, 80.85),
'Gawadar': (35.12, 89.97),
'Dadu': (39.28, 76.62) }
# initial state, a randomly-ordered itinerary
init_state = list(cities.keys())
random.shuffle(init_state)
# create a distance matrix
distance_matrix = {}
for ka, va in cities.items():
distance_matrix[ka] = {}
for kb, vb in cities.items():
if kb == ka:
distance_matrix[ka][kb] = 0.0
else:
distance_matrix[ka][kb] = distance(va, vb)
tsp = TravellingSalesmanProblem(init_state, distance_matrix)
# since our state is just a list, slice is the fastest way to copy
tsp.copy_strategy = "slice"
state, e = tsp.anneal()
while state[0] != 'Karachi':
state = state[1:] + state[:1] # rotate NYC to start
print("%i mile route:" % e)
for city in state:
print("\t", city)
```

Output

Task # 02

Write Python programs to implement Hill Climbing and Simulated Annealing Algorithms. Apply these algorithms to the traveling salesman problem below.

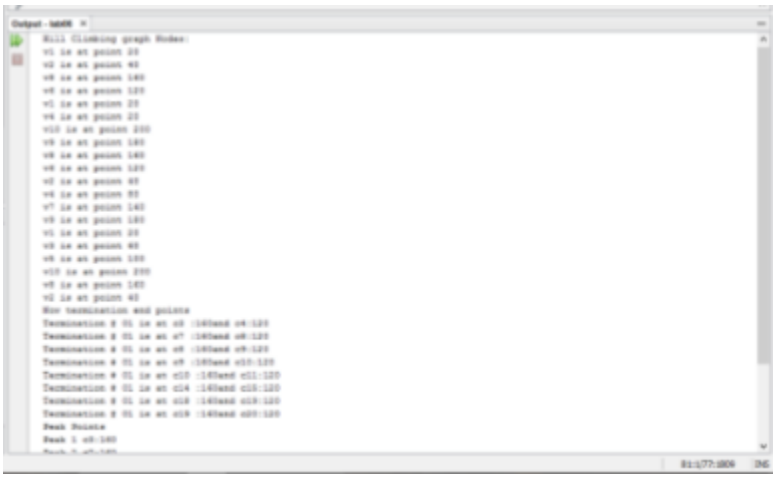
Code

```
__author__ = "Rozina"
__date__ = "$Mar 25, 2017 9:15:21 PM$"
if __name__ == "__main__":
v1=20
v2=40
v3=60
v4=80
v5=100
v6=120
v7=140
v8=160
v9=180
v10=200
c1=20
c2=40
c3=160
c4=120
c5=20
c6=20
c7=200
c8=180
```

```
c9=160
c10=120
c11=40
c12=80
c13=140
c14=180
c15=20
c16=60
c17=100
c18=200
c19=160
c20=40
print("Hill Climbing graph Nodes:")
print("v1 is at point "+str(c1))
print("v2 is at point "+str(c2))
print("v8 is at point "+str(c3))
print("v6 is at point "+str(c4))
print("v1 is at point "+str(c5))
print("v4 is at point "+str(c6))
print("v10 is at point "+str(c7))
print("v9 is at point "+str(c8))
print("v8 is at point "+str(c9))
print("v6 is at point "+str(c10))
print("v2 is at point "+str(c11))
print("v4 is at point "+str(c12))
print("v7 is at point "+str(c13))
print("v9 is at point "+str(c14))
print("v1 is at point "+str(c15))
print("v3 is at point "+str(c16))
print("v5 is at point "+str(c17))
print("v10 is at point "+str(c18))
print("v8 is at point "+str(c19))
print("v2 is at point "+str(c20))
print("Now termination end points")
print("Termination # 01 is at c3 :"+str(c3)+"and c4:"+str(c4))
print("Termination # 01 is at c7 :"+str(c3)+"and c8:"+str(c4))
print("Termination # 01 is at c8 :"+str(c3)+"and c9:"+str(c4))
print("Termination # 01 is at c9 :"+str(c3)+"and c10:"+str(c4))
print("Termination # 01 is at c10 :"+str(c3)+"and c11:"+str(c4))
print("Termination # 01 is at c14 :"+str(c3)+"and c15:"+str(c4))
print("Termination # 01 is at c18 :"+str(c3)+"and c19:"+str(c4))
```

```
print("Termination # 01 is at c19 :"+str(c3)+"and c20:"+str(c4))
print("Peak Points")
print("Peak 1 c3:"+str(c3))
print("Peak 2 c7:"+str(c3))
print("Peak 3 c8:"+str(c3))
print("Peak 4 c9:"+str(c3))
print("Peak 5 c10:"+str(c3))
print("Peak 6 c14:"+str(c3))
print("Peak 7 c18:"+str(c3))
print("Peak 8 c19:"+str(c3))
```

Output



CSL 411 –Artificial Intelligence Lab

FALL 2017

Lab # 07

Genetic Algorithm

COURSE INSTRUCTOR: TARIQ SIDDIQUE

LAB ENGINEER: TARWAN KUMAR

DEPARTMENT OF COMPUTER SCIENCE

BAHRIA UNIVERSITY, KARACHI CAMPUS

Task # 1

Genetic Algorithm to find fitness of function

Code

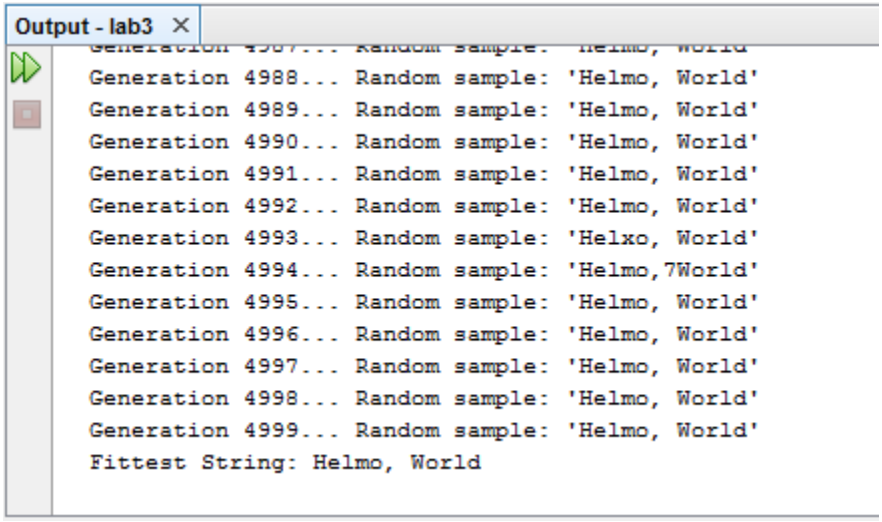
```
import random

#
# Global variables
# Setup optimal string and GA input variables.
#

OPTIMAL     = "Hello, World"
DNA_SIZE    = len(OPTIMAL)
POP_SIZE    = 20
GENERATIONS = 5000

#
# Helper functions
# These are used as support, but aren't direct GA-specific functions.
#

def weighted_choice(items):
  """
  Chooses a random element from items, where items is a list of tuples in
  the form (item, weight). weight determines the probability of choosing its
  respective item. Note: this function is borrowed from ActiveState Recipes.
  """
  weight_total = sum((item[1] for item in items))
  n = random.uniform(0, weight_total)
  for item, weight in items:
    if n < weight:
      return item
    n = n - weight
  return item

def random_char():
  """
  Return a random character between ASCII 32 and 126 (i.e. spaces, symbols,
  letters, and digits). All characters returned will be nicely printable.
  """
  return chr(int(random.randrange(32, 126, 1)))

def random_population():
  """
  Return a list of POP_SIZE individuals, each randomly generated via iterating
  DNA_SIZE times to generate a string of random characters with random_char().
  """
  pop = []
```

```
  for i in range(POP_SIZE):
    dna = ""
    for c in range(DNA_SIZE):
      dna += random_char()
    pop.append(dna)
  return pop

#
# GA functions
# These make up the bulk of the actual GA algorithm.
#

def fitness(dna):
  """
  For each gene in the DNA, this function calculates the difference between
  it and the character in the same position in the OPTIMAL string. These values
  are summed and then returned.
  """
  fitness = 0
  for c in range(DNA_SIZE):
    fitness += abs(ord(dna[c]) - ord(OPTIMAL[c]))
  return fitness

def mutate(dna):
  """
  For each gene in the DNA, there is a 1/mutation_chance chance that it will be
  switched out with a random character. This ensures diversity in the
  population, and ensures that is difficult to get stuck in local minima.
  """
  dna_out = ""
  mutation_chance = 100
  for c in range(DNA_SIZE):
    if int(random.random()*mutation_chance) == 1:
      dna_out += random_char()
    else:
      dna_out += dna[c]
  return dna_out

def crossover(dna1, dna2):
  """
  Slices both dna1 and dna2 into two parts at a random index within their
  length and merges them. Both keep their initial sublist up to the crossover
  index, but their ends are swapped.
  """
  pos = int(random.random()*DNA_SIZE)
  return (dna1[:pos]+dna2[pos:], dna2[:pos]+dna1[pos:])

#
# Main driver
# Generate a population and simulate GENERATIONS generations.
#
```

```
if __name__ == "__main__":
  # Generate initial population. This will create a list of POP_SIZE strings,
  # each initialized to a sequence of random characters.
  population = random_population()

  # Simulate all of the generations.
  for generation in range(GENERATIONS):
    print ("Generation %s... Random sample: '%s'" % (generation, population[0]))
    weighted_population = []

    # Add individuals and their respective fitness levels to the weighted
    # population list. This will be used to pull out individuals via certain
    # probabilities during the selection phase. Then, reset the population list
    # so we can repopulate it after selection.
    for individual in population:
      fitness_val = fitness(individual)

      # Generate the (individual,fitness) pair, taking in account whether or
      # not we will accidently divide by zero.
      if fitness_val == 0:
        pair = (individual, 1.0)
      else:
        pair = (individual, 1.0/fitness_val)

      weighted_population.append(pair)

    population = []

    # Select two random individuals, based on their fitness probabilites, cross
    # their genes over at a random point, mutate them, and add them back to the
    # population for the next iteration.
    for _ in range(POP_SIZE):
      # Selection
      ind1 = weighted_choice(weighted_population)
      ind2 = weighted_choice(weighted_population)

      # Crossover
      ind1, ind2 = crossover(ind1, ind2)

      # Mutate and add back into the population.
      population.append(mutate(ind1))
      population.append(mutate(ind2))

  # Display the highest-ranked string after all generations have been iterated
  # over. This will be the closest string to the OPTIMAL string, meaning it
  # will have the smallest fitness value. Finally, exit the program.
  fittest_string = population[0]
  minimum_fitness = fitness(population[0])

  for individual in population:
```

```
  ind_fitness = fitness(individual)
  if ind_fitness <= minimum_fitness:
    fittest_string = individual
    minimum_fitness = ind_fitness

print ("Fittest String: %s" % fittest_string)
exit(0)
```

Output



Task # 2

TSP using genetic Algorithm

Code

```
import math
import random
class City:
  def __init__(self, x=None, y=None):
    self.x = None
    self.y = None
    if x is not None:
      self.x = x
    else:
      self.x = int(random.random() * 200)
    if y is not None:
      self.y = y
    else:
      self.y = int(random.random() * 200)
  def getX(self):
    return self.x
  def getY(self):
    return self.y
  def distanceTo(self, city):
    xDistance = abs(self.getX() - city.getX())
    yDistance = abs(self.getY() - city.getY())
    distance = math.sqrt( (xDistance*xDistance) + (yDistance*yDistance) )
    return distance
  def __repr__(self):
    return str(self.getX()) + ", " + str(self.getY())
class TourManager:
  destinationCities = []
  def addCity(self, city):
    self.destinationCities.append(city)
  def getCity(self, index):
    return self.destinationCities[index]
  def numberOfCities(self):
    return len(self.destinationCities)
class Tour:
  def __init__(self, tourmanager, tour=None):
    self.tourmanager = tourmanager
    self.tour = []
    self.fitness = 0.0
    self.distance = 0
    if tour is not None:
      self.tour = tour
    else:
      for i in range(0, self.tourmanager.numberOfCities()):
```

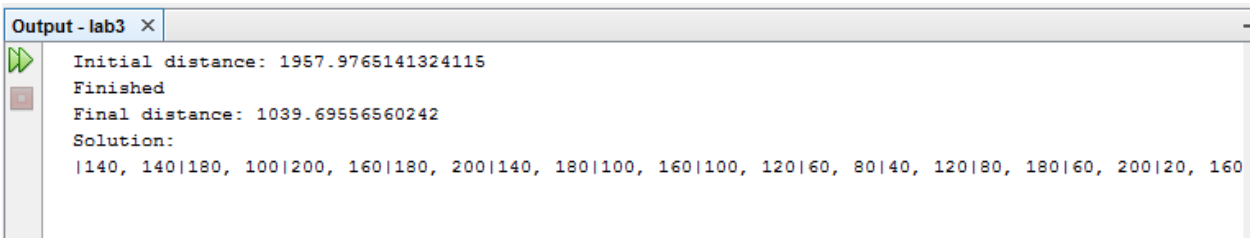
```
            self.tour.append(None)
    def __len__(self):
        return len(self.tour)
    def __getitem__(self, index):
        return self.tour[index]

    def __setitem__(self, key, value):
        self.tour[key] = value
    def __repr__(self):
        geneString = "|"
        for i in range(0, self.tourSize()):
            geneString += str(self.getCity(i)) + "|"
        return geneString
    def generateIndividual(self):
        for cityIndex in range(0, self.tourmanager.numberOfCities()):
            self.setCity(cityIndex, self.tourmanager.getCity(cityIndex))
        random.shuffle(self.tour)
    def getCity(self, tourPosition):
        return self.tour[tourPosition]
    def setCity(self, tourPosition, city):
        self.tour[tourPosition] = city
        self.fitness = 0.0
        self.distance = 0
    def getFitness(self):
        if self.fitness == 0:
            self.fitness = 1/float(self.getDistance())
        return self.fitness
    def getDistance(self):
        if self.distance == 0:
            tourDistance = 0
            for cityIndex in range(0, self.tourSize()):
                fromCity = self.getCity(cityIndex)
                destinationCity = None
                if cityIndex+1 < self.tourSize():
                    destinationCity = self.getCity(cityIndex+1)
                else:
                    destinationCity = self.getCity(0)
                tourDistance += fromCity.distanceTo(destinationCity)
            self.distance = tourDistance
        return self.distance
    def tourSize(self):
        return len(self.tour)
    def containsCity(self, city):
        return city in self.tour
class Population:
    def __init__(self, tourmanager, populationSize, initialise):
        self.tours = []
        for i in range(0, populationSize):
            self.tours.append(None)
        if initialise:
            for i in range(0, populationSize):
```

```
            newTour = Tour(tourmanager)
            newTour.generateIndividual()
            self.saveTour(i, newTour)
    def __setitem__(self, key, value):
        self.tours[key] = value
    def __getitem__(self, index):
        return self.tours[index]

    def saveTour(self, index, tour):
        self.tours[index] = tour
    def getTour(self, index):
        return self.tours[index]
    def getFittest(self):
        fittest = self.tours[0]
        for i in range(0, self.populationSize()):
            if fittest.getFitness() <= self.getTour(i).getFitness():
                fittest = self.getTour(i)
        return fittest
    def populationSize(self):
        return len(self.tours)
class GA:
    def __init__(self, tourmanager):
        self.tourmanager = tourmanager
        self.mutationRate = 0.015
        self.tournamentSize = 5
        self.elitism = True
    def evolvePopulation(self, pop):
        newPopulation = Population(self.tourmanager, pop.populationSize(), False)
        elitismOffset = 0
        if self.elitism:
            newPopulation.saveTour(0, pop.getFittest())
            elitismOffset = 1
        for i in range(elitismOffset, newPopulation.populationSize()):
            parent1 = self.tournamentSelection(pop)
            parent2 = self.tournamentSelection(pop)
            child = self.crossover(parent1, parent2)
            newPopulation.saveTour(i, child)
        for i in range(elitismOffset, newPopulation.populationSize()):
            self.mutate(newPopulation.getTour(i))
        return newPopulation
    def crossover(self, parent1, parent2):
        child = Tour(self.tourmanager)
        startPos = int(random.random() * parent1.tourSize())
        endPos = int(random.random() * parent1.tourSize())
        for i in range(0, child.tourSize()):
            if startPos < endPos and i > startPos and i < endPos:
                child.setCity(i, parent1.getCity(i))
            elif startPos > endPos:
                if not (i < startPos and i > endPos):
                    child.setCity(i, parent1.getCity(i))
        for i in range(0, parent2.tourSize()):
```

```
        if not child.containsCity(parent2.getCity(i)):
          for ii in range(0, child.tourSize()):
            if child.getCity(ii) == None:
              child.setCity(ii, parent2.getCity(i))
              break
      return child
    def mutate(self, tour):
      for tourPos1 in range(0, tour.tourSize()):
        if random.random() < self.mutationRate:
          tourPos2 = int(tour.tourSize() * random.random())

          city1 = tour.getCity(tourPos1)
          city2 = tour.getCity(tourPos2)
          tour.setCity(tourPos2, city1)
          tour.setCity(tourPos1, city2)
    def tournamentSelection(self, pop):
      tournament = Population(self.tourmanager, self.tournamentSize, False)
      for i in range(0, self.tournamentSize):
        randomId = int(random.random() * pop.populationSize())
        tournament.saveTour(i, pop.getTour(randomId))
      fittest = tournament.getFittest()
      return fittest
if __name__ == '__main__':
    tourmanager = TourManager()
    # Create and add our cities
    city = City(60, 200)
    tourmanager.addCity(city)
    city2 = City(180, 200)
    tourmanager.addCity(city2)
    city3 = City(80, 180)
    tourmanager.addCity(city3)
    city4 = City(140, 180)
    tourmanager.addCity(city4)
    city5 = City(20, 160)
    tourmanager.addCity(city5)
    city6 = City(100, 160)
    tourmanager.addCity(city6)
    city7 = City(200, 160)
    tourmanager.addCity(city7)
    city8 = City(140, 140)
    tourmanager.addCity(city8)
    city9 = City(40, 120)
    tourmanager.addCity(city9)
    city10 = City(100, 120)
    tourmanager.addCity(city10)
    city11 = City(180, 100)
    tourmanager.addCity(city11)
    city12 = City(60, 80)
    tourmanager.addCity(city12)
    city13 = City(120, 80)
    tourmanager.addCity(city13)
```

```
city14 = City(180, 60)
tourmanager.addCity(city14)
city15 = City(20, 40)
tourmanager.addCity(city15)
city16 = City(100, 40)
tourmanager.addCity(city16)
city17 = City(200, 40)
tourmanager.addCity(city17)
city18 = City(20, 20)
tourmanager.addCity(city18)
city19 = City(60, 20)
tourmanager.addCity(city19)
city20 = City(160, 20)
tourmanager.addCity(city20)
# Initialize population
pop = Population(tourmanager, 50, True);
print ("Initial distance: " + str(pop.getFittest().getDistance()))
# Evolve population for 50 generations
ga = GA(tourmanager)
pop = ga.evolvePopulation(pop)
for i in range(0, 100):
  pop = ga.evolvePopulation(pop)
# Print final results
print ("Finished")
print ("Final distance: " + str(pop.getFittest().getDistance()))
print ("Solution:")
print (pop.getFittest())
```

Output



Task # 3

Eight queens using genetic Algorithm

Code

```
import math
import random
import sys

START_SIZE = 75 # Population size at start.
MAX_EPOCHS = 1000 # Arbitrary number of test cycles.
MATING_PROBABILITY = 0.7 # Probability of two chromosomes mating. Range: 0.0 < MATING_PROBABILITY < 1.0
MUTATION_RATE = 0.001 # Mutation Rate. Range: 0.0 < MUTATION_RATE < 1.0
MIN_SELECT = 10 # Minimum parents allowed for selection.
MAX_SELECT = 50 # Maximum parents allowed for selection. Range: MIN_SELECT < MAX_SELECT < START_SIZE
OFFSPRING_PER_GENERATION = 20 # New offspring created per generation. Range: 0 < OFFSPRING_PER_GENERATION < MAX_SELECT.
MINIMUM_SHUFFLES = 8 # For randomizing starting chromosomes
MAXIMUM_SHUFFLES = 20
PBC_MAX = 4 # Maximum Position-Based Crossover points. Range: 0 < PBC_MAX < 8 (> 8 isn't good).
MAX_LENGTH = 8 # chess board width.
class Chromosome:
    def __init__(self, maxLength):
        self.mMaxLength = maxLength
        self.mFitness = 0.0
        self.mSelected = False
        self.mSelectionProbability = 0.0
        self.mConflicts = 0

        self.mData = [0] * maxLength
        for i in range(self.mMaxLength):
            self.mData[i] = i
        return
    def compute_conflicts(self):
        x = 0
        y = 0
        tempx = 0
        tempy = 0
        board = []
        conflicts = 0
        dx = [-1, 1, -1, 1]
        dy = [-1, 1, 1, -1]
        done = False
        for i in range(self.mMaxLength):
            board.append([""] * self.mMaxLength)
            board[i][self.mData[i]] = "Q"
```

```
    # Walk through each of the Queens and compute the number of conflicts.
    for i in range(self.mMaxLength):
        x = i
        y = self.mData[i]

        # Check diagonals.
        for j in range(4):
            tempx = x
            tempy = y
            done = False
            while not done:
                tempx += dx[j]
                tempy += dy[j]
                if (tempx < 0 or tempx >= self.mMaxLength) or (tempy < 0 or tempy >= self.mMaxLength):
                    done = True
                else:
                    if board[tempx][tempy] == "Q":
                        conflicts += 1

    self.mConflicts = conflicts
    return

def get_conflicts(self):
    return self.mConflicts

def set_selection_probability(self, probability):
    self.mSelectionProbability = probability
    return

def get_selection_probability(self):
    return self.mSelectionProbability

def set_selected(self, isSelected):
    self.mSelected = isSelected
    return

def get_selected(self):
    return self.mSelected

def set_fitness(self, score):
    self.mFitness = score
    return

def get_fitness(self):
    return self.mFitness

def set_data(self, index, value):
    self.mData[index] = value
    return
```

```
    def get_data(self, index):
        return self.mData[index]

class NQueen1:
    def __init__(self, startSize, maxEpochs, matingProb, mutationRate, minSelect, maxSelect, generation, minShuffles,
maxShuffles, pbcMax, maxLength):
        self.mStartSize = startSize
        self.mEpochs = maxEpochs
        self.mMatingProbability = matingProb
        self.mMutationRate = mutationRate
        self.mMinSelect = minSelect
        self.mMaxSelect = maxSelect
        self.mOffspringPerGeneration = generation
        self.mMinimumShuffles = minShuffles
        self.mMaximumShuffles = maxShuffles
        self.mPBCMax = pbcMax
        self.mMaxLength = maxLength
        self.epoch = 0
        self.childCount = 0
        self.nextMutation = 0 # For scheduling mutations.
        self.mutations = 0
        self.population = []
        return
    def get_exclusive_random_integer(self, high, numberA):
        done = False
        numberB = 0
        while not done:
            numberB = random.randrange(0, high)
            if numberB != numberA:
                done = True
        return numberB
    def get_exclusive_random_integer_by_array(self, low, high, arrayA):
        done = False
        getRand = 0
        if high != low:
            while not done:
                done = True
                getRand = random.randrange(low, high)
                for i in range(len(arrayA)):
                    if getRand == arrayA[i]:
                        done = False
        else:
            getRand = high
        return getRand
    def math_round(self, inValue):
        outValue = 0
        if math.modf(inValue)[0] >= 0.5:
            outValue = math.ceil(inValue)
        else:
            outValue = math.floor(inValue)
        return outValue
```

```
def get_maximum(self):
    # Returns an array index.
    popSize = 0;
    thisChromo = Chromosome(self.mMaxLength)
    thatChromo = Chromosome(self.mMaxLength)
    maximum = 0
    foundNewMaximum = False
    done = False
    while not done:
        foundNewMaximum = False
        popSize = len(self.population)
        for i in range(popSize):
            if i != maximum:
                thisChromo = self.population[i]
                thatChromo = self.population[maximum]
                # The maximum has to be in relation to the Target.
                if math.fabs(thisChromo.get_conflicts() > thatChromo.get_conflicts()):
                    maximum = i
                    foundNewMaximum = True

        if foundNewMaximum == False:
            done = True

    return maximum

def get_minimum(self):
    # Returns an array index.
    popSize = 0;
    thisChromo = Chromosome(self.mMaxLength)
    thatChromo = Chromosome(self.mMaxLength)
    minimum = 0
    foundNewMinimum = False
    done = False

    while not done:
        foundNewMinimum = False
        popSize = len(self.population)
        for i in range(popSize):
            if i != minimum:
                thisChromo = self.population[i]
                thatChromo = self.population[minimum]
                # The minimum has to be in relation to the Target.
                if math.fabs(thisChromo.get_conflicts() < thatChromo.get_conflicts()):
                    minimum = i
                    foundNewMinimum = True

        if foundNewMinimum == False:
            done = True

    return minimum
```

```
def exchange_mutation(self, index, exchanges):
    i =0
    tempData = 0
    thisChromo = Chromosome(self.mMaxLength)
    gene1 = 0
    gene2 = 0
    done = False

    thisChromo = self.population[index]

    while not done:
        gene1 = random.randrange(0, self.mMaxLength)
        gene2 = self.get_exclusive_random_integer(self.mMaxLength, gene1)

        # Exchange the chosen genes.
        tempData = thisChromo.get_data(gene1)
        thisChromo.set_data(gene1, thisChromo.get_data(gene2))
        thisChromo.set_data(gene2, tempData)

        if i == exchanges:
            done = True

        i += 1

    self.mutations += 1
    return

def initialize_chromosomes(self):
    for i in range(self.mStartSize):
        newChromo = Chromosome(self.mMaxLength)
        self.population.append(newChromo)
        chromoIndex = len(self.population) - 1
        # Randomly choose the number of shuffles to perform.
        shuffles = random.randrange(self.mMinimumShuffles, self.mMaximumShuffles)
        self.exchange_mutation(chromoIndex, shuffles)
        newChromo = self.population[chromoIndex]
        newChromo.compute_conflicts()
    return
def get_fitness(self):
    # Lowest errors = 100%, Highest errors = 0%
    popSize = len(self.population)
    thisChromo = Chromosome(self.mMaxLength)
    bestScore = 0
    worstScore = 0
    # The worst score would be the one with the highest energy, best would be lowest.
    thisChromo = self.population[self.get_maximum()]
    worstScore = thisChromo.get_conflicts()
    # Convert to a weighted percentage.
    thisChromo = self.population[self.get_minimum()]
    bestScore = worstScore - thisChromo.get_conflicts()
    for i in range(popSize):
```

```
            thisChromo = self.population[i]
            thisChromo.set_fitness((worstScore - thisChromo.get_conflicts()) * 100.0 / bestScore)
        return
    def roulette_selection(self):
        j = 0
        popSize = 0
        genTotal = 0.0
        selTotal = 0.0
        rouletteSpin = 0.0
        thisChromo = Chromosome(self.mMaxLength)
        thatChromo = Chromosome(self.mMaxLength)
        done = False

        popSize = len(self.population)
        for i in range(popSize):
            thisChromo = self.population[i]
            genTotal += thisChromo.get_fitness()
        genTotal *= 0.01
        for i in range(popSize):
            thisChromo = self.population[i]
            thisChromo.set_selection_probability(thisChromo.get_fitness() / genTotal)
        for i in range(self.mOffspringPerGeneration):
            rouletteSpin = random.randrange(0, 99)
            j = 0
            selTotal = 0
            done = False
            while not done:
                thisChromo = self.population[j]
                selTotal += thisChromo.get_selection_probability()
                if selTotal >= rouletteSpin:
                    if j == 0:
                        thatChromo = self.population[j]
                    elif j >= popSize - 1:
                        thatChromo = self.population[popSize - 1]
                    else:
                        thatChromo = self.population[j - 1]

                    thatChromo.set_selected(True)
                    done = True
                else:
                    j += 1
        return
    def choose_first_parent(self):
        parent = 0
        thisChromo = Chromosome(self.mMaxLength)
        done = False
        while not done:
            # Randomly choose an eligible parent.
            parent = random.randrange(0, len(self.population) - 1)
            thisChromo = self.population[parent]
            if thisChromo.get_selected() == True:
```

```
            done = True
        return parent
    def choose_second_parent(self, parentA):
        parentB = 0
        thisChromo = Chromosome(self.mMaxLength)
        done = False
        while not done:
            # Randomly choose an eligible parent.
            parentB = random.randrange(0, len(self.population) - 1)
            if parentB != parentA:
                thisChromo = self.population[parentB]
                if thisChromo.get_selected() == True:
                    done = True
        return parentB
    def partially_mapped_crossover(self, chromA, chromB, child1, child2):
        thisChromo = Chromosome(self.mMaxLength)
        thisChromo = self.population[chromA]
        thatChromo = Chromosome(self.mMaxLength)
        thatChromo = self.population[chromB]
        newChromo1 = Chromosome(self.mMaxLength)
        newChromo1 = self.population[child1]
        newChromo2 = Chromosome(self.mMaxLength)
        newChromo2 = self.population[child2]
        crossPoint1 = random.randrange(0, self.mMaxLength)
        crossPoint2 = self.get_exclusive_random_integer(self.mMaxLength, crossPoint1)
        if crossPoint2 < crossPoint1:
            j = crossPoint1
            crossPoint1 = crossPoint2
            crossPoint2 = j
        # Copy Parent genes to offspring.
        for i in range(self.mMaxLength):
            newChromo1.set_data(i, thisChromo.get_data(i))
            newChromo2.set_data(i, thatChromo.get_data(i))
        for i in range(crossPoint1, crossPoint2 + 1):
            # // Get the two items to swap.
            item1 = thisChromo.get_data(i)
            item2 = thatChromo.get_data(i)
            pos1 = 0
            pos2 = 0
            # Get the items' positions in the offspring.
            for j in range(self.mMaxLength):
                if newChromo1.get_data(j) == item1:
                    pos1 = j
                elif newChromo1.get_data(j) == item2:
                    pos2 = j
            # Swap them.
            if item1 != item2:
                newChromo1.set_data(pos1, item2)
                newChromo1.set_data(pos2, item1)
            # Get the items'  positions in the offspring.
            for j in range(self.mMaxLength):
```

```
            if newChromo2.get_data(j) == item2:
                pos1 = j
            elif newChromo2.get_data(j) == item1:
                pos2 = j
        # Swap them.
        if item1 != item2:
            newChromo2.set_data(pos1, item1)
            newChromo2.set_data(pos2, item2)
    return
def position_based_crossover(self, chromA, chromB, child1, child2):
    k = 0
    numPoints = 0
    tempArray1 = [0] * self.mMaxLength
    tempArray2 = [0] * self.mMaxLength
    matchFound = False
    thisChromo = Chromosome(self.mMaxLength)
    thisChromo = self.population[chromA]
    thatChromo = Chromosome(self.mMaxLength)
    thatChromo = self.population[chromB]
    newChromo1 = Chromosome(self.mMaxLength)
    newChromo1 = self.population[child1]
    newChromo2 = Chromosome(self.mMaxLength)
    newChromo2 = self.population[child2]

    # Choose and sort the crosspoints.
    numPoints = random.randrange(0, self.mPBCMax) # if PBC_MAX is set any higher than 6 or 8.
    crossPoints = [0] * numPoints
    for i in range(numPoints):
        crossPoints[i] = self.get_exclusive_random_integer_by_array(0, self.mMaxLength - 1, crossPoints)
    # Get non-chosens from parent 2
    k = 0
    for i in range(self.mMaxLength):
        matchFound = False
        for j in range(numPoints):
            if thatChromo.get_data(i) == thisChromo.get_data(crossPoints[j]):
                matchFound = True
        if matchFound == False:
            tempArray1[k] = thatChromo.get_data(i)
            k += 1
    # Insert chosens into child 1.
    for i in range(numPoints):
        newChromo1.set_data(crossPoints[i], thisChromo.get_data(crossPoints[i]))
    # Fill in non-chosens to child 1.
    k = 0
    for i in range(self.mMaxLength):
        matchFound = False
        for j in range(numPoints):
            if i == crossPoints[j]:
                matchFound = True
        if matchFound == False:
            newChromo1.set_data(i, tempArray1[k])
```

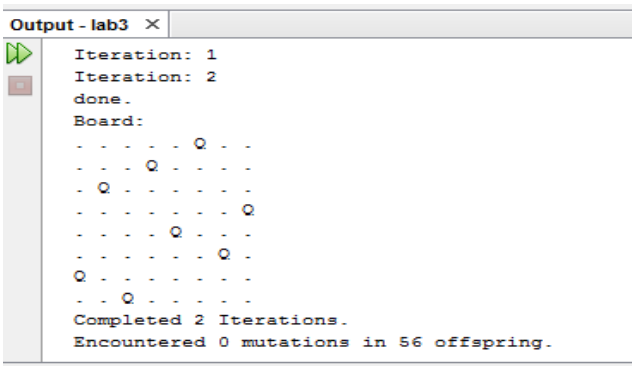
```
            k += 1
    # Get non-chosens from parent 1
    k = 0
    for i in range(self.mMaxLength):
        matchFound = False
        for j in range(numPoints):
            if thisChromo.get_data(i) == thatChromo.get_data(crossPoints[j]):
                matchFound = True

        if matchFound == False:
            tempArray2[k] = thisChromo.get_data(i)
            k += 1
    # Insert chosens into child 2.
    for i in range(numPoints):
        newChromo2.set_data(crossPoints[i], thatChromo.get_data(crossPoints[i]))
    # Fill in non-chosens to child 2.
    k = 0
    for i in range(self.mMaxLength):
        matchFound = False
        for j in range(numPoints):
            if i == crossPoints[j]:
                matchFound = True
        if matchFound == False:
            newChromo2.set_data(i, tempArray2[k])
            k += 1

    return
def displacement_mutation(self, index):
    j = 0
    point1 = 0
    length = 0
    point2 = 0
    tempArray1 = [0] * self.mMaxLength
    tempArray2 = [0] * self.mMaxLength
    thisChromo = Chromosome(self.mMaxLength)
    thisChromo = self.population[index]

    # Randomly choose a section to be displaced.
    point1 = random.randrange(0, self.mMaxLength)
    #sys.stdout.write(str(point1))
    #sys.stdout.write(str(self.mMaxLength))
    # Generate re-insertion point.
    candidate = self.mMaxLength - (point1 + 2)
    if candidate <= 0:
        candidate = 1
    point2 = self.get_exclusive_random_integer(candidate, point1)
    j = 0
    for i in range(self.mMaxLength): # Get non-chosen
        if i < point1 or i > point1 + length:
            tempArray1[j] = thisChromo.get_data(i)
            j += 1
```

```
    j = 0
    for i in range(point1, point1 + length + 1): # Get chosen
        tempArray2[j] = thisChromo.get_data(i)
        j += 1
    j = 0
    for i in range(point2, point2 + length + 1): # Place chosen
        thisChromo.set_data(i, tempArray2[j])
        j += 1
    j = 0
    for i in range(i, self.mMaxLength): # Place non-chosen
        if i < point2 or i > point2 + length:
            thisChromo.set_data(i, tempArray1[j])
            j += 1
    self.mutations += 1
    return
def do_mating(self):
    getRand = 0
    parentA = 0
    parentB = 0
    newChildIndex1 = 0
    newChildIndex2 = 0
    newChromo1 = Chromosome(self.mMaxLength)
    newChromo2 = Chromosome(self.mMaxLength)
    for i in range(self.mOffspringPerGeneration):
        parentA = self.choose_first_parent()
        # Test probability of mating.
        getRand = random.randrange(0, 100)

        if getRand <= self.mMatingProbability * 100:
            parentB = self.choose_second_parent(parentA)
            newChromo1 = Chromosome(self.mMaxLength)
            newChromo2 = Chromosome(self.mMaxLength)
            self.population.append(newChromo1)
            newIndex1 = len(self.population) - 1
            self.population.append(newChromo2)
            newIndex2 = len(self.population) - 1
            self.partially_mapped_crossover(parentA, parentB, newIndex1, newIndex2)
            # self.position_based_crossover(parentA, parentB, newIndex1, newIndex2)
            if self.childCount - 1 == self.nextMutation:
                self.exchange_mutation(newIndex1, 1)
                # self.displacement_mutation(newIndex1)
            elif self.childCount == self.nextMutation:
                self.exchange_mutation(newIndex2, 1)
                # self.displacement_mutation(newIndex2)
            newChromo1 = self.population[newIndex1]
            newChromo1.compute_conflicts()
            newChromo2 = self.population[newIndex2]
            newChromo2.compute_conflicts()
            self.childCount += 2
            # Schedule next mutation.
            if math.fmod(self.childCount, self.math_round(1.0 / self.mMutationRate)) == 0:
```

```
            self.nextMutation = self.childCount + random.randrange(0, self.math_round(1.0 / self.mMutationRate))
        return
    def prep_next_epoch(self):
        popSize = 0;
        thisChromo = Chromosome(self.mMaxLength)
        # Reset flags for selected individuals.
        popSize = len(self.population)
        for i in range(popSize):
            thisChromo = self.population[i]
            thisChromo.set_selected(False)
        return
    def print_best_solution(self, bestSolution = Chromosome(MAX_LENGTH)):
        board = []
        for i in range(self.mMaxLength):
            board.append([""] * self.mMaxLength)
            board[i][bestSolution.get_data(i)] = "Q"

        # Display the board.
        sys.stdout.write("Board:\n")
        for j in range(self.mMaxLength):
            for i in range(self.mMaxLength):
                if board[i][j] == "Q":
                    sys.stdout.write("Q ")
                else:
                    sys.stdout.write(". ")

            sys.stdout.write("\n")
        return
    def genetic_algorithm(self):
        popSize = 0
        thisChromo = Chromosome(self.mMaxLength)
        done = False
        self.mutations = 0
        self.nextMutation = random.randrange(0, self.math_round(1.0 / self.mMutationRate))
        while not done:
            popSize = len(self.population)
            for i in range(popSize):
                thisChromo = self.population[i]
                if thisChromo.get_conflicts() == 0 or self.epoch == self.mEpochs:
                    done = True
            self.get_fitness()
            self.roulette_selection()
            self.do_mating()
            self.prep_next_epoch()
            self.epoch += 1
            # This is here simply to show the runtime status.
            sys.stdout.write("Epoch: " + str(self.epoch) + "\n")
        sys.stdout.write("done.\n")
        if self.epoch != self.mEpochs:
            popSize = len(self.population)
            for i in range(popSize):
```

```
            thisChromo = self.population[i]
            if thisChromo.get_conflicts() == 0:
                self.print_best_solution(thisChromo)
        sys.stdout.write("Completed " + str(self.epoch) + " epochs.\n")
        sys.stdout.write("Encountered " + str(self.mutations) + " mutations in " + str(self.childCount) + " offspring.\n")
        return
if __name__ == '__main__':
    nq1 = NQueen1(START_SIZE, MAX_EPOCHS, MATING_PROBABILITY, MUTATION_RATE, MIN_SELECT,
MAX_SELECT, OFFSPRING_PER_GENERATION, MINIMUM_SHUFFLES, MAXIMUM_SHUFFLES, PBC_MAX,
MAX_LENGTH)
    nq1.initialize_chromosomes()
    nq1.genetic_algorithm()
```

Output



CSL 411 –Artificial Intelligence Lab

FALL 2017

Lab # 08

Decision Tree and Linear Regression

COURSE INSTRUCTOR: TARIQ SIDDIQUE

LAB ENGINEER: TARWAN KUMAR

DEPARTMENT OF COMPUTER SCIENCE

BAHRIA UNIVERSITY, KARACHI CAMPUS

Task # 1

Write a program in Python to implement the ID3 decision tree algorithm. You should read in a tab delimited dataset, and output to the screen your decision tree and the training set accuracy in some readable format.

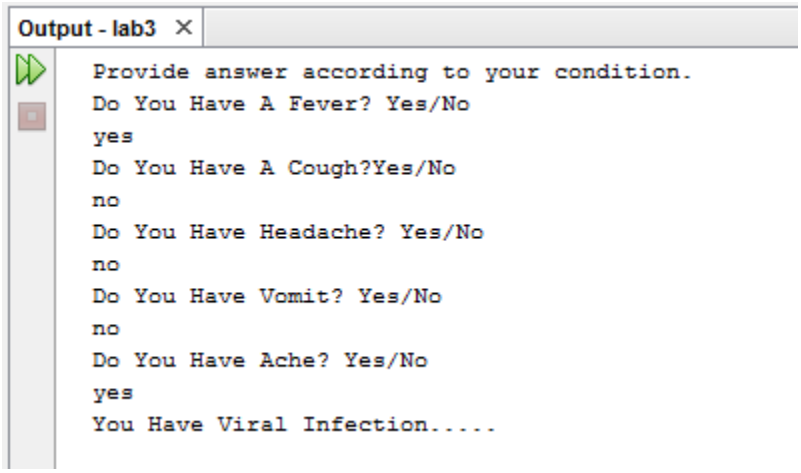
Code

```
if __name__ == "__main__":
    print("Provide answer according to your condition.");
    condition = input("Do You Have A Fever? Yes/No\n")
    condition.lower()
    if condition == "yes" :
        condition = input("Do You Have A Cough?Yes/No\n")
        condition.lower()
        if condition == "yes" :
            condition = input("Do You Have Sortness Of Breath? Yes/No\n")
            conditiom.lower()
            if condition == "yes" :
                print("You Have Pneumonia.....")
            else:
                condition = input("Do You Have Headache? Yes/No\n")
                condition.lower()
                if condition == "yes" :
                    print("You Have Viral.....")
                else:
                    print("Consult A Doctor.....")
        else:
            condition = input("Do You Have Headache? Yes/No\n")
            condition.lower()
            if condition == "yes" :
```

```
    condition = input("Do You Have Pain? Yes/No\n")
    condition.lower()
    if condition == "yes" :
        print("You Have Meningitis.....")
    else:
        condition = input("Do You Have Vomit? Yes/No\n")
        condition.lower()
        if condition == "yes" :
            print("You Have Digestive Tract.....")
        else:
            print("Consult A Doctor.....")
else:
    condition = input("Do You Have Vomit? Yes/No\n")
    condition.lower()
    if condition == "yes" :
        print("Consult A Doctor.....")
    else:
        condition = input("Do You Have Ache? Yes/No\n")
        condition.lower()
        if condition == "yes" :
            print("You Have Viral Infection.....")
        else:
            condition = input("Do You Have Sore Throat? Yes/No\n")
            condition.lower()
            if condition == "yes" :
                print("You Have Throat Infection.....")
            else:
                condition = input("Do You Have Back Pain? Yes/No\n")
                condition.lower()
```

```
            if condition == "yes" :
                print("You Have Kidney Infection.....")
            else:
                condition = input("Do You Have Pain Urinate? Yes/No\n")
                condition.lower()
                if condition == "yes" :
                    print("You Have Urinary Tract Infection.....")
                else:
                    condition = input("Do You Have Exposed to Sun? Yes/No\n")
                    condition.lower()
                    if condition == "yes" :
                        print("You Have Sun Stroke.....")
else:
    print("You Are completely Okay!")
```

Output



Task # 2

Write a program in Python to implement the linear regression algorithm. You should read in a tab delimited dataset, and output to the screen your final linear regression expression.

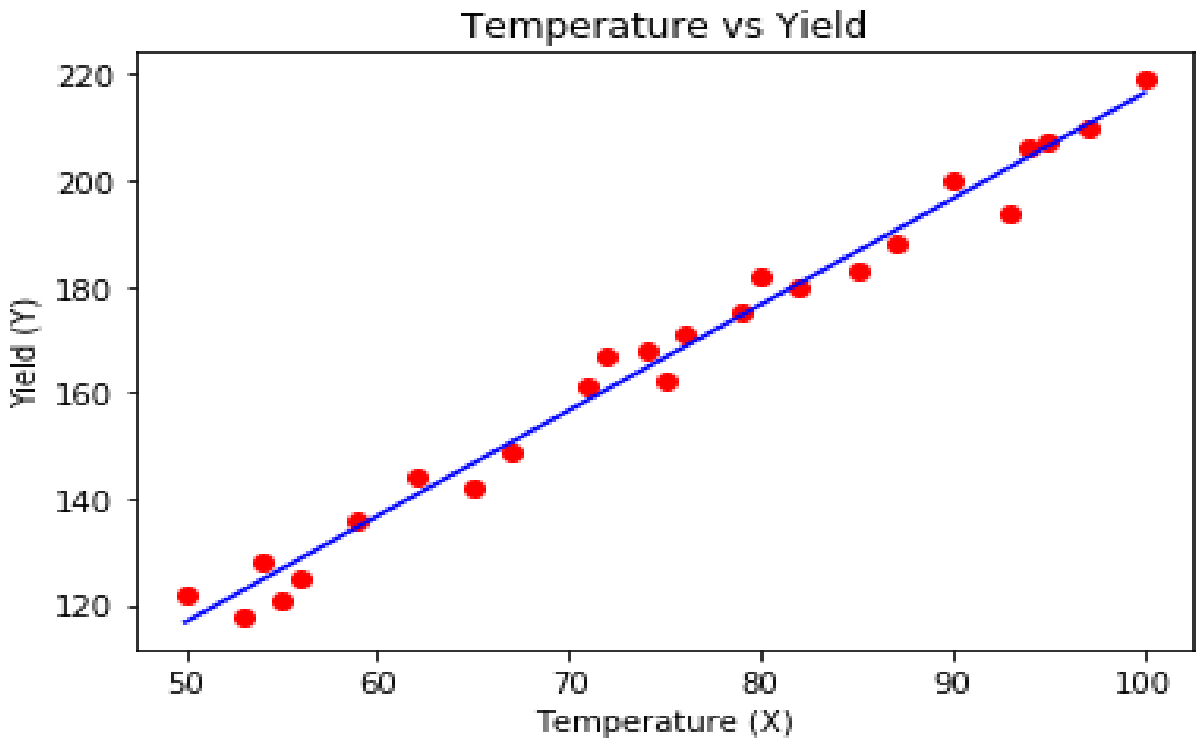
Code

```
# Required Packages
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, linear_model


# Function to get data
def get_data(file_name):
	data = pd.read_csv(file_name)
	temp_x_parameter = []
	Yield_y_parameter = []
	for x1,y1 in zip(data['Temperature'],data['Yield']):
		temp_x_parameter.append([float(x1)])
		Yield_y_parameter.append(float(y1))
	return temp_x_parameter,Yield_y_parameter


# Function to know which Tv show will have more viewers
def more_viewers(x1,y1):
	regr1 = linear_model.LinearRegression()
	regr1.fit(x1, y1)
	predicted_value1 = regr1.predict(32)
	print(predicted_value1)
```

```
plt.scatter(x1,y1,color='red')
plt.plot(x1,regr1.predict(x1), color='blue')
plt.title('Salary vs Experience (Traning Set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

```
x1,y1 = get_data('data.csv')
#print x1,y1,x2,y2
more_viewers(x1,y1)
```

Output



CSL 411 –Artificial Intelligence Lab

FALL 2017

Lab # 09

Expert Systems

COURSE INSTRUCTOR: TARIQ SIDDIQUE

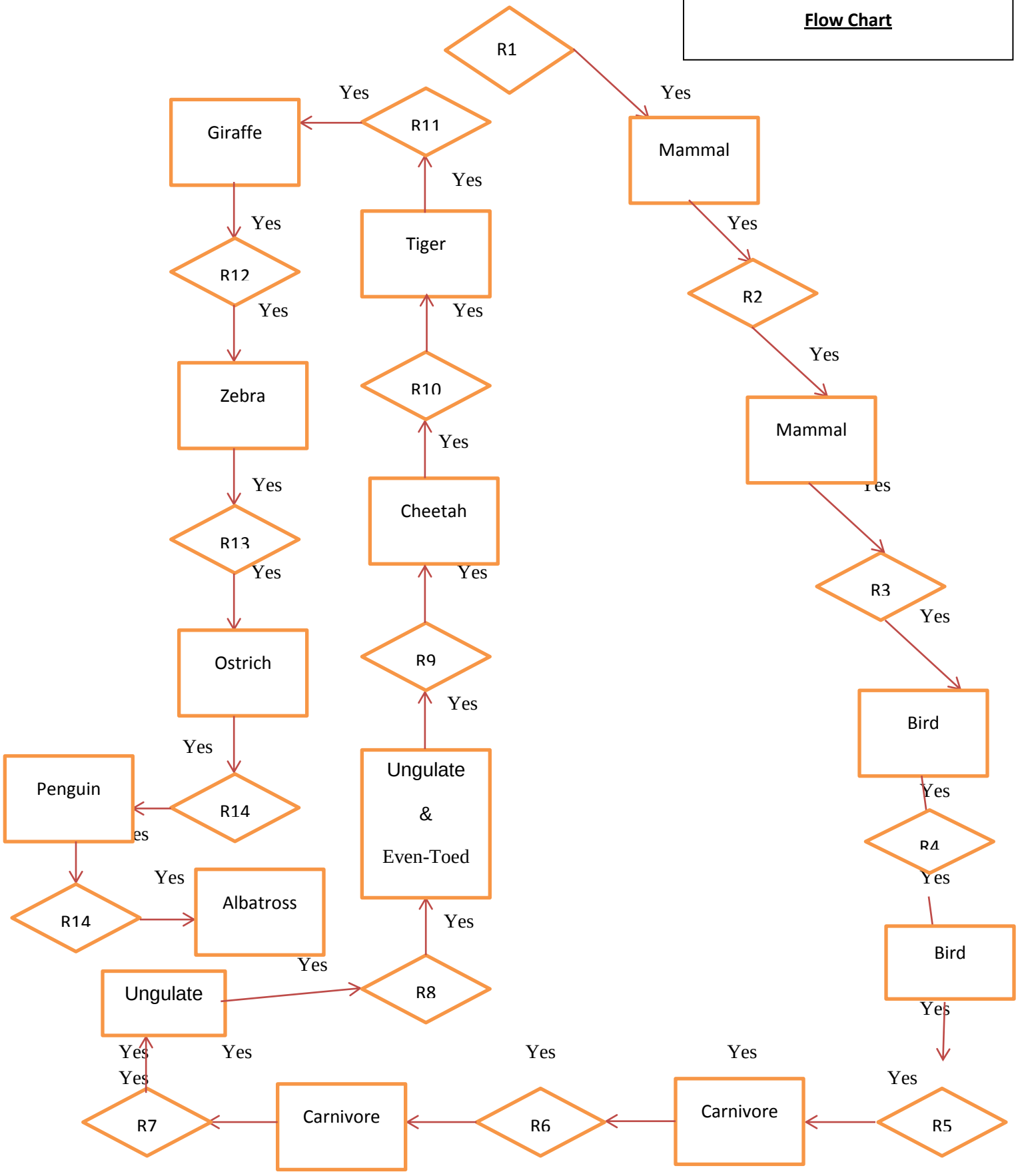
LAB ENGINEER: TARWAN KUMAR

DEPARTMENT OF COMPUTER SCIENCE

BAHRIA UNIVERSITY, KARACHI CAMPUS

Task # 1

Flow Chart



Task # 1

Code

```
if __name__ == "__main__":
    print("Provide answer according to your condition.");
    condition = input("The Animal Has Hair? Yes/No\n")
    condition.lower()
    if condition == "Yes" :
        print("It is a Mammal (R1)");
    else:
        print("It is Not a Mammal");
    condition = input("The Animal Gives Milk? Yes/No\n")
    condition.lower()
    if condition == "Yes":
        print("It is a Mammal (R2)");
    else:
        print("It is Not a Mammal");
    condition = input("The Animal has feathers? Yes/No\n")
    condition.lower()
    if condition == "Yes":
        print("It is a Bird (R3)");
    else:
        print("It is Not a Bird");
    condition = input("The Animal Flies? Yes/No\n") and input("The Animal Lays Eggs? Yes/No\n")
    condition.lower()
    if condition == "Yes":
        print("It is a Bird (R4)");
    else:
```

```
    print("It is Not a Bird");


condition = input("The Animal is a Mammal? Yes/No\n") and input("The Animal Eats Meat? Yes/No\n")

condition.lower()

if condition == "Yes":

    print("It is a Carnivore (R5)");

else:

    print("It is Not a Carnivore");

condition = input("The Animal is a Mammal? Yes/No\n") and input("The Animal Has Pointed Teeth? Yes/No\n") and input("The Animal Has Claws? Yes/No\n") and input("The Animal's Eyes Point Forward? Yes/No\n")

condition.lower()

if condition == "Yes":

    print("It is a Carnivore (R6)");

else:

    print("It is Not a Carnivore");

condition = input("The Animal is a Mammal? Yes/No\n") and input("The Animal Has Hooves? Yes/No\n")

condition.lower()

if condition == "Yes":

    print("It is an Ungulate (R7)");

else:

    print("It is Not an Ungulate");

condition = input("The Animal is a Mammal? Yes/No\n") and input("The Animal Chews Cud? Yes/No\n")

condition.lower()

if condition == "Yes":

    print("It is an Ungulate And It is Even-Toed (R8)");

else:

    print("It is Not an Ungulate And It is Not Even-Toed");

condition = input("The Animal is a Carnivore? Yes/No\n") and input("The Animal Has a Tawny Colour? Yes/No\n") and input("The Animal Has Dark Spots? Yes/No\n")

condition.lower()
```

```
    if condition == "Yes":
        print("It is a Cheetah (R9)");
    else:
        print("It is Not a Cheetah");
    condition = input("The Animal is a Carnivore? Yes/No\n") and input("The Animal Has a Tawny Colour? Yes/No\n") and input("The Animal Has Black Stripes? Yes/No\n")
    condition.lower()
    if condition == "Yes":
        print("It is a Tiger (R10)");
    else:
        print("It is Not A Tiger");
    condition = input("The Animal is an Ungulate? Yes/No\n") and input("The Animal Has Long Legs? Yes/No\n") and input("The Animal Has a Long Neck? Yes/No\n") and input("The Animal Has Dark Spot? Yes/No\n")
    condition.lower()
    if condition == "Yes":
        print("It is a Giraffe (R11)");
    else:
        print("It is Not a Giraffe");
    condition = input("The Animal is an Ungulate? Yes/No\n") and input("The Animal Has a White colour? Yes/No\n") and input("The Animal Has Black Stripes? Yes/No\n")
    condition.lower()
    if condition == "Yes":
        print("It is a Zebra (R12)");
    else:
        print("It is Not a Zebra");
    condition = input("The Animal is a Bird? Yes/No\n") and input("The Animal Does Not Fly? Yes/No\n") and input("The Animal Has Long Legs? Yes/No\n") and input("The Animal Has a Long Neck? Yes/No\n") and input("The Animal is Black And White? Yes/No\n")
    condition.lower()
    if condition == "Yes":
        print("It is an Ostrich (R13)");
```

```
    else:
        print("It is Not an Ostrich");
    condition = input("The Animal is a Bird? Yes/No\n") and input("The Animal Does Not Fly? Yes/No\n") and input("The Animal Swims? Yes/No\n") and input("The Animal is a Black And White? Yes/No\n")
    condition.lower()
    if condition == "Yes":
        print("It is a Penguin (R14)");
    else:
        print("It is Not a Penguin");
    condition = input("The Animal is a Bird? Yes/No\n") and input("The Animal is a Good Flier? Yes/No\n")
    condition.lower()
    if condition == "Yes":
        print("It is an Albatross (R15)");
    else:
        print("It is Not an Albatross");


else:
    print("!!!!");
```

Output

```
Provide answer according to your condition.
The Animal Has Hair? Yes/No
Yes
It is a Mammal (R1)
The Animal Gives Milk? Yes/No
Yes
It is a Mammal (R2)
The Animal has feathers? Yes/No
Yes
It is a Bird (R3)
The Animal Flies? Yes/No
Yes
The Animal Lays Eggs? Yes/No
Yes
It is a Bird (R4)
The Animal is a Mammal? Yes/No
Yes
The Animal Eats Meat? Yes/No
Yes
It is a Carnivore (R5)
The Animal is a Mammal? Yes/No
Yes
The Animal Has Pointed Teeth? Yes/No
Yes
The Animal Has Claws? Yes/No
No
The Animal's Eyes Point Forward? Yes/No
No
It is Not a Carnivore
The Animal is a Mammal? Yes/No
Yes
```

```
The Animal is a Mammal? Yes/No
Yes
The Animal Has Hooves? Yes/No
Yes
It is an Ungulate (R7)
The Animal is a Mammal? Yes/No
Yes
The Animal Chews Cud? Yes/No
No
It is Not an Ungulate And It is Not Even-Toed
The Animal is a Carnivore? Yes/No
Yes
The Animal Has a Tawny Colour? Yes/No
Yes
The Animal Has Dark Spots? Yes/No
Yes
It is a Cheetah (R9)
The Animal is a Carnivore? Yes/No
Yes
The Animal Has a Tawny Colour? Yes/No
Yes
The Animal Has Black Stripes? Yes/No
Yes
It is a Tiger (R10)
The Animal is an Ungulate? Yes/No
No
The Animal Has Long Legs? Yes/No
No
The Animal Has a Long Neck? Yes/No
No
The Animal Has Dark Spot? Yes/No
No
```

```
The Animal Has Dark Spot? Yes/No
No
It is Not a Giraffe
The Animal is an Ungulate? Yes/No
No
The Animal Has a White colour? Yes/No
No
The Animal Has Black Stripes? Yes/No
No
It is Not a Zebra
The Animal is a Bird? Yes/No
Yes
The Animal Does Not Fly? Yes/No
Yes
The Animal Has Long Legs? Yes/No
Yes
The Animal Has a Long Neck? Yes/No
Yes
The Animal is Black And White? Yes/No
Yes
It is an Ostrich (R13)
The Animal is a Bird? Yes/No
No
The Animal Does Not Fly? Yes/No
No
The Animal Swims? Yes/No
No
The Animal is a Black And White? Yes/No
No
It is Not a Penguin
The Animal is a Bird? Yes/No
Yes
```

```
The Animal is a Bird? Yes/No
Yes
The Animal is a Good Flier? Yes/No
No
It is Not an Albatross
```

Task # 2

Code

```
if __name__ == "__main__":
    print("Provide answer according to your condition.");
    condition = input("Open Circuit or Run? Yes/No\n")
    condition.lower()
    if condition == "Yes":
        condition = input("Fusible link blown? Yes/No\n")
        condition.lower()
        if condition == "Yes":
            condition = input("Short after blown link? Yes/No\n")
            condition.lower()
            if condition == "Yes":
                print("Check for short between ground and run circuit through next connector");
                condition = input("Short circuit? Yes/No\n")
                condition.lower()
                if condition == "Yes":
                    print("Locate the short by visual inspection and replace wire, relay or device");
                else:
                    print("Check for short between ground and run circuit through next connector");
            else:
                print("Replace with new fusible link rated same or lower")
        else:
            condition = input("ignition switch open? Yes/No\n")
            condition.lower()
            if condition == "Yes":
                print("replace or bypass ignition switch");
```

```
        else:
            condition = input("battery connector goog? Yes/No\n")
            condition.lower()
            if condition == "Yes":
                print("check for open between positive and run circuit through next connector")
            else:
                print("laugh, clean or replace");
            condition = input("Open? Yes/No\n")
            condition.lower()
            if condition == "Yes":
                print("locate the open (break) by visual inspection and replace wire, relay or device");
            else:
                print("check for open between positive and run circuit through next connector");
else:
        condition = input("Starting problem? Yes/No\n")
        condition.lower()
        if condition == "Yes" :
            print("see flowchart for starting")
        else:
            condition = input("Accessory failure? Yes/No\n")
            condition.lower()
            if condition == "Yes":
                condition = input("All accessories fail? Yes/No\n")
                condition.lower()
                if condition == "Yes":
                    print("Accessory fuse first. check accessory circuit for open");
                else:
                    print("Check power at accessory terminals. check accessory ground mounting. remove and test");
            else:
```

```
condition = input("Battery running down? Yes/No\n")
condition.lower()
if condition == "Yes" :
    condition = input("Alternator tested OK? Yes/No\n")
    condition.lower()
    if condition == "Yes":
        condition = input("Battery test OK? Yes/No\n")
        condition.lower()
        if condition == "Yes":
            print("check hidden light drain, high computer draw, alarm system, etc..");
        else:
            print("test specific gravity, levels, if near lifespan, replace");
    else:
        condition = input("voltage regulator good? Yes/No\n")
        condition.lower()
        if condition == "Yes":
            print("check belt, grounds, all alternator connectors, voltages");
        else:
            print("check voltage regulator ground, replace, otherwise control");
else:
    condition = input("single lamp failure? Yes/No\n")
    condition.lower()
    if condition == "Yes" :
        print("replace lamp, check ground, wire direct");
    else:
        condition = input("blinker failure? Yes/No\n")
        condition.lower()
        if condition == "Yes":
            print("test with emergency flasher, swap blinker relay, bulb, swith failure");
```

```
        else:

            print("check fuse, failure in switch or writing for multiple lamp failure, i.e. headlights, emergency flashers, brake lights")
```

Output

```
Provide answer according to your condition.
Open Circuit or Run? Yes/No
Yes
Fusible link blown? Yes/No
Yes
Short after blown link? Yes/No
Yes
Check for short between ground and run circuit through next connector
Short circuit? Yes/No
Yes
Locate the short by visual inspection and replace wire, relay or device
```

CSL 411 –Artificial Intelligence Lab

FALL 2017

Lab # 10

Neural Networks

COURSE INSTRUCTOR: TARIQ SIDDIQUE

LAB ENGINEER: TARWAN KUMAR

DEPARTMENT OF COMPUTER SCIENCE

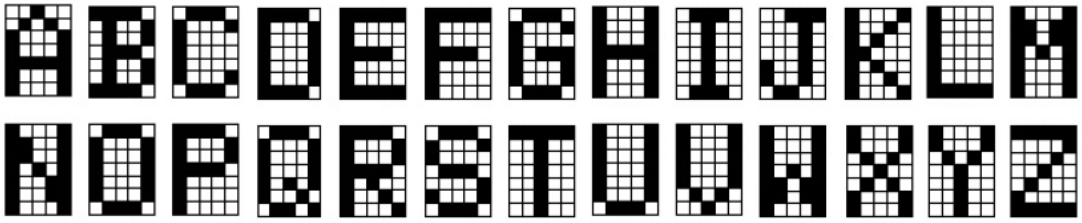
BAHRIA UNIVERSITY, KARACHI CAMPUS

Task # 1

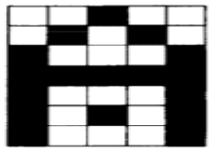
Exercise

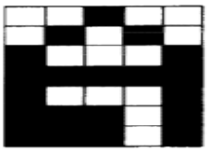
Exercise 1

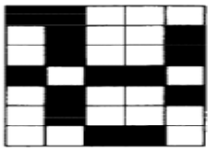
1. Add **AND and OR** logic gates using above artificial neural network example.
2. Modify the above digit recognition code to recognize the alphabets from A-Z (7×5 Grid).Test the program with different input values and explain the outputs. Store the patterns values into ***text file*** and load into training pattern and test pattern arrays.

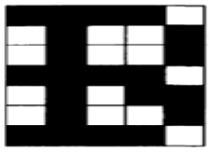


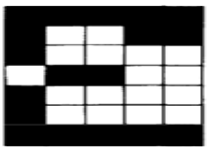
Use the test patterns that looks like the following 8 patterns:


pattern 1

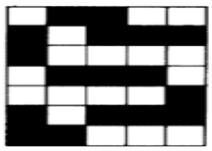

pattern 2

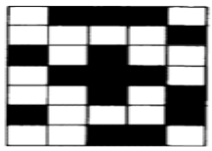

pattern 3


pattern 4


pattern 5


pattern 6


pattern 7


pattern 8

Code

```
import random
import math
 #import NeuralNetwork
class NeuralNetwork:
def __init__(self,nrInput,nrHidden,nrOutput,learnRate):
      self._learnRate = learnRate
self._input = [0 for iin range(nrInput)]
self._hidden = [0 for iin range(nrHidden)]
self._output = [0 for iin range(nrOutput)]
      self._expOutput = [0 for iin range(nrOutput)]
self._win = [[0 for iin range(nrHidden)]for j in range(nrInput)]
      self._wout = [[0 for iin range(nrOutput)]for j in range(nrHidden)]
      self._dwin = [[0 for iin range(nrHidden)]for j in range(nrInput)]
      self._dwout = [[0 for iin range(nrOutput)]for j in range(nrHidden)]
self.Init()

def Init(self):
for iin range(len(self._input)):
for j in range (len(self._hidden)):
self._win[i][j] = random.random() - 0.5

for iin range(len(self._hidden)):
for j in range(len(self._output)):
            self._wout[i][j] = random.random() - 0.5
self.ClearMatrixDelta();


def Run(self,input):
self.SetInput(input)
self.FeedForward()
return self.GetOutput()

def Train(self,input,expOutput):
self.ClearMatrixDelta()
for iin range(len(input)):
self.SetInput(input[i])
self.SetExpOutput(expOutput[i])
self.FeedForward()
self.BackPropagateError()
self.Learn()


def ClearMatrixDelta(self):
for iin range(len(self._input)):
for j in range(len(self._hidden)):
                self._dwin[i][j] = 0;
for iin range(len(self._hidden)):
for j in range(len(self._output)):
                self._dwout[i][j] = 0;


def SetInput(self,input):
for iin range(len(input)):
self._input[i] = input[i];

def FeedForward(self):
for iin range(len(self._hidden)):
```

```
sumh = 0.0
for j in range(len(self._input)):
sumh += self._input[j] * self._win[j][i];
self._hidden[i] = 1.0 / (1.0 + math.exp(-sumh))

for iin range(len(self._output)):
sumh = 0.0
for j in range(len(self._hidden)):
sumh += self._hidden[j] * self._wout[j][i]
self._output[i] = 1.0 / (1.0 + math.exp(-sumh))

def GetOutput(self):
return self._output

def SetExpOutput(self,expOutput):
for iin range(len(expOutput)):
            self._expOutput[i] = expOutput[i]

def BackPropagateError(self):

erro = [0 for iin range(len(self._output))]
errh = [0 for j in range(len(self._hidden))]
for iin range(len(self._output)):
erro[i] = self._output[i] * (1.0 - self._output[i]) * (self._expOutput[i] - self._output[i])

for iin range(len(self._hidden)):
sumerr = 0.0
for j in range(len(self._output)):
sumerr += self._wout[i][j] * erro[j]
errh[i] = self._hidden[i] * (1.0 - self._hidden[i]) * sumerr

for iin range(len(self._hidden)):
for j in range(len(self._output)):
              self._dwout[i][j] +=  erro[j] * self._hidden[i]
for iin range(len(self._input)):
for j in range(len(self._hidden)):
              self._dwin[i][j] += errh[j] * self._input[i]

def Learn(self):
for iin range(len(self._hidden)):
for j in range(len(self._output)):
            self._wout[i][j] += self._learnRate * self._dwout[i][j]
for iin range(len(self._input)):
for j in range(len(self._hidden)):
self._win[i][j] += self._learnRate * self._dwin[i][j];



def TestXOR():

input_patterns =[[ 0.0, 0.0 ],
                 [ 1.0, 0.0 ],
                 [ 0.0, 1.0 ],
                 [ 1.0, 1.0 ]]

ideal_output = [[0.0],
                [1.0],
                [1.0],
                [0.0]]
```

```
nnet = NeuralNetwork(2, 5, 1, 0.7)

   print("Training network. Please wait...")
for iin range(5000):
nnet.Train(input_patterns, ideal_output);


   print("Training finished.\n");
         #Console.ReadKey(true);

for iin range(len(input_patterns)):
outst = nnet.Run(input_patterns[i]);
        print("XOR({",input_patterns[i][0],"}, {",input_patterns[i][1],"}) = {", outst[0],"}")


   print("Similar data test:\n");
   a = [0.1,0.9]
outi = nnet.Run(a)
   print("XOR(",a[0],",",a[1],") = {",outi[0],"}")


def TestOR():

input_patterns =[[ 0.0, 0.0 ],
                 [ 1.0, 0.0 ],
                 [ 0.0, 1.0 ],
                 [ 1.0, 1.0 ]]

ideal_output = [[0.0],
                [1.0],
                [1.0],
                [1.0]]

nnet = NeuralNetwork(2, 5, 1, 0.7)

   print("Training network. Please wait...")
for iin range(5000):
nnet.Train(input_patterns, ideal_output);


   print("Training finished.\n");
      #Console.ReadKey(true);

for iin range(len(input_patterns)):
outst = nnet.Run(input_patterns[i]);
      print("OR({",input_patterns[i][0],"}, {",input_patterns[i][1],"}) = {", outst[0],"}")


   print("Similar data test:\n");
   a = [0.1,0.9]
outi = nnet.Run(a)
   print("OR(",a[0],",",a[1],") = {",outi[0],"}")


def TestAND():

input_patterns =[[ 0.0, 0.0 ],
                  [ 1.0, 0.0 ],
```

```
                 [ 0.0, 1.0 ],
                 [ 1.0, 1.0 ]]

ideal_output = [[0.0],
                [1.0],
                [1.0],
                [1.0]]

nnet = NeuralNetwork(2, 5, 1, 0.7)

   print("Training network. Please wait...")
for iin range(5000):
nnet.Train(input_patterns, ideal_output);


   print("Training finished.\n");
         #Console.ReadKey(true);

for iin range(len(input_patterns)):
outst = nnet.Run(input_patterns[i]);
        print("AND({",input_patterns[i][0],"}, {",input_patterns[i][1],"}) = {", outst[0],"}")


   print("Similar data test:\n");
   a = [0.1,0.9]
outi = nnet.Run(a)
   print("AND(",a[0],",",a[1],") = {",outi[0],"}")



def TestDigitPattern():
   pattern0 = [0, 1, 1, 1, 0,
               0, 1, 0, 1, 0,
               0, 1, 0, 1, 0,
               0, 1, 0, 1, 0,
               0, 1, 1, 1, 0]

   pattern1 = [0, 0, 1, 0, 0,
               0, 1, 1, 0, 0,
               0, 0, 1, 0, 0,
               0, 0, 1, 0, 0,
               0, 0, 1, 0, 0]

   pattern2 = [0, 1, 1, 1, 0,
               0, 0, 0, 1, 0,
               0, 1, 1, 1, 0,
               0, 1, 0, 0, 0,
               0, 1, 1, 1, 0]

   pattern3 = [0, 1, 1, 1, 0,
               0, 0, 0, 1, 0,
               0, 1, 1, 1, 0,
               0, 0, 0, 1, 0,
               0, 1, 1, 1, 0]

   pattern4 = [0, 1, 0, 1, 0,
               0, 1, 0, 1, 0,
               0, 1, 1, 1, 0,
```

```
            0, 0, 0, 1, 0,
            0, 0, 0, 1, 0]

    pattern5 = [0, 1, 1, 1, 0,
            0, 1, 0, 0, 0,
            0, 1, 1, 1, 0,
            0, 0, 0, 1, 0,
            0, 1, 1, 1, 0]

    pattern6 = [0, 1, 1, 1, 0,
            0, 1, 0, 0, 0,
            0, 1, 1, 1, 0,
            0, 1, 0, 1, 0,
            0, 1, 1, 1, 0]

    pattern7 = [0, 1, 1, 1, 0,
            0, 0, 0, 1, 0,
            0, 0, 0, 1, 0,
            0, 0, 1, 0, 0,
            0, 1, 0, 0, 0]

    pattern8 = [0, 1, 1, 1, 0,
            0, 1, 0, 1, 0,
            0, 1, 1, 1, 0,
            0, 1, 0, 1, 0,
            0, 1, 1, 1, 0]

    pattern9 = [0, 1, 1, 1, 0,
            0, 1, 0, 1, 0,
            0, 1, 1, 1, 0,
            0, 0, 0, 1, 0,
            0, 1, 1, 1, 0]

TrainPatterns = [pattern0, pattern1, pattern2, pattern3, pattern4,
                 pattern5, pattern6, pattern7, pattern8, pattern9]

TrainResults = [[0.0],
                [0.1],
                [0.2],
                [0.3],
                [0.4],
                [0.5],
                [0.6],
                [0.7],
                [0.8],
                [0.9]]

nnet = NeuralNetwork(25, 25, 1, 0.8)

    print("Training network. Please wait...\n")
for iin range(10000):
nnet.Train(TrainPatterns, TrainResults)

    print("Training finished.\n")

for iin range(len(TrainPatterns)):
        output = nnet.Run(TrainPatterns[i])
        print("{",i,"} pattern = {",output[0],"} which is digit {", int(output[0]*10),"}")
```

```
    print("Similar data test:\n")
    testPattern1 = [0, 0.0, 1.0, 0, 0,
                    0, 1.0, 0.9, 0, 0,
                    0, 0.0, 1.0, 0, 0,
                    0, 0.0, 0.9, 0, 0,
                    0, 0.0, 1.0, 0, 0]

    outv1 = nnet.Run(testPattern1)

    print("TP similar 1 = {", outv1[0],", Digit={", int(outv1[0] * 10),"}")

    testPattern5 = [0, 1.0, 0.9, 0.9, 0,
                    0, 1.0, 0.0, 0.0, 0,
                    0, 1.0, 0.9, 1.1, 0,
                    0, 0.0, 0.0, 1.0, 0,
                    0, 1.1, 0.9, 0.9, 0]

    outv5 = nnet.Run(testPattern5)
    print("TP similar 5 = {", outv5[0],"}, Digit={", int(outv5[0] * 10),"}")



if __name__ == "__main__":
TestXOR()
TestOR()
TestAND()
TestDigitPattern()
```

Output

```
Training network. Please wait...
Training finished.

XOR({ 0.0 }, { 0.0 }) = { 0.03395301475385688 }
XOR({ 1.0 }, { 0.0 }) = { 0.9650065038323471 }
XOR({ 0.0 }, { 1.0 }) = { 0.9702317178040295 }
XOR({ 1.0 }, { 1.0 }) = { 0.03508217282335742 }
Similar data test:

XOR( 0.1 , 0.9 ) = { 0.9370007755952223 }
Training network. Please wait...
Training finished.

OR({ 0.0 }, { 0.0 }) = { 0.02242394529720877 }
OR({ 1.0 }, { 0.0 }) = { 0.987034905278737 }
OR({ 0.0 }, { 1.0 }) = { 0.9871315531781601 }
OR({ 1.0 }, { 1.0 }) = { 0.9960841099415735 }
Similar data test:

OR( 0.1 , 0.9 ) = { 0.9871232235565717 }
Training network. Please wait...
Training finished.

AND({ 0.0 }, { 0.0 }) = { 0.02215851007657903 }
AND({ 1.0 }, { 0.0 }) = { 0.9870842865449098 }
AND({ 0.0 }, { 1.0 }) = { 0.986942866417925 }
AND({ 1.0 }, { 1.0 }) = { 0.9960623707490505 }
Similar data test:

AND( 0.1 , 0.9 ) = { 0.9869597153430022 }
```

CSL 411 –Artificial Intelligence Lab

FALL 2017

Lab # 11

Prolog-I

COURSE INSTRUCTOR: TARIQ SIDDIQUE

LAB ENGINEER: TARWAN KUMAR

DEPARTMENT OF COMPUTER SCIENCE

BAHRIA UNIVERSITY, KARACHI CAMPUS

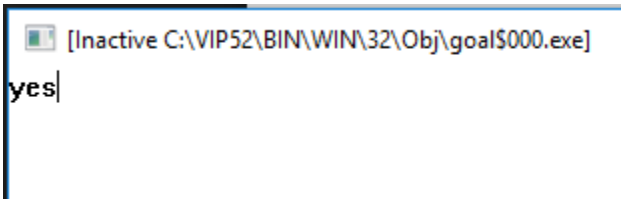
Task # 1

Task # 1a

Code

```
PREDICATES

nondeterm likes(symbol,symbol)

CLAUSES

likes(ellen,tennis).

likes(john,football).

likes(tom,baseball).

likes(eric,swimming).

likes(mark,tennis).

likes(bill,Activity):-


likes(tom, Activity).


GOAL

likes(bill, baseball).
```

Output

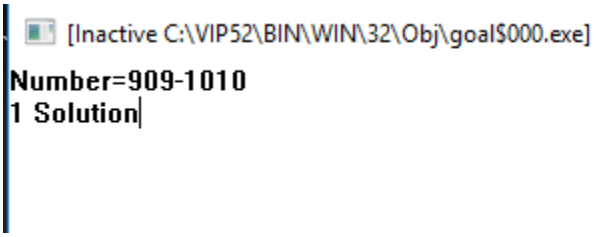


Task # 1b

Code

```
PREDICATES

phone_number(symbol,symbol)

CLAUSES

phone_number("Albert","EZY-3665").

phone_number("Betty","555-5233").

phone_number("Carol","909-1010").

phone_number("Dorothy","438-8400").

Goal:

phone_number("Carol", Number).

%phone_number(Who, "438-8400").

%phone_number("Albert", Number).

%phone_number(Who, Number).
```

Output

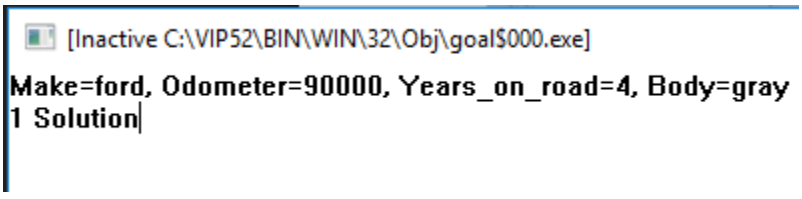


Task # 1c

Code

```
PREDICATES

car(symbol,long,integer,symbol,long)

truck(symbol,long,integer,symbol,long)

nondeterm vehicle(symbol,long,integer,symbol,long)

CLAUSES

car(chrysler,130000,3,red,12000).

car(ford,90000,4,gray,25000).

car(datsun,8000,1,red,30000).

truck(ford,80000,6,blue,8000).

truck(datsun,50000,5,orange,20000).

truck(toyota,25000,2,black,25000).

vehicle(Make,Odometer,Age,Color,Price):-

car(Make,Odometer,Age,Color,Price);

truck(Make,Odometer,Age,Color,Price).

GOAL

car(Make, Odometer, Years_on_road, Body, 25000).
```

Output



Task # 1d

Code

```
PREDICATES
nondeterm can_buy(symbol, symbol)
nondeterm person(symbol)
nondeterm car(symbol)
likes(symbol, symbol)
for_sale(symbol)
CLAUSES
can_buy(X,Y):- person(X),
car(Y),
likes(X,Y),-
for_sale(Y).
person(kelly).
person(judy).
person(ellen).
person(mark).
car(lemon).
car(hot_rod).
likes(kelly, hot_rod).
likes(judy, pizza).
likes(ellen, tennis).
likes(mark, tennis).
for_sale(pizza).
```

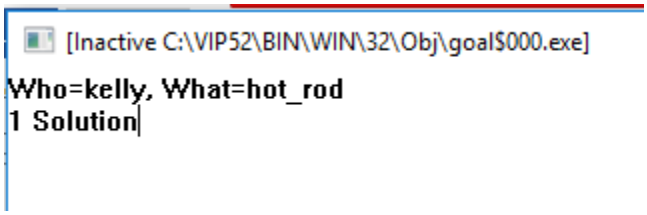
for_sale(lemon).

for_sale(hot_rod).

GOAL

can_buy(Who,What).

Output



Task # 2

Code

```
PREDICATES

person_info(symbol,symbol,symbol,symbol)

CLAUSES

person_info("Rahim","15","FootBall","Dog").

person_info("Mohsin","11","VolleyBall","Cat").

person_info("Sohail","25","Card","Cow").

person_info("Kamal","30","Swimming","Dog").

person_info("Haseeb","11","FootBall","Goat").

person_info("Shakeel","25","VolleyBall","Cat").

person_info("Abrar","15","Swimming","Dog").

person_info("Raju","30","Swimming","Dog").

person_info("Javed","40","FootBall","Cow").

person_info("Waleed","30","VolleyBall","Cat").

GOAL:

person_info(P_name,Age,Hobby,Pet),Age<="15".
```

Output

```
P_name=Rahim, Age=15, Hobby=FootBall, Pet=Dog
P_name=Mohsin, Age=11, Hobby=VolleyBall, Pet=Cat
P_name=Haseeb, Age=11, Hobby=FootBall, Pet=Goat
P_name=Abrar, Age=15, Hobby=Swimming, Pet=Dog
4 Solutions
```

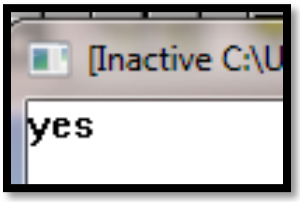
Task # 3

Task # 3a

Code

```
PREDICATES
 nondeterm star(symbol)
 nondeterm planet(symbol)
 nondeterm orbits(symbol,symbol)
 nondeterm heavenlyBody(symbol)
 nondeterm geoCentric(symbol)

CLAUSES
 star(sun).
 planet(venus):- orbits(venus,sun).
 orbits(moon,earth).
 heavenlyBody(satellite):- orbits(satellite,satellite).
 heavenlyBody(x):- star(x),planet(y).
 geoCentric(solarSystem):-orbits(star,planet).

GOAL
 star(sun).
```

Output



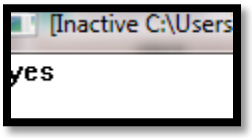
Task # 3b

Code

```
PREDICATES
 nondeterm star(symbol)
 nondeterm planet(symbol)
 nondeterm orbits(symbol,symbol)
 nondeterm heavenlyBody(symbol)
 nondeterm geoCentric(symbol)
CLAUSES
```

```
star(sun).
 planet(venus):- orbits(venus,sun).
 orbits(moon,earth).
 orbits(x,y).
 heavenlyBody(satellite):- orbits(satellite,satellite).
 heavenlyBody(x):- star(x),planet(y).
 geoCentric(solarSystem):- orbits(star,planet).
GOAL
 %star(sun).
 orbits(moon,earth).
```

Output

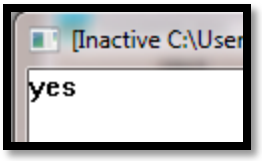


Task # 3c

Code

```
PREDICATES
 nondeterm star(symbol)
 nondeterm planet(symbol)
 nondeterm orbits(symbol,symbol)
 %nondeterm heavenlyBody(symbol)
 %nondeterm geoCentric(symbol)
 nondeterm helioCentric(symbol)
CLAUSES
 star(sun).
 planet(earth).
 planet(venus).
 orbits(moon,earth).
 orbits(planet,star).
 helioCentric(solarsSystem):-orbits(planet,star).
GOAL
 %star(sun).
 %orbits(moon,earth).
 helioCentric(solarSystem).
```

Output

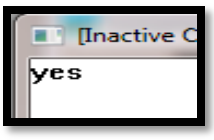


Task # 3d

Code

```
PREDICATES
 nondeterm star(symbol)
 nondeterm planet(symbol)
 nondeterm orbits(symbol,symbol)
 nondeterm heavenlyBody(symbol)
 nondeterm geoCentric(symbol)
 nondeterm helioCentric(symbol)
CLAUSES
 star(sun).
 planet(earth).
 planet(venus).
 heavenlyBody(star).
 heavenlyBody(planet).
 heavenlyBody(satellite):-orbits(satellite,satellite).
 orbits(moon,earth).
 orbits(planet,star).
 geoCentric(solarSystem):-orbits(star,planet).
 helioCentric(solarSystem):-orbits(planet,star).
GOAL
 %star(sun).
 %orbits(moon,earth).
 %helioCentric(solarSystem).
 heavenlyBody(satellite).
```

Output



CSL 411 –Artificial Intelligence Lab

FALL 2017

Lab # 12

Prolog-II

COURSE INSTRUCTOR: TARIQ SIDDIQUE

LAB ENGINEER: TARWAN KUMAR

DEPARTMENT OF COMPUTER SCIENCE

BAHRIA UNIVERSITY, KARACHI CAMPUS

Task # 1

Code

```
DOMAINS  list = integer*. NewList = integer*. Item = integer

PREDICATES
remove(Item, List, Newlist)

CLAUSES
remove(X, [], []).
remove(X,[H|T],ProcessedTail):-
H = X,
!,
remove(X,T, ProcessedTail).
remove(X,[H|T],[H|ProcessedTail]):-
remove(X,T, ProcessedTail).

GOAL
remove (-45,[-45, 9, 165, 10], A).
```

Output

Task # 2

Code

```
DOMAINS
List = integer*

PREDICATES
asclist(List)

CLAUSES
asclist([Num]).
asclist([First|[Second|Rest]]):-
First < Second, asclist([Second|Rest]).

GOAL
asclist([3,4,5]).
```

Output

```
yes
```

Task # 3

Code

```
domains
list=integer*
predicates
sum(list,integer)
nondeterm sumN(symbol,list)

clauses
sumN(X,L):-
        sum(L,Sum),
        write(X),
        write(":",Sum),
        write("\n").
                sum([],0).
        sum([X|Tail],Sum):-
        sum(Tail,Temp),
        Sum=Temp+X.

Goal
sumN(john,[11,7,3]).
```

Output