|  |  |  |  |
| --- | --- | --- | --- |
| **S.no** | **Date** | **Name of the Experiment** | **Signature** |
| **1** |  | Crash Course on Python – I & II |  |
| **2** |  | Implementation of Binary Search Algorithm in Python |  |
| **3** |  | Implementation of Bubble Sort Algorithm in Python |  |
| **4** |  | Implementation of Best First Search Algorithm |  |
| **5** |  | Implementation of A\* Algorithm |  |
| **6** |  | Building Semantic Network in Python |  |
| **7** |  | Design and Deployment of an Expert System |  |
| **8** |  | Building Bayesian Networks in Python |  |
| **9** |  | Building Markov Chain Model |  |
| Submitted for the Continuous Assessment\_\_\_ Practial Examination Held on \_\_\_\_\_\_\_\_\_\_\_\_  Internal Examiner External Examiner  Name: Name: | | | |

**CONTENTS**

**EXERCISE: 1 Crash Course on Python – I & II**

**Aim :**

To study and practice about the basic datatypes ,Conditional Statement, Loops and Function of python using Jupyter Notebook.

**Requirements**:

1. Jupyter Notebook

**Coding**

**Numeric datatype:**

***# Integer Example***

num1 = 10

print("Value:", num1, "Type:", type(num1))

***#Float Example***

num2 = 3.14

print("Value:", num2, "Type:", type(num2))

***# Complex Example***

num3 = 2 + 3j

print("Value:", num3, "Type:", type(num3))

**String datatype:**

name1="HOLA FOLKS"

print("Value:", name1, "Type:", type(name1))

**List datatype:**

name2 = [1, 2, 3, "four", "five"]

print("Value:", name2, "Type:", type(name2))

**Tuple datatype:**

name3 = (1, 2, 3, "four", "five")

print("Value:", name3, "Type:", type(name3))

**Set datatype:**

name4 = {1, 2, 3, "four", "five"}

print("Value:", name4, "Type:", type(name4))

**Dictionary datatype:**

name5 = {"name": "Vikneshraj D", "age": 18, "city": "Hubli"}

print("Value:", name5, "Type:", type(name5))

**LOOPS**

***#For Loops***

fruits = ["apple", "banana", "cherry"]

for fruit in fruits:

print(fruit)

***#While Loop***

A**=**input("Enter the number: ")

val **=** 0

i **=** 0

**while** i **<=** int(A):

val **+=** i

i **+=** 1

print(f"The sum is {val}")

**FUNCTION**

def add\_numbers(x, y):

sum\_result = x + y

return sum\_result

result = add\_numbers(3, 4)

print(result)

**Conditional Statement**

x = int(input("Enter The number"))

if x > 0 :

print("The number is positive ")

elif x < 0 :

print("The number is negative")

else:

print("The number is ZERO")

**Result:**

Thus the way we declare and execute basic datatypes ,Conditional Statement, Loops and Function of python is verified Successfully

**EXERCISE: 1 Crash Course on Python – I & II**

**OUTPUT:**

**Numeric datatype:**

***# Integer Example***

**Value:** 10 **Type:** <class 'int'>

***# Float Example***

**Value:** 10 **Type**: <class 'float'>

***# Complex Number Example***

**Value:** 2+3j **Type:** <class 'complex'>

**String datatype:**

**Value:** HOLA FOLKS **Type:** <class 'str'>

**List datatype:**

**Value:** [1, 2, 3, 'four', 'five'] **Type:** <class 'list'>

**Tuple datatype:**

**Value:** (1, 2, 3, 'four', 'five') **Type:** <class 'tuple'>

**Set datatype:**

**Value:** {1, 2, 3, 'four', 'five'} **Type:** <class 'set'>

**Dictionary datatype:**

**Value:** {'name': 'Vikneshraj D', 'age': 18, 'city': 'Hubli'}

**Type:** <class 'dict'>

***#For Loops***

apple

banana

cherry

***#While Loop***

Enter the number: 10

The sum is 55

**FUNCTION**

7

**Conditional Statement**

Enter The number 10

The number is positive

**EXERCISE: 2 Implementation of Binary Search Algorithm in Python**

**Aim:**

To Implement the Binary Search Algorithm in Python

**Requirements**:

1.Jupyter Notebook

**Coding**

data = [30,31,18,15,20,19,11,1,9,10,7,6,4,5,16,12,22,25,27,28,35,33,32,38,37,21]

data.sort()

print(data)

elem = int(input("Enter the search element:"))

**def** binary\_search (data, elem):

low = 0

high = len(data) - 1

**while** low <= high:

middle = (low + high)//2

**if** data[middle] == elem:

print(f"The searching element {elem} present at index value {middle} in dataset")

**break**

**elif** data[middle] > elem:

high = middle - 1

**else :**

low = middle + 1

**if** data[middle] != elem:

print(f"The searching element {elem} is not present in dataset")

**return -1**

binary\_search (data, elem)

**Result:**

Thus the way we declare and execute the Binary Search Algorithm

in Python is Verified Successfully

**EXERCISE:2 Implementation of Binary Search Algorithm in Python**

**OUTPUT:**

[1, 4, 5, 6, 7, 9, 10, 11, 12, 15, 16, 18, 19, 20, 21, 22, 25, 27, 28, 30, 31, 32, 33, 35, 37,38]

Enter the search element:10

The searching element 10 present at index value 6 in dataset

**EXERCISE:3 Implementation of Bubble Sort Algorithm in Python**

**Aim :**

To Implement the Bubble Sort Algorithm in Python

**Requirements:**

1. **Jupyter Notebook**

**Coding:**

**def** bubbleSort(data):

**for** i in range(len(data)):

**for** j in range(0, len(data) - i - 1):

**if** data[j] **>** data[j + 1]:

temp = data[j]

data [j] = data [j + 1]

data [j + 1] = temp

data = [-2, 45, 0, 11, 9, 15, -11, 21, 12]

print('Before Sorting the Array in Ascending Order:')

print(data)

bubbleSort(data)

print('After Before Sorting the Array in Ascending Order:')

print(data)

**Result:**

Thus the way we declare and execute Bubble Sort Algorithm in Python is Verified Successfully

**EXERCISE: 3 Implementation of Bubble Sort Algorithm in Python**

**OUTPUT:**

Before Sorting the Array in Ascending Order:

[-2, 45, 0, 11, 9, 15, -11, 21, 12]

After Before Sorting the Array in Ascending

Order:

[-11, -2, 0, 9, 11, 12, 15, 21, 45]

**EXERCISE: 4 Implementation of Best First Search Algorithm**

**Aim:**

To Implement the Best First Search Algorithm in Python

**Requirements**:

1. Jupyter Notebook

**Coding**

**from** queue **import** PriorityQueue

v **=** 14

graph **=**[[] **for** i **in** range (v)]

def best\_first\_search(actual\_src, target, n):

visited = [False] \* n

pq = PriorityQueue()

pq.put((0, actual\_src))

visited[actual\_src] = True

while pq.empty() == False:

u = pq.get()[1]

print(u, end=” “)

if u == target:

break

for v, c in graph[u]:

if visited[v] == False:

visited[v] = True

pq.put((c, v))

print()

def addedge(x, y, cost):

graph[x].append((y, cost))

graph[y].append((x, cost))

addedge(0, 1, 3)

addedge(0, 2, 6)

addedge(0, 3, 5)

addedge(1, 4, 9)

addedge(1, 5, 8)

addedge(2, 6, 12)

addedge(2, 7, 14)

addedge(3, 8, 7)

addedge(8, 9, 5)

addedge(8, 10, 6)

addedge(9, 11, 1)

addedge(9, 12, 10)

addedge(9, 13, 2)

source = 0

target = 14

best\_first\_search(source, target, v)

**Result:**

Thus the way we declare and execute Best First Search Algorithm in Python is Verified Successfully

**EXERCISE: 4 Implementation of Best First Search Algorithm**

**OUTPUT:**

0 1 3 2 8 9 11 13 10 5 4 12 6 7

**EXERCISE: 5 Implementation of A\* Algorithm**

**Aim**:

To Implement the A\* Algorithm by the use of python library networkx

**Requirements**:

1. Jupyter Notebook

**Coding**

Pip install **networkx**

Import **networkx** as **nx**

Import **matplotlib.pyplot** as plt

**%**matplotlib inline

Def dist(a, b):

(x1, y1) = a

(x2, y2) = b

Return (( x1 – x2) \*\* 2 + (y1 – y2) \*\*2) \*\* 0.5

G = nx.grid\_graph(dim=[4, 4])

Nx.set\_edge\_attributes(G, {e: e[1][0] \* 2 for e in G.edges()}, “cost”)

pos **=** nx**.**spring\_layout(G)

nx**.**draw(G, pos, with\_labels **=** **True**, node\_color**=**"#00FFFF")

edge\_labels **=** nx**.**get\_edge\_attributes(G, "cost")

nx**.**draw\_networkx\_edge\_labels(G, pos, edge\_labels **=** edge\_labels)

plt**.**show()")

path **=** nx**.**astar\_path(G, (1, 0), (3, 2), heuristic **=** dist, weight **=**"cost")

length **=** nx**.**astar\_path\_length(G, (1, 0), (3, 2), heuristic **=** dist, weight **=**"cost”)

print(‘Path :’, path)

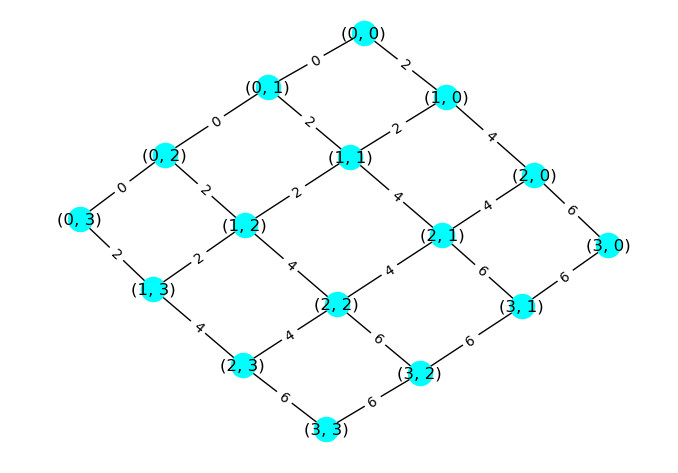
print(‘Path Length’, lengt)

**Result:**

Thus the way we declare and execute A\* Algorithm by the use of Python library networkx is Verified Successfully

**EXERCISE: 5 Implementation of A\* Algorithm**

**OUTPUT:**



**EXERCISE: 6**  **Building Semantic Network in Python**

**Aim :**

To Build a Semantic Network by the use of python library network

**Requirements**:

1. Jupyter Notebook

**Coding**

Import **networkx** as **nx**

Import **matplotlib.pyplot** as plt

%matplotlib notebook

Graph\_Mark =nx.DiGraph(Info = “Mark’s Details”)

Graph\_Mark.add\_node(“Mark”,pos=(0,0))

Graph\_Mark.add\_node(“cat”,pos=(-2,6))

Graph\_Mark.add\_node(“student”,pos=(2,-5))

Graph\_Mark.add\_node(“animal”,pos=(1,6))

Graph\_Mark.add\_node(“california”,pos=(4,6))

Graph\_Mark.add\_node(“spinoff”,pos=(-5,-5))

Graph\_Mark.add\_node(“soccer”,pos=(-5,2))

Graph\_Mark.add\_node(“sports club”,pos=(0,-8))

Graph\_Mark.add\_node(“CSU”,pos=(5,-1))

Pos=nx.get\_node\_attributes(graph\_Mark,”pos”)

graph\_Mark.add\_edge("Mark", "cat", weight="has a")

graph\_Mark.add\_edge("Mark", "student", weight="is a")

graph\_Mark.add\_edge("cat", "animal", weight="is a")

graph\_Mark.add\_edge("Mark", "soccer", weight="plays")

graph\_Mark.add\_edge("Mark", "spinoff", weight="is a part of")

graph\_Mark.add\_edge("Mark", "california", weight="lives in")

graph\_Mark.add\_edge("Mark", "animal", weight="loves")

graph\_Mark.add\_edge("student", "CSU", weight="in")

graph\_Mark.add\_edge("spinoff", "sports club", weight="is a")

graph\_Mark.add\_edge("CSU", "california", weight="is in")

weight =nx.get\_edge\_attributes(graph\_Mark, "weight")

plt.figure()

nx.draw\_networkx(graph\_Mark,pos,font\_weight=’bold’,node\_size=2000,

font\_size= 10)

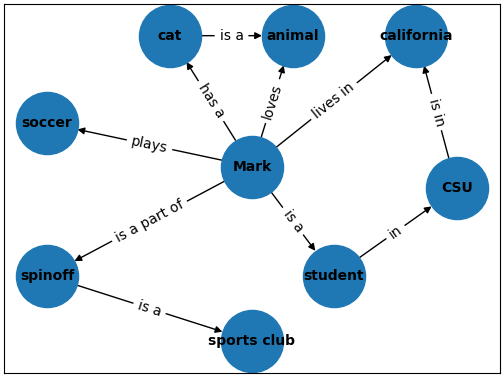
nx.draw\_networkx\_edge\_labels(graph\_Mark,pos,edge\_labels=weight)

**Result:**

Thus the way we declare and execute Semantic Network by the use of python library networkx is Verified Successfully

**EXERCISE: 6**  **Building Semantic Network in Python**

**OUTPUT:**



**EXERCISE:7 Design and Deployment of an Expert System**

**Aim :**

To Design and Deployment of an Expert System by the use of library experta

**Requirements**:

1. Jupyter Notebook

**Coding:**

pip install **experta**

from **experta** import \*

class meds(KnowledgeEngine):

@DefFacts()

def \_initial\_action(self):

yield Fact(action ='load')

***# Starting Questions***

@Rule(Fact(action = 'load'), NOT(Fact(fulltime = W())))

def start\_quest(self):

print("Welcome to the Medical Expert System. ")

self.declare(Fact(intro = input("Please enter your name: ")))

self.declare(Fact(fulltime = input("Do you want to enter the Medical Expert System? ")))

***# Not interested in entering***

@Rule(Fact(action = 'load'), (Fact(fulltime = 'no')))

def exiting(self):

print("Thank you!")

***# Rule 1: Checking Covid Symptom #1 - Fever***

@Rule(Fact(action = 'load'), (Fact(fulltime = 'yes')))

def fever\_check(self):

self.declare(Fact(Fever = input("Do you have fever for the last few days? ")))

***# Rule 2: Checking Covid Symptom #2 - Dry Cough***

@Rule(Fact(action = 'load'), AND(Fact(fulltime = 'yes'), NOT(Fact(Fever = 'not sure'))))

def cough\_check(self):

self.declare(Fact(Cough = input("Do you have dry cough for the last few days? ")))

***# Rule 3: Checking Covid Symptom #3 - Tiredness***

@Rule(Fact(action='load'), AND(Fact(fulltime = 'yes'), NOT(Fact(Fever = 'not sure')), NOT(Fact(Cough = 'not sure'))))

def tired\_check(self):

self.declare(Fact(Tired = input("Have you been feeling tired? ")))

***# Diagnosis uptil Rule 3***

@Rule(Fact(action='load'), AND(Fact(fulltime='yes'), AND(Fact(Fever = 'yes'), Fact(Cough = 'no'), Fact(Tired = 'no'))))

def accept\_1(self):

print("You have fever, please take rest and have Paracetamol")

@Rule(Fact(action='load'), AND(Fact(fulltime='yes'), AND(Fact(Fever = 'no'), Fact(Cough = 'yes'), Fact(Tired = 'no'))))

def accept\_2(self):

print("You just have dry cough. Please gargle, steam and have lots of hot water.")

@Rule(Fact(action='load'), AND(Fact(fulltime='yes'), Fact(Fever = 'yes'), Fact(Cough = 'yes'), Fact(Tired = 'yes')))

def accept\_3(self):

print("You are showing symptoms of COVID-19. Please get yourself tested and stay quarentined.")

@Rule(Fact(action='load'), AND(Fact(fulltime='yes'), Fact(Fever = 'no'), Fact(Cough = 'yes'), Fact(Tired = 'yes')))

def accept\_4(self):

print("Please visit the doctor as you may have a throat infection.")

@Rule(Fact(action='load'), AND(Fact(fulltime='yes'), Fact(Fever = 'yes'), Fact(Cough = 'no'), Fact(Tired = 'yes')))

def accept\_5(self):

print("You may be having a viral infection. Take ample rest. If it presists please visit a doctor.")

***# Enter advance expert system****.*

@Rule(Fact(action = 'load'), AND(Fact(fulltime = 'yes'), OR(Fact(Fever = 'yes'), Fact(Fever = 'no')), OR(Fact(Cough = 'yes'), Fact(Cough = 'no')),

OR(Fact(Tired = 'yes'), Fact(Tired = 'no'))))

def adv\_expt(self):

print("You have completed the simple medical expert system.")

self.declare(Fact(dep\_dive = input("Do you want to dive deeper into the expert system? ")))

***# Deciding.***

@Rule(Fact(action = 'load'), AND(Fact(fulltime = 'yes'), Fact(dep\_dive = 'no')))

def div\_reject(self):

print("Thank you for using our expert system.")

***# Rule 4: Checking Covid Symptom #4 - Shortness of breath***

@Rule(Fact(action = 'load'), AND(Fact(fulltime = 'yes'), Fact(dep\_dive = 'yes')))

def breath(self):

self.declare(Fact(breathing = input("Have you been experiencing shortness of breath? ")))

***# Rule 5: Checking Covid Symptom #5 - Chest Pain***

@Rule(Fact(action = 'load'), AND(Fact(fulltime = 'yes'), Fact(dep\_dive = 'yes'),OR(Fact(breathing = 'yes'), Fact(breathing = 'no'))))

def chest\_pain(self):

self.declare(Fact(chest = input("Have you been experiencing acute chest pain or pressure? ")))

***# Rule 6: Checking Covid Symptom #6 - Loss of speech or movement***

@Rule(Fact(action = 'load'), AND(Fact(fulltime = 'yes'), Fact(dep\_dive = 'yes'), OR(Fact(breathing = 'yes'), Fact(breathing = 'no')),

OR(Fact(chest = 'yes'), Fact(chest = 'no'))))

def speech\_loss(self):

self.declare(Fact(loss = input("Have you been experiencing any loss of speech or movement? ")))

***#Diagnosis 4-6***

@Rule(Fact(action='load'), AND(Fact(fulltime='yes'), Fact(dep\_dive = 'yes'), Fact(breathing = 'yes'), Fact(loss = 'no'), Fact(chest = 'no')))

def accept\_6(self):

print("You seem to be having shortness of breath. Even if you are not COVID positve, this is serious.")

print("Go to the doctor immediately.")

@Rule(Fact(action='load'), AND(Fact(fulltime='yes'), Fact(dep\_dive = 'yes'), Fact(breathing = 'no'), Fact(loss = 'yes'), Fact(chest = 'no')))

def accept\_7(self):

print("You seem to be having either loss of speech or movement. Even if you are not COVID positve, this is serious.")

print("Go to the doctor immediately.")

@Rule(Fact(action='load'), AND(Fact(fulltime='yes'), Fact(dep\_dive = 'yes'), Fact(breathing = 'no'), Fact(loss = 'no'), Fact(chest = 'yes')))

def accept\_8(self):

print("You seem to be having chest pain. Even if you are not COVID positve, this is serious.")

print("Go to the doctor immediately.")

@Rule(Fact(action='load'), AND(Fact(fulltime='yes'), Fact(dep\_dive = 'yes'), Fact(breathing = 'yes'), Fact(loss = 'no'), Fact(chest = 'yes')))

def accept\_9(self):

print("You seem to be having chest pain and shortness of breath. Even if you are not COVID positve, this is serious.")

print("Go to the doctor immediately.")

@Rule(Fact(action='load'), AND(Fact(fulltime='yes'), Fact(dep\_dive = 'yes'), Fact(breathing = 'no'), Fact(loss = 'yes'), Fact(chest = 'yes')))

def accept\_10(self):

print("You seem to be having chest pain and loss of speech or motion. Even if you are not COVID positve, this is serious.")

print("Go to the doctor immediately.")

@Rule(Fact(action='load'), AND(Fact(fulltime='yes'), Fact(dep\_dive = 'yes'), Fact(breathing = 'yes'), Fact(loss = 'yes'), Fact(chest = 'no')))

def accept\_11(self):

print("You seem to be having shortness of breath and loss of speech or movement. Even if you are not COVID positve, this is serious.")

print("Go to the doctor immediately.")

@Rule(Fact(action='load'), AND(Fact(fulltime='yes'), Fact(dep\_dive = 'yes'), Fact(breathing = 'yes'), Fact(loss = 'yes'), Fact(chest = 'yes')))

def accept\_12(self):

print("You seem to be having chest pain, shortness of breathing and loss of speech or movement Even if you are not COVID positve, this is serious.")

print("Go to the doctor immediately.")

Engine = meds()

Engine.reset()

Engine.run()

**Result:**

Thus the way we declare and execute the Expert System by the use of library experta in Python is verified Successfully.

**EXERCISE: 7 Design and Deployment of an Expert System**

**Output:**

Welcome to the Medical Expert system.

please enter your name : **Vikneshraj D**

Do you want to enter the Medical expert system? **yes**

Do you have fever for the last few days? **yes**

Do you have dry cough for the last few days? **yes**

Have you been feeling tired? **yes**

You are showing symptoms of COVID-19.Please get yourself tested and stay quarantined.

You have completed the simple medical expert system.

Do you want to dive deeper into the expert system? **yes**

Have you been experiencing shortness of breath? **yes**

Have you been experiencing acute chest pain or pressure? **yes**

Have you been experiencing any loss of speech or movement? **yes**

You seem to be having chest pain and shortness of breath and loss of speech or movement.

Even if you are not COVID positive, this is serious. Go to the doctor immediately.

**EXERCISE: 8 Building Bayesian Networks in Python**

**Aim :**

To Build a Bayesian Networks by the use of python library protopunica

**Requirements**:

1. Jupyter Notebook

**Coding**

Pip install **protopunica**

From **protopunica** import **\***

smoking **=** Node(DiscreteDistribution({"High smoking":0.7,"Low smoking":0.3}),name**=**"smoking")

asbes\_consum **=**Node(DiscreteDistribution({"High Cons":0.3,"Low Cons":0.7}),name**=**"asbes\_consum")

cancer **=** Node(ConditionalProbabilityTable([

["High smoking", "High Cons", "Pos", 0.4],

["High smoking", "High Cons", "Neg", 0.6],

["High smoking", "Low Cons", "Pos", 0.3],

["High smoking", "Low Cons", "Neg", 0.7],

["Low smoking", "Low Cons", "Pos", 0.1],

["Low smoking", "Low Cons", "Neg", 0.9],

["Low smoking", "High Cons", "Pos", 0.02],

["Low smoking", "High Cons", "Neg", 0.98],],

[smoking**.**distribution, asbes\_consum**.**distribution]), name**=**"cancer")

scan **=** Node(ConditionalProbabilityTable([

["Pos","scan\_pos",0.8],

["Pos","scan\_neg",0.2],

["Neg","scan\_pos",0.1],

["Neg","scan\_neg",0.9]],[cancer**.**distribution]),name**=**"scan")

Blood\_vomiting **=** Node(ConditionalProbabilityTable([

["Pos","B.V\_pos",0.7],

["Pos","B.V\_neg",0.3],

["Neg","B.V\_pos",0.2],

["Neg","B.V\_neg",0.8]],[cancer**.**distribution]),name**=**"Blood\_vomiting ")

model=BayesianNetwork()

model**.**add\_states(smoking,asbes\_consum,cancer,scan,Blood\_vomiting)

model**.**add\_edge(smoking,cancer)­­­­­­­

model**.**add\_edge(asbes\_consum,cancer)

model**.**add\_edge(cancer,scan)

model**.**add\_edge(cancer,Blood\_vomiting)

model**.**bake()

model

probability**=**model**.**probability([["Low smoking","Low Cons","Pos","scan\_pos","B.V\_pos"]])

probability

probability**=**model**.**probability([["High smoking","High Cons","Pos","scan\_pos","B.V\_pos"]])

probability

**>>>** print(model**.**predict([["Low smoking", "Low Cons","Neg","scan\_pos",None]]))

predictions**=** model**.**predict\_proba({"Blood\_vomiting": "B.V\_pos"})

predictions

predictions**=** model**.**predict\_proba({"scan": "scan\_pos"})

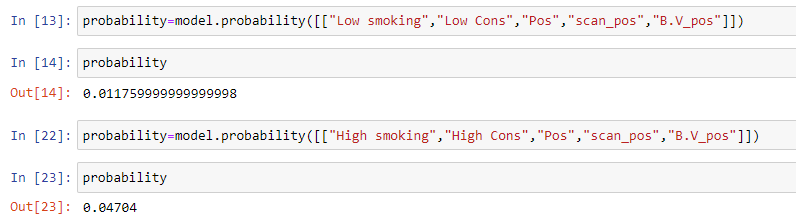
predictions

**Result:**

Thus the way we declare and execute Bayesian Networks by the use of python library protopunica is Verified Successfully

**EXERCISE: 8 Building Bayesian Networks in Python**

**OUTPUT:**

****

**EXERCISE:9 Building Markov Chain Model**

**Aim :**

To Build a Markov Chain Model by the use of python library protopunica and numpy

**Requirements**:

1. Jupyter Notebook

**Coding**

from **protopunica** import **\***

import **numpy** as **np**

start **=** DiscreteDistribution({"PIZZA":1,"Veg":0})

Transitions **=** ConditionalProbabilityTable([

["PIZZA", "PIZZA", 0.75],

["PIZZA", "VEG", 0.25],

["VEG", "VEG", 0.6],

["VEG", "PIZZA", 0.4],], [start])

Model**=**MarkovChain([start,Transitions])

Random\_samples**=**Model**.**sample(2)

print(Random\_samples)

log\_probability **=** Model**.**log\_probability(Random\_samples)

Probability\_of\_Occurance**=** np**.**exp(log\_probability)

Probability\_of\_Occurance

log\_probability\_Food\_Sequence **=** Model**.**log\_probability(["PIZZA","PIZZA","PIZZA"])

Probability\_of\_Food **=** np**.**exp(log\_probability\_Food\_Sequence )

print (Probability\_of\_Food)

**Result:**

Thus the way we declare and execute Markov Chain Model by the use

of python library protopunica and numpy is Verified Successfully

**EXERCISE: 9 Building Markov Chain Model**

**OUTPUT:**

**Probability of Occurance**

0.25

**Probability of Food**

0.5625