Applied Artificial Intelligence

A PRACTICAL REPORT

ON

Applied Artificial Intelligence

SUBMITTED BY

Mr. RAHUL KEWAT

Roll No: 22006

UNDER THE GUIDANCE OF

Mrs. Anshika Gautam

Submitted in fulfillment of the requirements for qualifying

MSc. IT Part II Semester - III Examination 2023-2024

University of Mumbai

Department of Information Technology

R.D. & S.H National College of Arts, Commerce &
S.W.A. Science College Bandra (West), Mumbai – 400 050

R. D. & S. H. National & S. W. A. Science College

Bandra (W), Mumbai – 400050.

Department of Information Technology

M.Sc. (IT - SEMESTER III)

This is to certify that Applied Artificial Intelligence Practicals performed at R.D & S.H National & S.W.A. Science College by Mr. **Rahul Kewat** holding Seat No. _____ studying Master of Science in Information Technology Semester – III has been satisfactorily completed as prescribed by the University of Mumbai, during the year 2023– 2024.

Subject In-Charge **Coordinator In-Charge** **Externa Examiner**

College Stamp

INDEX

Practical: 1

Aim: Design an Expert System using AIML

Writeup:

Description:

Car health expert system?

A car health expert system is a computer program that uses artificial intelligence to diagnose car problems based on a set of symptoms.

The system is typically rule-based, meaning that it contains a set of rules that define the relationships between symptoms and problems.

When the user enters a set of symptoms, the system uses the rules to determine the most likely problem.

This expert system can be used to determine if a car is healthy based on its symptoms.

If the car is not healthy the system will recommend a repair.

If the car is healthy the the system will recommend a your is healthy.

Step 1: Platform used to build a car health expert system

- Python is a powerful language that can be used to build car health expert systems

Step 2:

Source Code:

```
print("Car health expert system by Rahul")

def is_car_healthy(symptoms):

  if "engine_light_on" in symptoms or "low_oil_pressure" in symptoms:
   return False
  else:
     return True

car = input('Enter car issue:')
def get_car_repair(symptoms):

    if is_car_healthy(symptoms):
        return "Your car is healthy"
    elif "engine_light_on" in symptoms:
        return "Get your engine checked"
    elif "low_oil_pressure" in symptoms:
        return "Add oil to your car"
    else:
        return "I don't know what's wrong with your car"
def main():
    symptoms = [ "low_oil_pressure",]


    if is_car_healthy(symptoms):
        print("Your car is healthy")
```

```
    else:
        print("Your car needs repair")
        print("The repair is:", get_car_repair(symptoms))
main()
```

Step 3:

Output:

```
PS C:\Users\Dell\Desktop\MSC IT Part 1> & C:/Users/Dell/AppData/Local/Programs/Python/Python310/python.exe "c
Car health expert system by Rahul
Enter car issue:engine_light_on
Your car needs repair
The repair is: Get your engine checked
PS C:\Users\Dell\Desktop\MSC IT Part 1> & C:/Users/Dell/AppData/Local/Programs/Python/Python310/python.exe "c
:/Users/Dell/Desktop/MSC IT Part 1/expert2.py"
Car health expert system by Rahul
Enter car issue:check_break
Your car is healthy
PS C:\Users\Dell\Desktop\MSC IT Part 1> & C:/Users/Dell/AppData/Local/Programs/Python/Python310/python.exe "c
:/Users/Dell/Desktop/MSC IT Part 1/expert2.py"
Car health expert system by Rahul
Enter car issue:low_oil_pressure
Your car needs repair
The repair is: Add oil to your car
PS C:\Users\Dell\Desktop\MSC IT Part 1>
```

Practical: 2

Aim: Design a bot using AIML

Writeup:

Description:

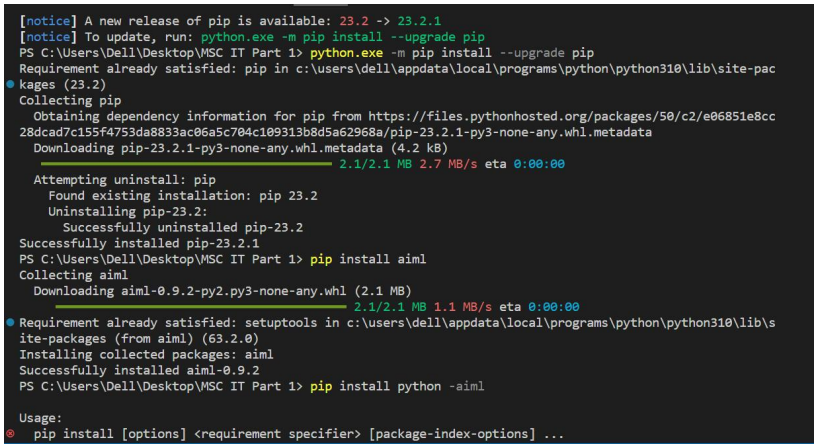
What is AIML?

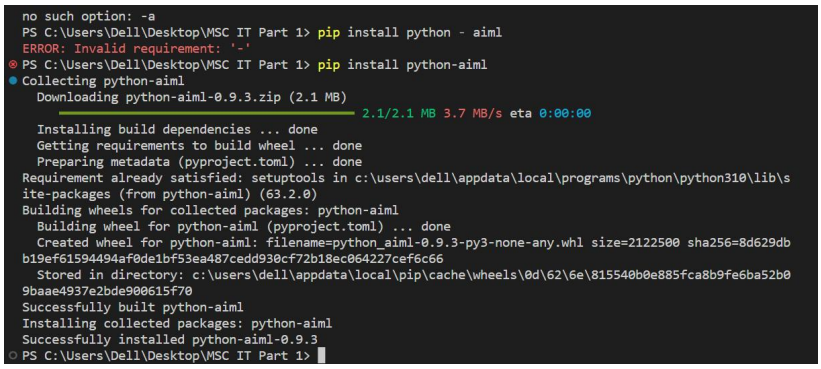
AIML stands for Artificial Intelligence Modelling Language. AIML is an XML based markup language meant to create artificial intelligent applications. AIML makes it possible to create human interfaces while keeping the implementation simple to program, easy to understand and highly maintainable.

AIML Tags/Description

- <aiml>– defines the beginning and end of a AIML document.
- <category>– defines the unit of knowledge in bot's knowledge base.
- <pattern>– defines the pattern to match what a user may input to an bot.
- <template>– defines the response of a bot to user's input.

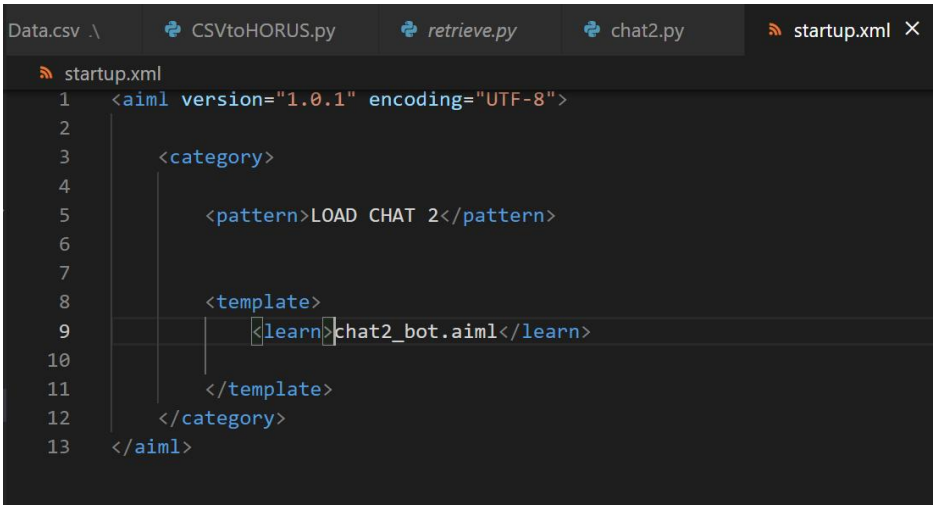
Step 1: install aiml





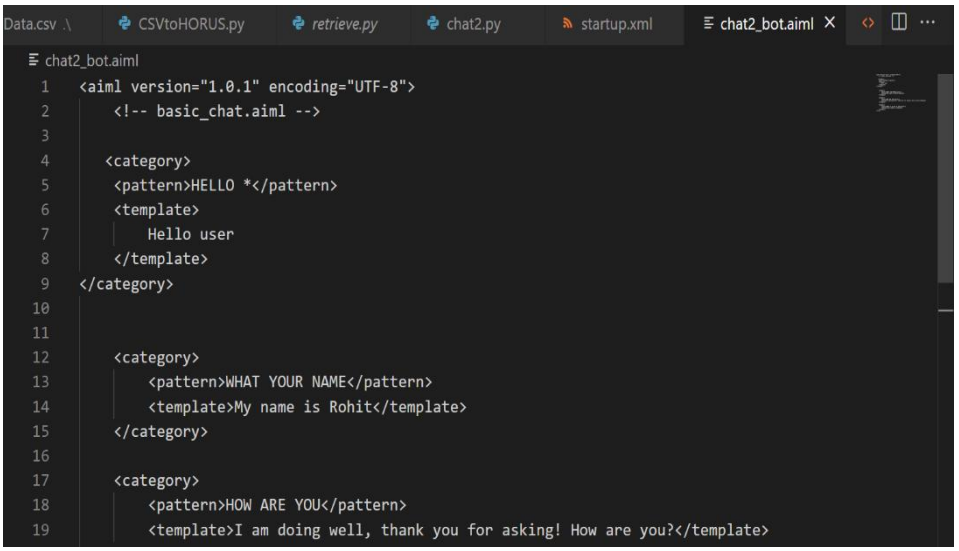
Step 2: write code for startup.xml

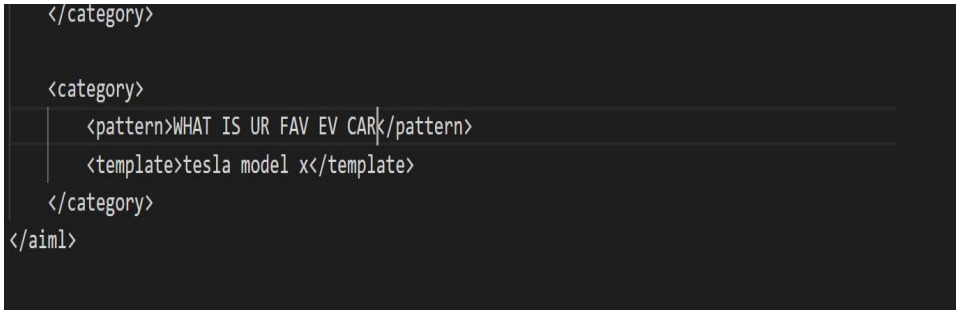
Code:



Step 3: write code for chat2_bot.aiml

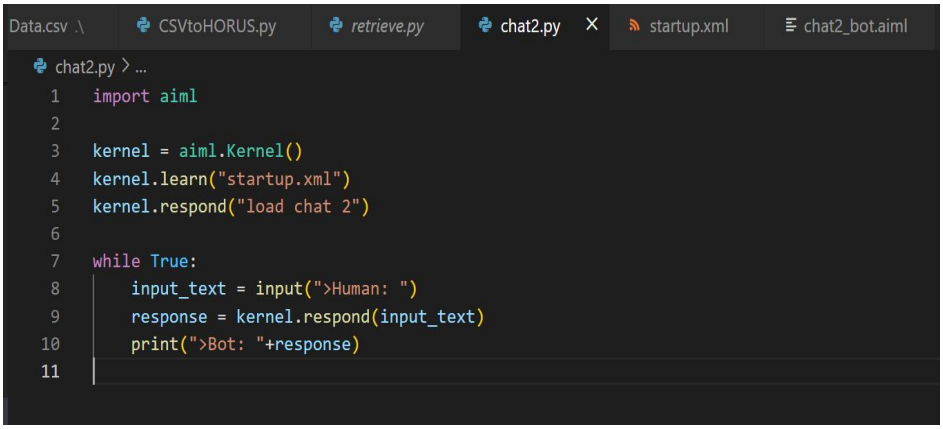
Code:





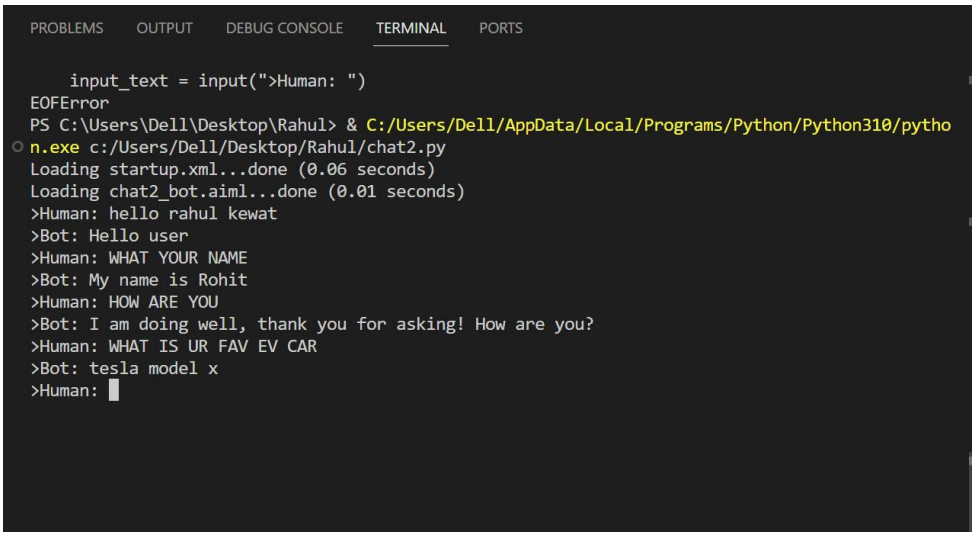
Step 4: write code for chat.py

Code:



Step 5

Output:



Practical: 3

Aim: Implement Bayes Theorem using Python

Writeup:

Description:

Bayes' Theorem states the following for any two events H and E:

Bayes' Theorem is stated as

P(H/E) = P(E/H)*P(H)/P(E)

- P(H|E): The probability of event H, given event E has occurred.
- P(E|H): The probability of event E, given event H has occurred.
- P(H): The probability of event H.
- P(E): The probability of event E.

For example, suppose the probability of the weather being cloudy is 40%.

Also suppose the probability of rain on a given day is 43%.

Also suppose the probability of clouds on a rainy day is 85%.

If it's cloudy outside on a given day, what is the probability that it will rain that day?

Solution:

- P(H)=P(cloudy) = 0.40
- P(E/H)=P(cloudy | rain) = 0.85
- P(~H) = 1-P(h)
  1-0.40
  P(~H) = 0.6

- P(E/~H) = 1-P(E/H)
  1-0.85
  P(E/~H) = 0.15

- P(E) = P(E/H)*P(H)+P(E/~H)*P(~H)
  0.85*0.40+0.15*0.6

  P(E) = P(rain) = 0.43

- P(H/E) = P(rain/cloud) ?
- Thus we can calculate:

  P(H/E) = P(E/H)*P(H)/P(E)

  0.34/0.43
  0.79
  P(H/E) = P(rain/cloud) is : 0.79

Example: Bayes Theorem in python

Suppose we know the following probabilities:

- P(H)=P(cloudy) = 0.40
- P(E/H)=P(cloudy | rain) = 0.85
- P(E) = P(rain) = 0.43

To Calculate P(H/E) or P(rain/cloud) we use following code:

Code:

```
      def bayes_theorem(p_h,p_e_given_h, p_e_given_not_h):
    not_h=1-p_h
    # p_e is prob of rain
    p_e=p_e_given_h*p_h+p_e_given_not_h*not_h
    # p(h/e) is prob of rain/cloud
    p_h_given_e=(p_e_given_h*p_h)/p_e
    return p_h_given_e
# p(h)
p_h = float(input("Enter prob of cloudiness given humidity : "))
# p(e/h)
p_e_given_h=float(input("Enter prob of cloudiness or raininess given humidity :
"))
# p(e/~h)
p_e_given_not_h=float(input("Enter prob of cloud/rain not given humidity : "))
result=bayes_theorem(p_h, p_e_given_h, p_e_given_not_h)
print("The prob of rain/cloud that is P(H/E)=", round(result,2))
```

Output:

```
PS C:\Users\Dell\Desktop\Rahul> & C:/Users/Dell/AppData/Local/Programs/Python/Python310/pytho
n.exe "c:/Users/Dell/Desktop/Rahul/bayes theorem.py"
Enter prob of cloudiness given humidity : 0.40
Enter prob of cloudiness or raininess given humidity : 0.85
Enter prob of cloud/rain not given humidity : 0.15
The prob of rain/cloud that is P(H/E)= 0.79
PS C:\Users\Dell\Desktop\Rahul>
```

Practical: 4

Aim: Implement Conditional Probability and joint probability using Python

Writeup:

4A: Joint Probability

Description:

What is Joint Probability?

The probability of two (or more) events is called the joint probability. The joint probability of two or more random variables is referred to as the joint probability distribution.The joint probability for events A and B is calculated as the probability of event A given event B multiplied by the probability of event B.

This can be stated formally as follows:

P(A and B) = P(A given B) * P(B)

The calculation of the joint probability is sometimes called the fundamental rule of probability or the "product rule" of probability or the "chain rule" of probability

P(A and B) = P(A given B) * P(B) = P(B given A) * P(A)

Code:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

sns.set()
data=pd.read_csv('bloodpressure.csv',header=None,names=['patient','gender','ag
egrp',
                                                        'bp_before','bp_after']
)
data['patient']=pd.to_numeric(data['patient'],errors='coerce')
data['gender']=pd.to_numeric(data['gender'],errors='coerce')
data['agegrp']=pd.to_numeric(data['agegrp'],errors='coerce')
data['bp_before']=pd.to_numeric(data['bp_before'],errors='coerce')
data['bp_after']=pd.to_numeric(data['bp_after'],errors='coerce')

sns.jointplot(data=data,patient='patient',gender='gender',agegrp='agegrp',
              bpbefore='bp_before',bpafter='bp_after',
              kind='kde').plot_joint(sns.scatterplot)
plt.show()
```

Output:

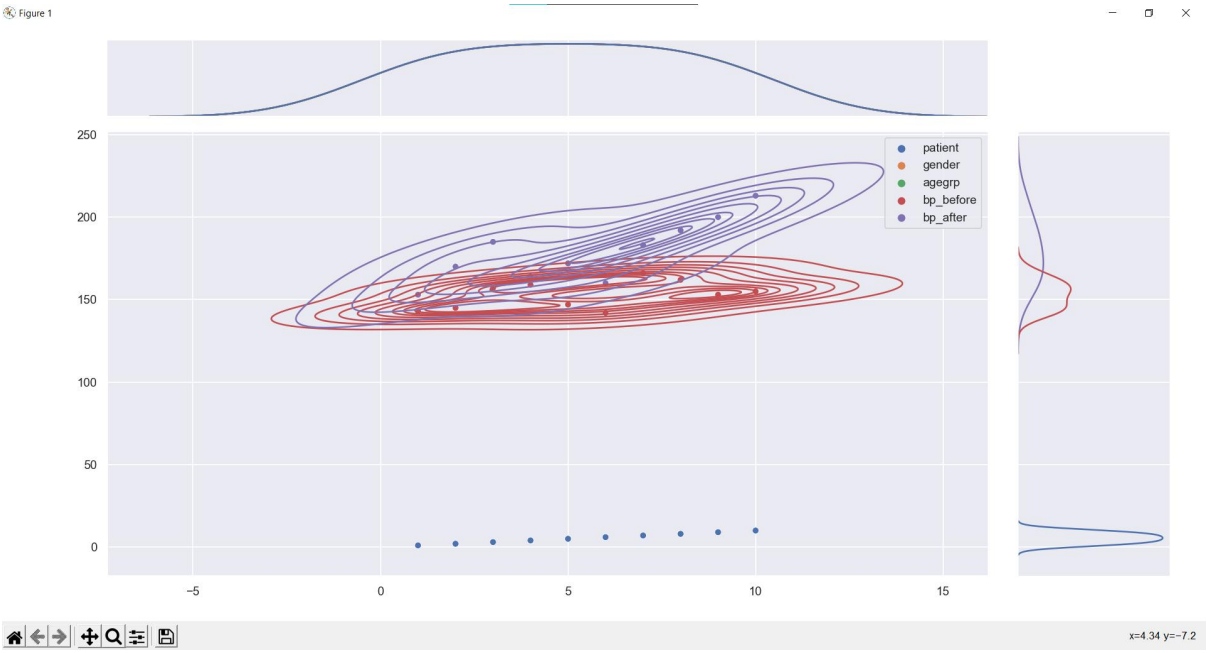


Figure 1
250
200
150
100
50
0
−5
0
5
10
15
patient
gender
agegrp
bp_before
bp_after
x=4.34 y=−7.2

4B: Conditional Probability

Description:

The probability of one event given the occurrence of another event is called the conditional probability. The conditional probability of one to one or more random variables is referred to as the conditional probability distribution.

For example, the conditional probability of event A given event B is written formally as:

- **P(A given B)**

The "given" is denoted using the pipe "|" operator; for example:

- **P(A | B)**

The conditional probability for events A given event B is calculated as follows:

- **P(A given B) = P(A and B) / P(B)**

Code:

```
import numpy as np
import pandas as pd
df=pd.read_csv('student-mat.csv')
print(df.head(3))
num_rows=len(df)
df['grade_A'] = np.where(df['G3']*5 >= 80, 1, 0)
df['high_absenses'] = np.where(df['absenses'] >= 10, 1, 0)
df['Count'] = 1
df=df[['grade_A','high_absenses','Count']]
print(df.head())
pivot_table=pd.pivot_table(df,values='Count',index=['grade_A'],columns=['high_
absenses'],aggfunc=np.size,fill_value=0)
print(pivot_table)
```

Output:

```
PS & C:/Users/Dell/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Dell/OneDrive
/Documents/Rahul Kewat/python file/conditional.py"
  school sex  age address famsize Pstatus  Medu  ...  Dalc Walc health absenses G1  G2  G3
0     GP   F   18       U     GT3       A     4  ...     1    1      3        6  5   6   6
1     GP   F   17       U     GT3       T     1  ...     1    1      3        4  5   5   6

[2 rows x 33 columns]
   grade_A  high_absenses  Count
0        0              0      1
1        0              0      1
high_absenses  0
grade_A
0              2
PS C:\Users\Dell\OneDrive\Documents\Rahul Kewat\python file>
```

Practical: 5

Aim: A program to implement Rule Based System.

Writeup:

Description:

What is Rule Based System?

A rule-based system is a system that applies human-made rules to store, sort and manipulate data. In doing so, it mimics human intelligence.

To work, rule-based systems require a set of facts or source of data, and a set of rules for manipulating that data. These rules are sometimes referred to as 'If statements' as they tend to follow the line of 'IF X happens THEN do Y'.

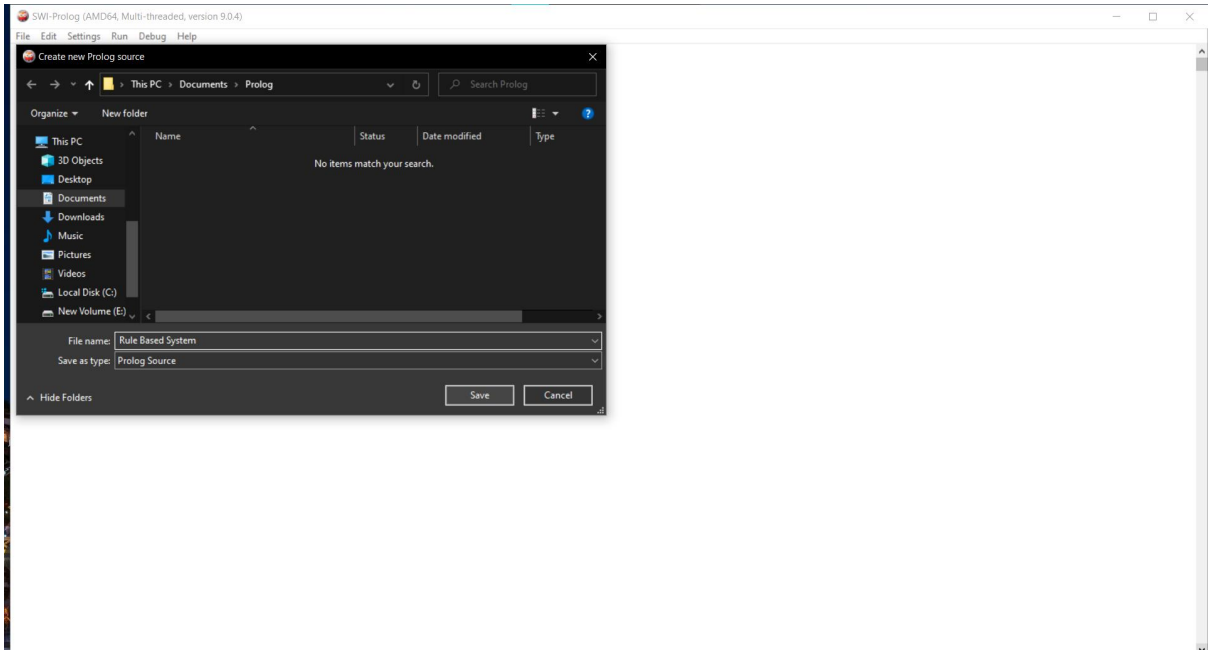
Automation software like Think Automation is a good example. It automates processes by breaking them down into steps.

- First comes the data or new business event
- Then comes the analysis: the part where the system conditonally processes the data against its rules
- Then comes any subsequent automated follow-up actions

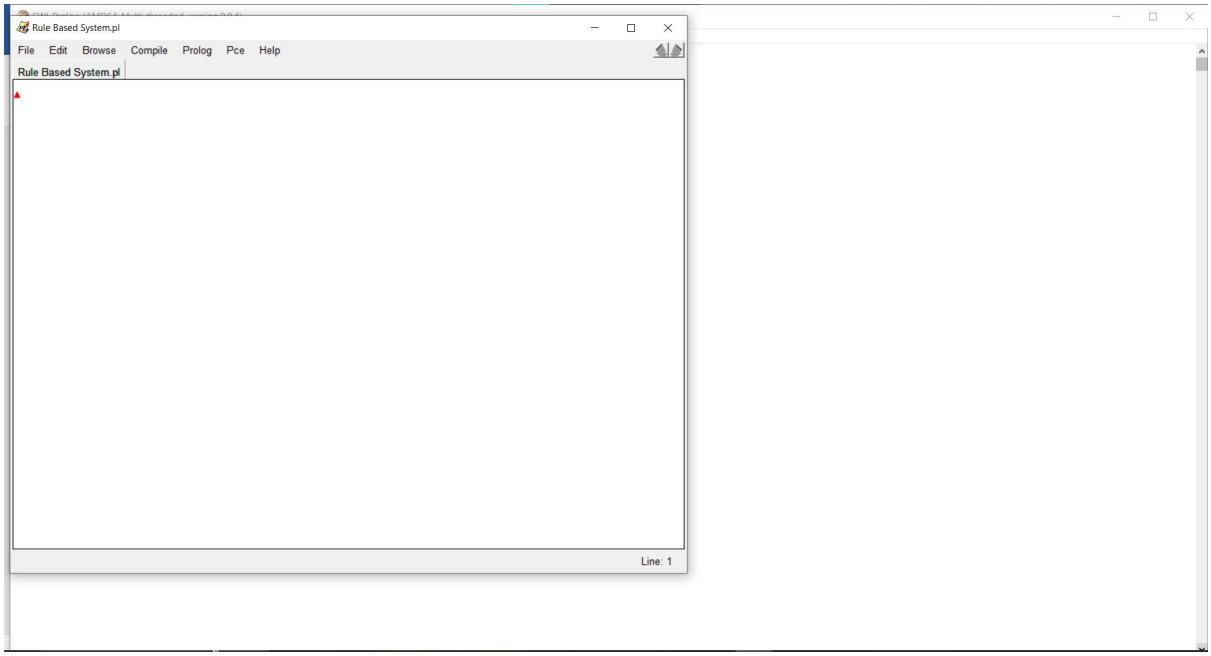
Step 1: Open Prolog Window

Step 2: Click on File and Select **New..**

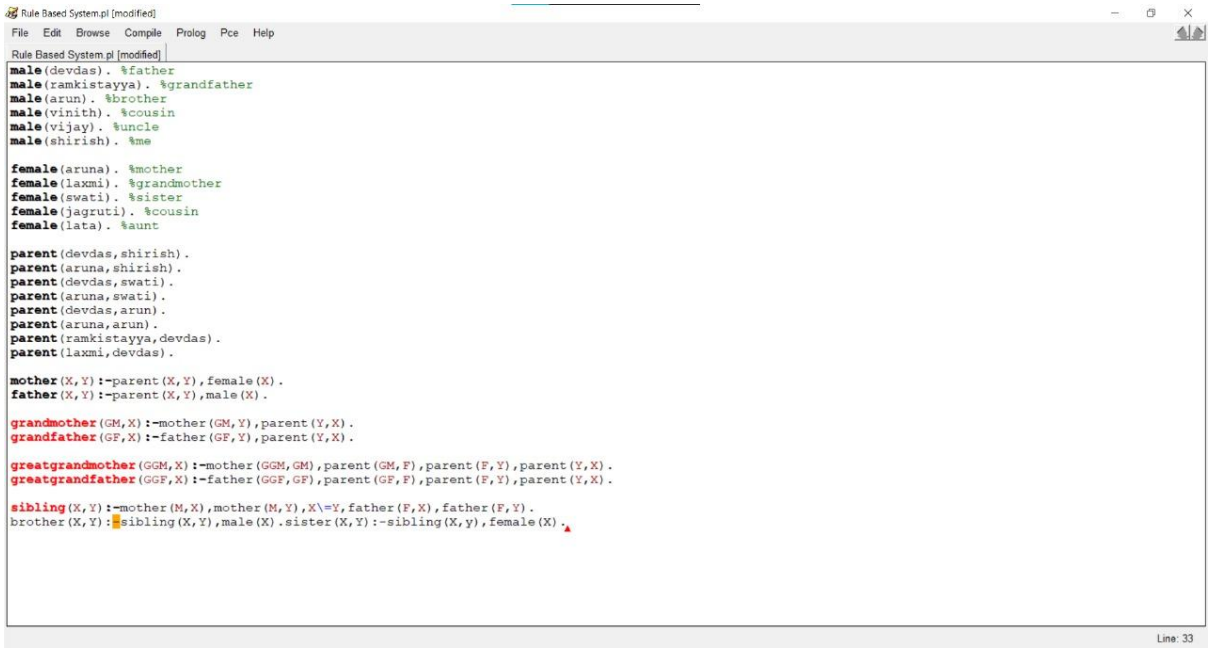
Step 3: Create new Prolog source **(Here it is Rule Based System)** And **Click on Save**



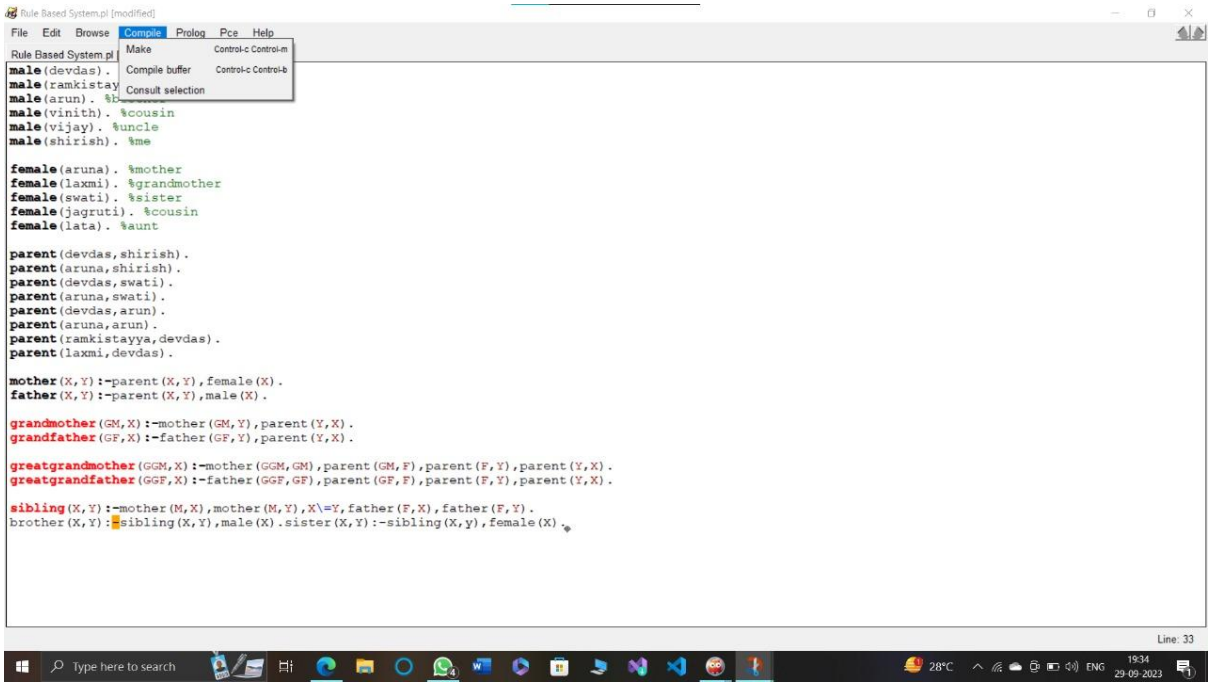
Step 4: New Window Open



Step 5: Type Code:



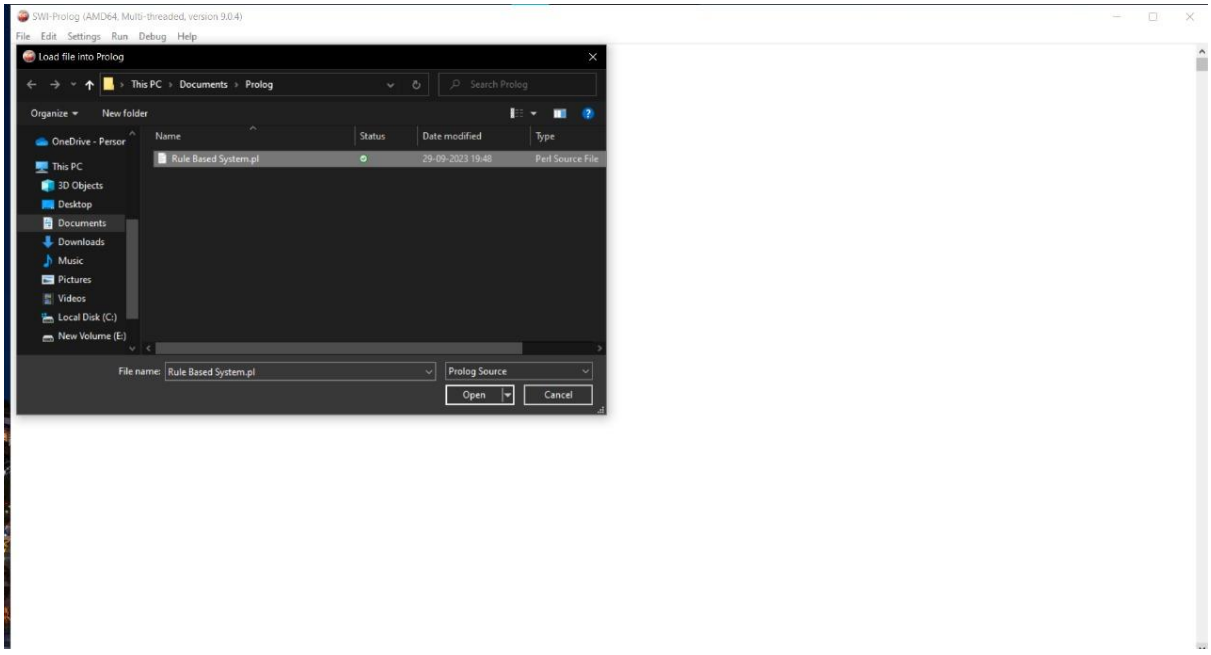
Step 6: Click on Compile and **Select Compile buffer**



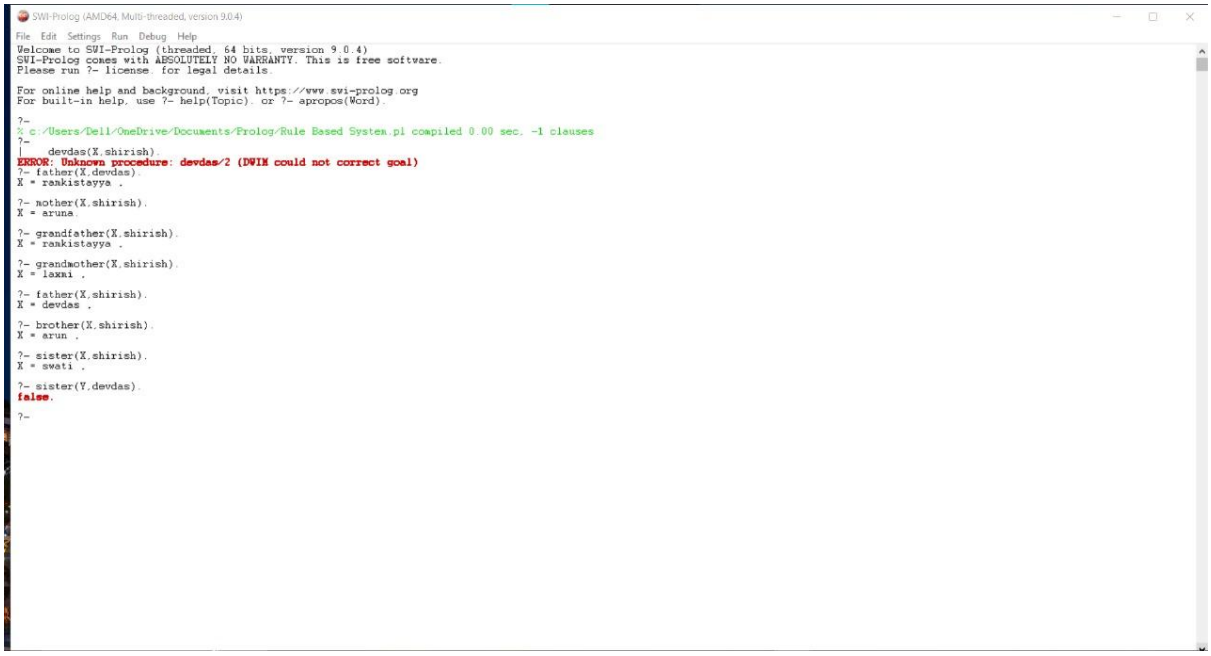
Step 7: Go to Console Window

Step 8: Click on File and Select **Consult**

Step 9: Select **Rule Based System.pl** and **Click on Open**



Step 10: Output



Practical: 6

Aim: Design a Fuzzy based application using Python

Writeup:

Description:

What is Fuzzy based application?

Fuzzy sets were introduced by Lotfi Zadeh (1921–2017) in 1965.

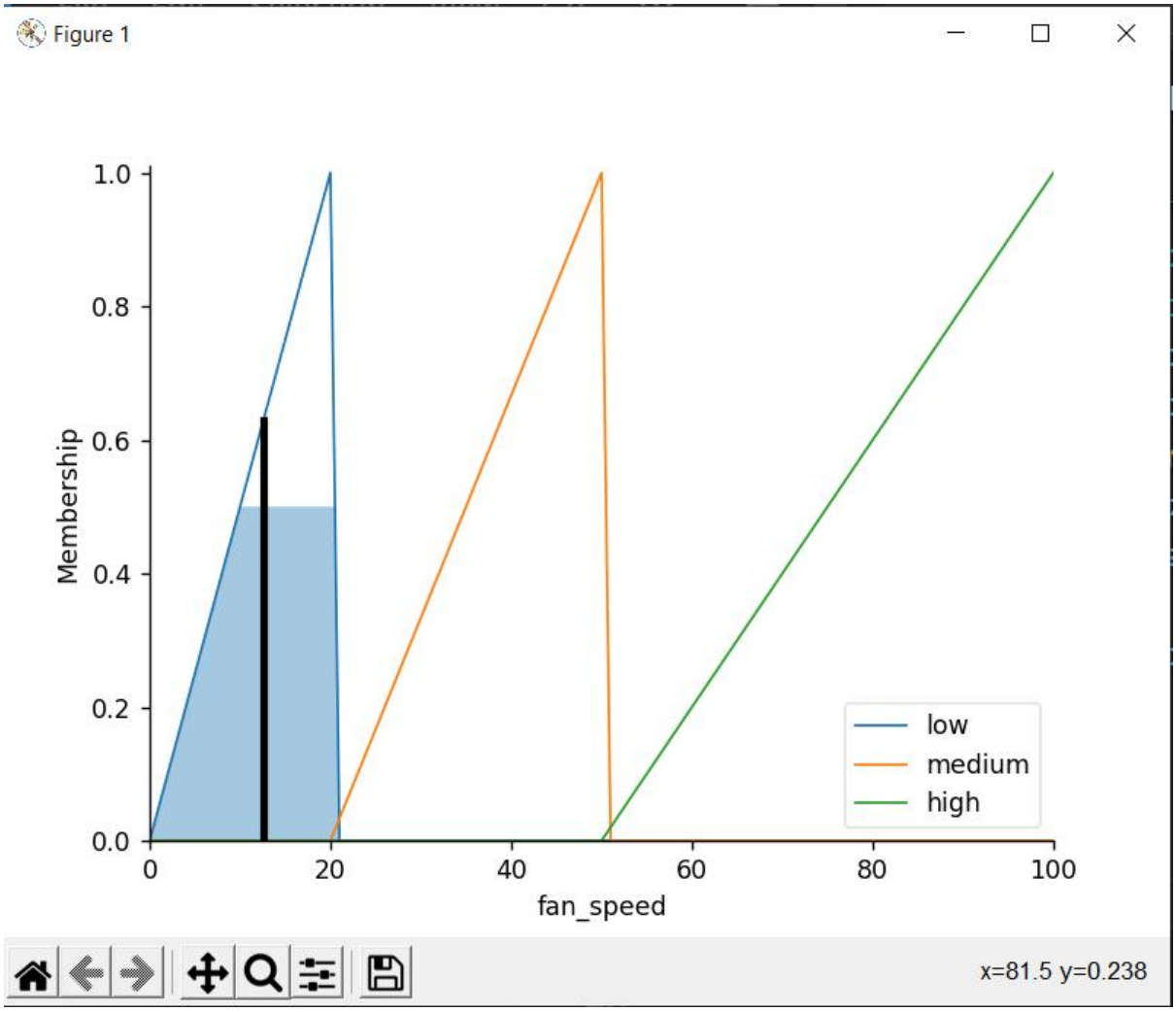
Unlike crisp sets, a fuzzy set allows partial belonging to a set, that is defined by a degree of membership, denoted by μ, that can take any value from 0 (element does not belong at all in the set) to 1 (element belongs fully to the set).

It is evident that if we remove all the values of belonging except from 0 and 1, the fuzzy set will collapse to a crisp set that was described in the previous section.

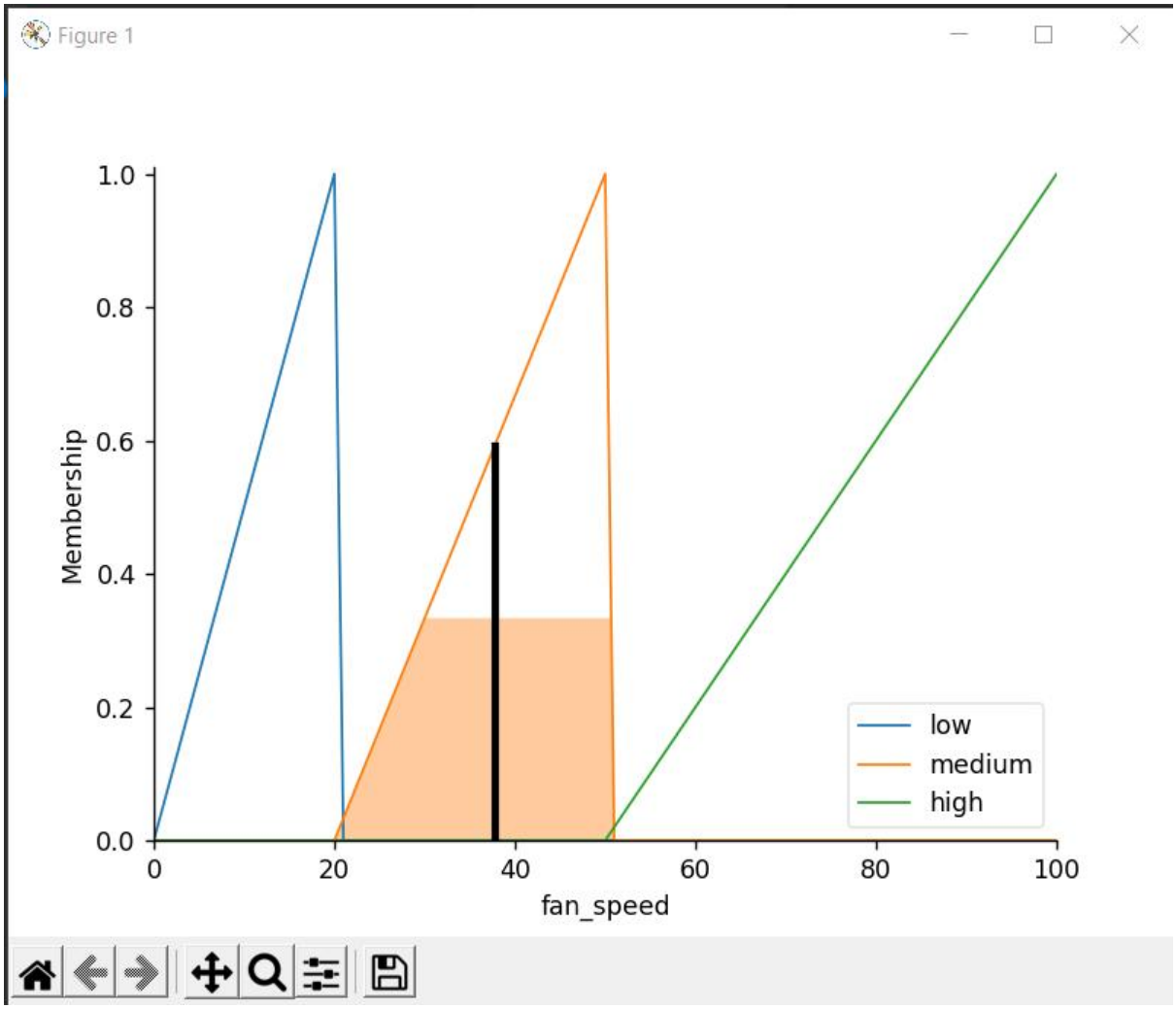
Code:

```
import numpy as np
import skfuzzy as fuzzy
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt

temperature = ctrl.Antecedent(np.arange(0, 101, 1), 'temperature')
humidity = ctrl.Antecedent(np.arange(0, 101, 1), 'humidity')
fan_speed = ctrl.Consequent(np.arange(0, 101, 1), 'fan_speed')

temperature['cold'] = fuzzy.trimf(temperature.universe, [0, 20,20])
temperature['medium'] = fuzzy.trimf(temperature.universe, [20,50,50])
temperature['hot'] = fuzzy.trimf(temperature.universe, [50,100,100])

humidity['low'] = fuzzy.trimf(humidity.universe, [0, 20,20])
humidity['medium'] = fuzzy.trimf(humidity.universe, [20,50,50])
humidity['high'] = fuzzy.trimf(humidity.universe, [50,100,100])

fan_speed['low'] = fuzzy.trimf(fan_speed.universe, [0, 20,20])
fan_speed['medium'] = fuzzy.trimf(fan_speed.universe, [20,50,50])
fan_speed['high'] = fuzzy.trimf(fan_speed.universe, [50,100,100])

rule1 = ctrl.Rule(temperature['cold'] & humidity['low'], fan_speed['low'])
rule2 = ctrl.Rule(temperature['medium'] & humidity['medium'],
fan_speed['medium'])
rule3 = ctrl.Rule(temperature['hot'] & humidity['high'], fan_speed['high'])
fan_speed_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
fan_speed_simulation = ctrl.ControlSystemSimulation(fan_speed_ctrl)
fan_speed_simulation.input['temperature'] = int(input('Enter temperature:'))
fan_speed_simulation.input['humidity'] = int(input('Enter humidity:'))
fan_speed_simulation.compute()
output_speed = fan_speed_simulation.output['fan_speed']
fan_speed.view(sim=fan_speed_simulation)
plt.show()
print(f"Fan Speed: {output_speed}")
```

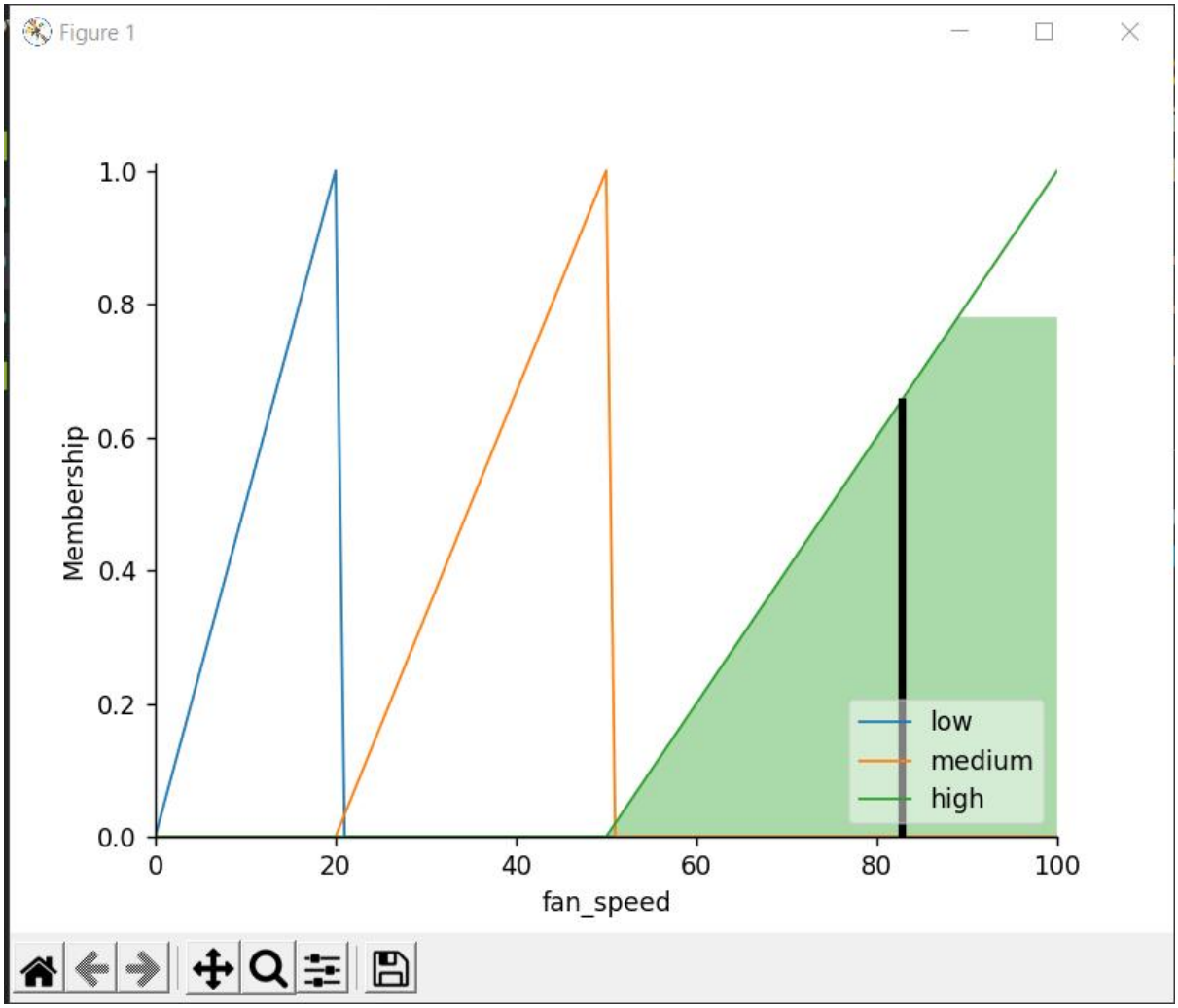
Output:




```
ograms/Python/Python310/python.exe "c:/Users/Dell/OneDrive/Documents/Rahul Kewat/python file/
Fuzzy Based Application.py"
Enter temperature:15
Enter humidity:10
Fan Speed: 12.611111111111109
```




```
ograms/Python/Python310/python.exe "c:/Users/Dell/OneDrive/Documents/Rahul Kewat/python file/
Fuzzy Based Application.py"
Enter temperature:35
Enter humidity:30
Fan Speed: 37.75555555555557
```




```
ograms/Python/Python310/python.exe "c:/Users/Dell/OneDrive/Documents/Rahul Kewat/python file/
Fuzzy Based Application.py"
Enter temperature:95
Enter humidity:89
Fan Speed: 82.67213114754095
```

Practical: 7

Aim: Write an application to simulate supervised and un-supervised learning model.

Writeup:

Description:

What is supervised learning?

Supervised learning is a type of machine learning where the algorithm is trained using labeled data**.** This means that the data used to train the algorithm has already been tagged with the correct answer.

The algorithm then uses this labeled data to learn how to classify new, unlabeled data.

Supervised learning is used in two types of problems: **classification** and **regression.**

In a **classification problem**, the output variable is a category, such as “Red” or “Blue”.

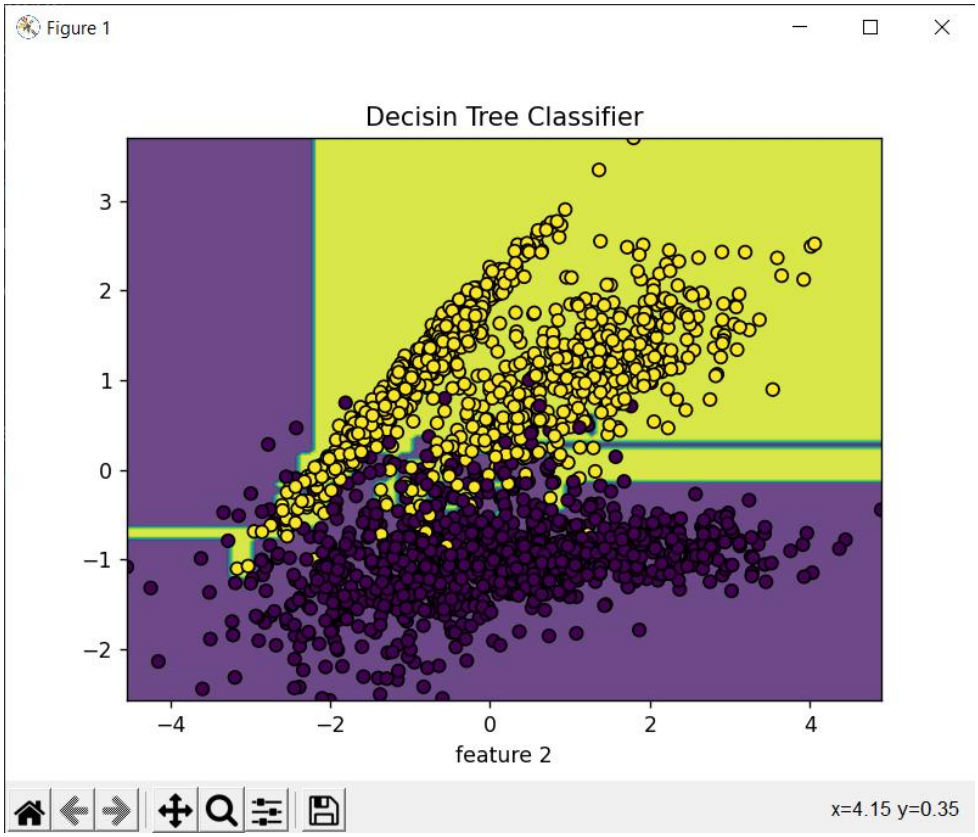
In a **regression problem**, the output variable is a real value, such as “dollars” or “weight”

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Create a simulated dataset for classification
X, y = make_classification(
    n_samples=2000,
    n_features=2,      # Total number of features
    n_informative=2,  # Number of informative features
    n_redundant=0,    # Number of redundant features
    random_state=50
)

# Split the dataset into a training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create and train a supervised learning model (Logistic Regression)
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)

# Make predictions using the trained classifier
y_pred = classifier.predict(X_test)

# Calculate and display the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Supervised Learning Accuracy: {accuracy:.2f}")

classification_rep = classification_report(y_test,y_pred)
print("Classification Report\n", classification_rep)

xx, yy = np.meshgrid(np.linspace(X[:,0].min(),X[:,0].max(),100),
```

```
                    np.linspace(X[:,1].min(),X[:,1].max(),100))
Z = classifier.predict(np.c_[xx.ravel(),yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx,yy,Z,alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c = y, marker='o',edgecolors='k')
plt.xlabel('feature 1')
plt.xlabel('feature 2')
plt.title('Decisin Tree Classifier')
plt.show()
```

Output:



```
PS C:\Users\Dell\OneDrive\Documents\Rahul Kewat\python file> & C:/Users/Dell/AppData/Local/Pr
ograms/Python/Python310/python.exe "c:/Users/Dell/OneDrive/Documents/Rahul Kewat/python file/
Supervised.py"
Supervised Learning Accuracy: 0.95
Classification Report
              precision    recall  f1-score   support

           0       0.95      0.94      0.95       186
           1       0.95      0.96      0.95       214

    accuracy                           0.95       400
   macro avg       0.95      0.95      0.95       400
weighted avg       0.95      0.95      0.95       400
```

Description:

What is unsupervised learning?

Unsupervised learning is the training of machine using information that is neither classified nor labelled and allowing the algorithm to act on that information without guidance. Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data. Unsupervised learning classified into two categories of algorithms:

- **Clustering:** A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behaviour.
- **Association:** An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Types of Unsupervised Learning:

Clustering:

- Exclusive (partitioning)
- Agglomerative
- Overlapping
- Probabilistic

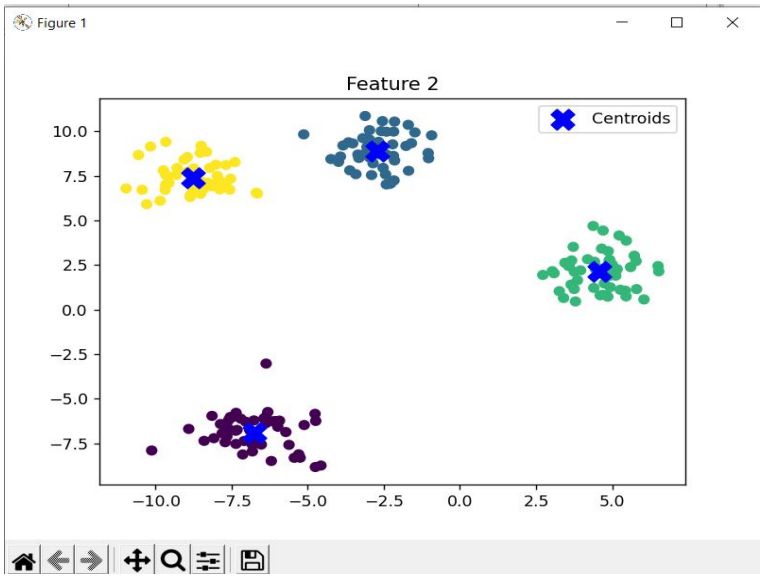
Clustering Types:

- Hierarchical clustering
- K-means clustering
- Principal Component Analysis
- Singular Value Decomposition
- Independent Component Analysis

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

n_samples=200
n_features=2
n_clusters=4

X, _ = make_blobs(n_samples=n_samples, n_features=n_features,
centers=n_clusters, random_state=42)

n_init_value=10
kmeans=KMeans(n_clusters=n_clusters, n_init=n_init_value)
kmeans.fit(X)

cluster_labels=kmeans.predict(X)
centroids = kmeans.cluster_centers_


plt.scatter(X[:, 0], X[:, 1], c = cluster_labels, cmap='viridis')
plt.scatter(centroids[:, 0], centroids[:, 1], c='blue', marker='X', s=200,
label='Centroids')
plt.legend()
plt.title('K-Means Clustering')
plt.title('Feature 1')
plt.title('Feature 2')
plt.show()
```

Output:



Practical: 8

Aim: Write an application to implement Clustering algorithm.

Writeup:

Description:

What is Clustering?

Clustering is a technique in artificial intelligence and machine learning that involves grouping similar data points together based on certain features or characteristics they share. The goal of clustering is to identify patterns or structures within a dataset, where data points within the same cluster are more similar to each other than to those in other clusters. Clustering is an unsupervised learning method, meaning that it does not require labeled data and does not involve making predictions; instead, it focuses on discovering inherent structures within the data.

Code:

Clustering.py

```python
from numpy import where
from sklearn.datasets import make_classification
from matplotlib import pyplot

X, y = make_classification(n_samples=1000, n_features=2, n_informative=2,
                           n_redundant=0, n_clusters_per_class=1, random_state=4)

for class_value in range(2):

    row_ix = where(y == class_value)

    pyplot.scatter(X[row_ix,0], X[row_ix, 1])

pyplot.show()
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Dell\OneDrive\Documents\Rahul Kewat\python file>
```

Output:

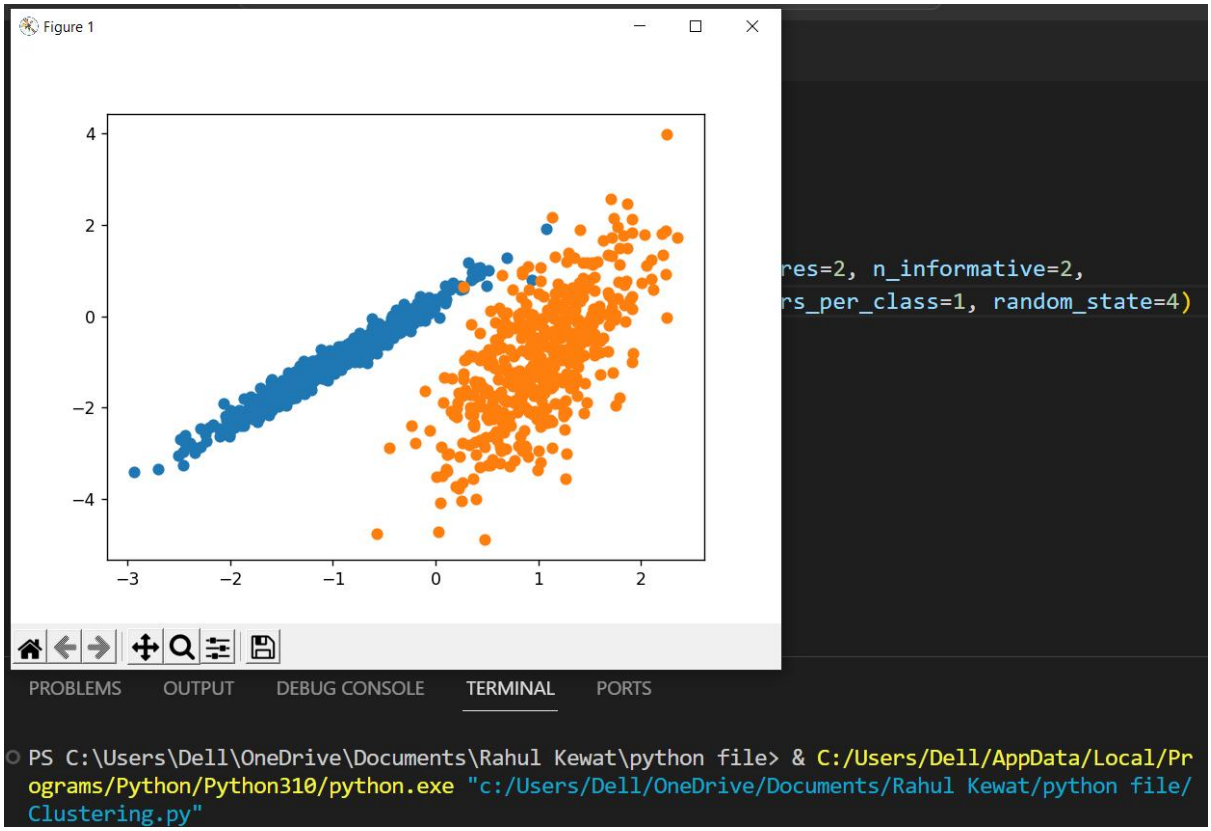


Figure 1
4
2
0
−2
−4
−3
−2
−1
0
1
2
res=2, n_informative=2,
rs_per_class=1, random_state=4)
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS C:\Users\Dell\OneDrive\Documents\Rahul Kewat\python file> & C:/Users/Dell/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Dell/OneDrive/Documents/Rahul Kewat/python file/Clustering.py"

Practical: 9

Aim: Write a Program to implement BFS algorithm.

Writeup:

Description:

What is Breadth-First Search?

The Breadth First Search (BFS) algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level.

Relation between BFS for Graph and Tree traversal:

Breadth-First Traversal (or Search) for a graph is similar to the Breadth-First Traversal of a tree. The only catch here is, that, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we divide the vertices into two categories:

- Visited and
- Not visited.

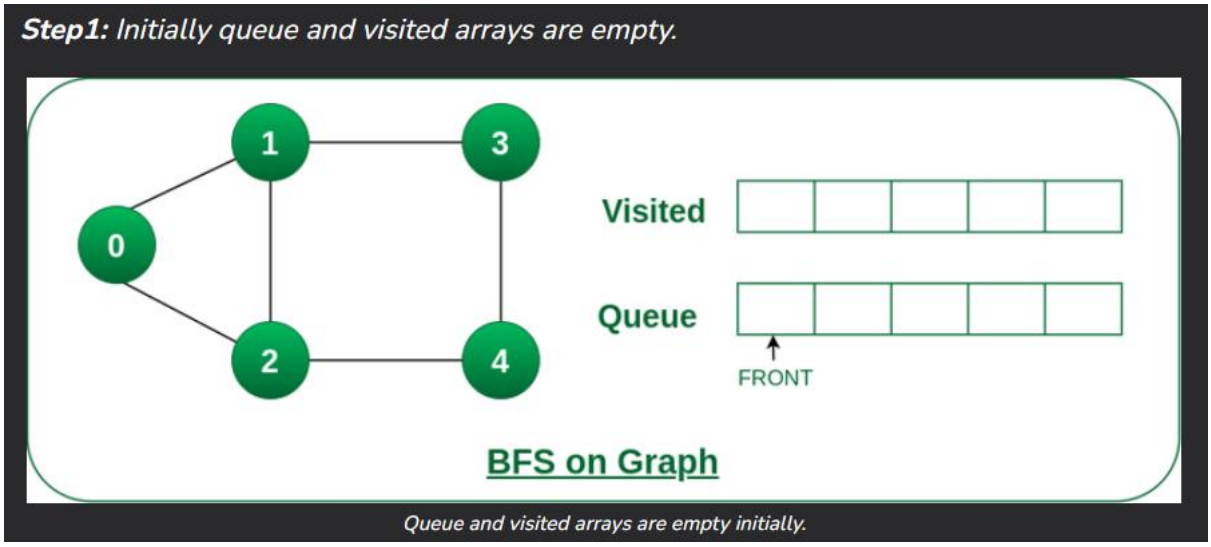
How does BFS work?

Starting from the root, all the nodes at a particular level are visited first and then the nodes of the next level are traversed till all the nodes are visited.

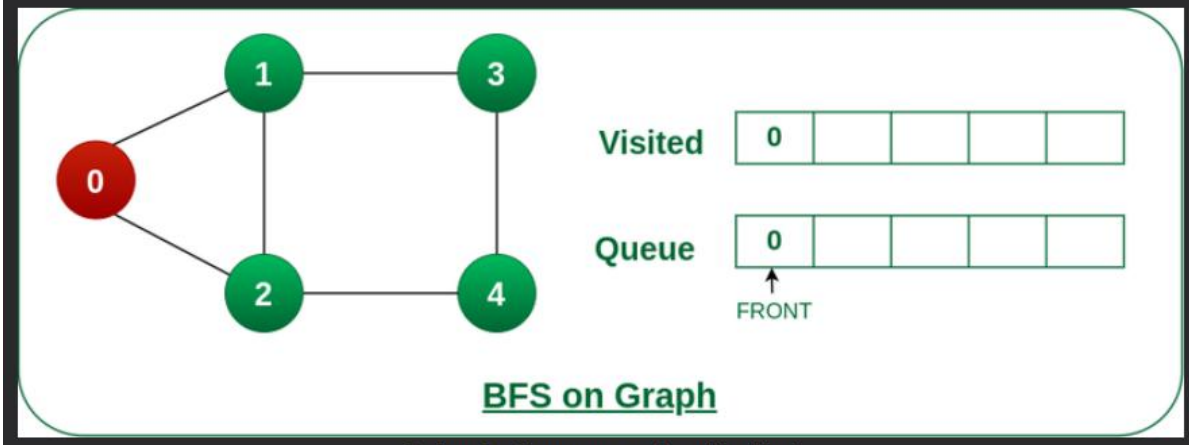
To do this a queue is used. All the adjacent unvisited nodes of the current level are pushed into the queue and the nodes of the current level are marked visited and popped from the queue.

Illustration:

Let us understand the working of the algorithm with the help of the following example.

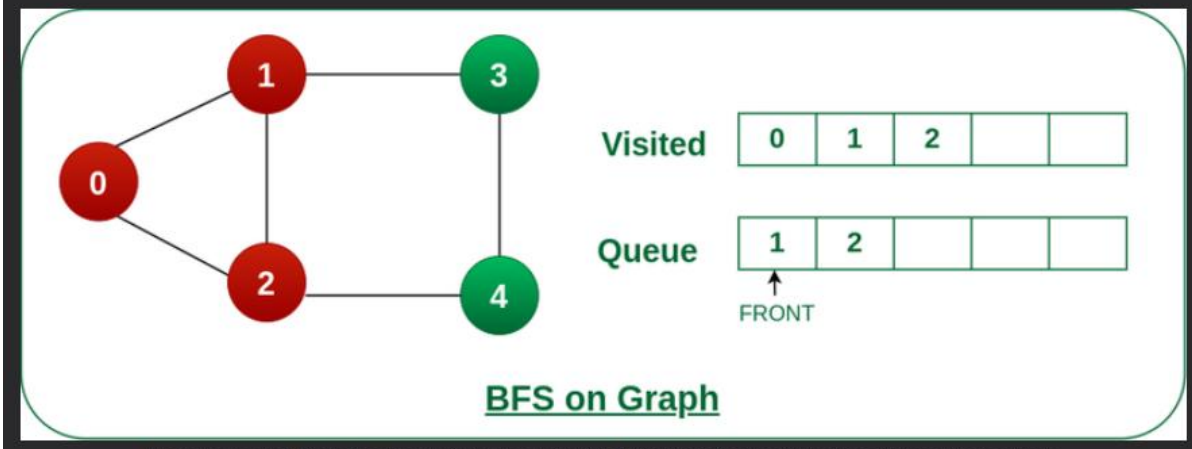


Step2: *Push node 0 into queue and mark it visited.*



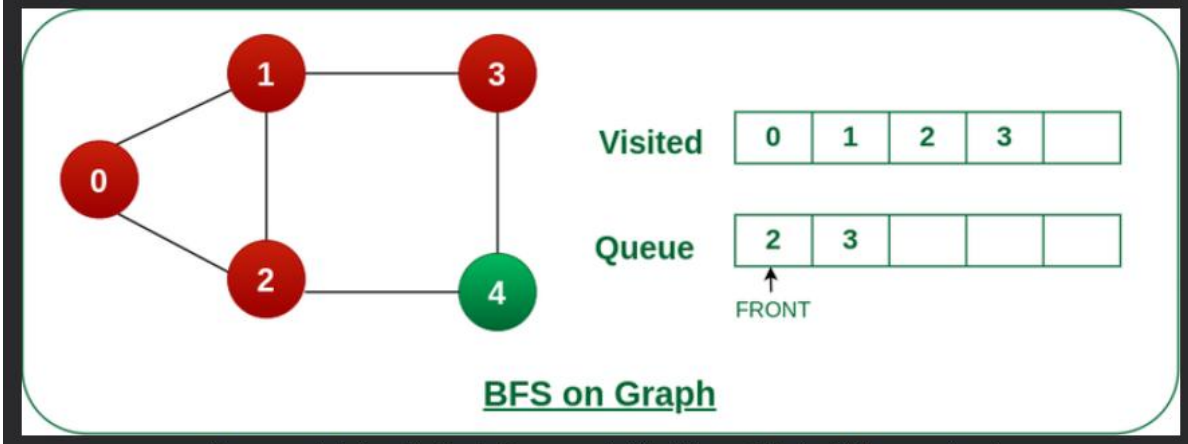
Push node 0 into queue and mark it visited.

Step 3: *Remove node 0 from the front of queue and visit the unvisited neighbours and push them into queue.*



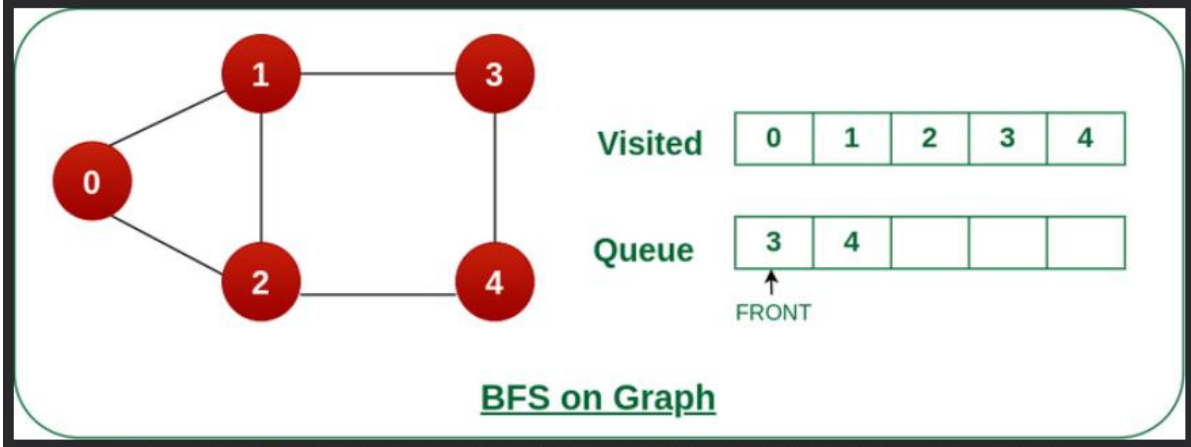
Remove node 0 from the front of queue and visited the unvisited neighbours and push into queue.

Step 4: *Remove node 1 from the front of queue and visit the unvisited neighbours and push them into queue.*



Remove node 1 from the front of queue and visited the unvisited neighbours and push

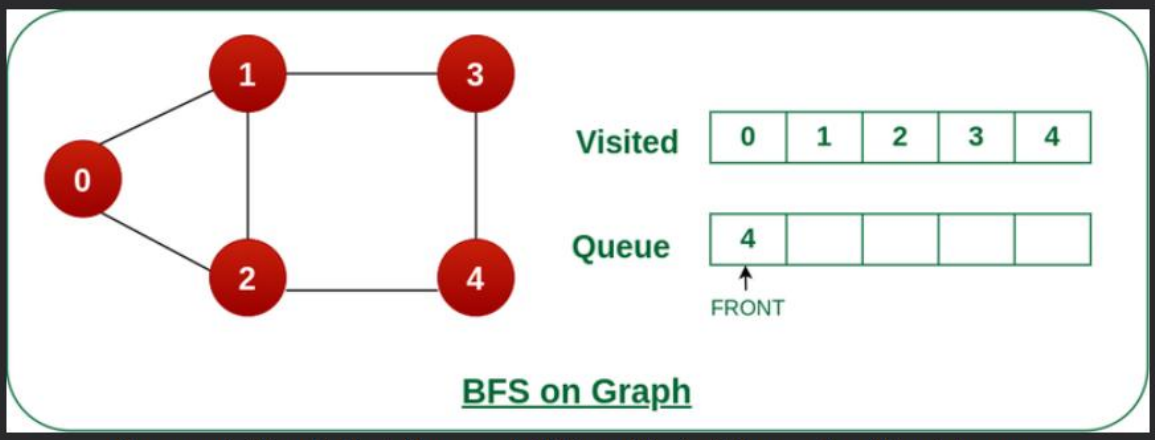
Step 5: *Remove node 2 from the front of queue and visit the unvisited neighbours and push them into queue.*



Remove node 2 from the front of queue and visit the unvisited neighbours and push them into queue.

Step 6: *Remove node 3 from the front of queue and visit the unvisited neighbours and push them into queue.*

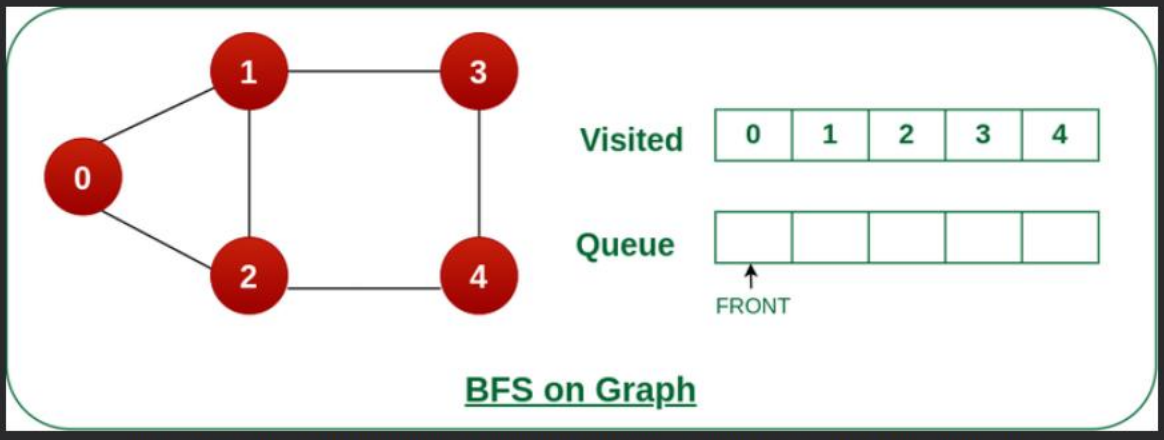
As we can see that every neighbours of node 3 is visited, so move to the next node that are in the front of the queue.



Remove node 3 from the front of queue and visit the unvisited neighbours and push them into queue.

Steps 7: *Remove node 4 from the front of queue and visit the unvisited neighbours and push them into queue.*

As we can see that every neighbours of node 4 are visited, so move to the next node that is in the front of the queue.



Remove node 4 from the front of queue and visit the unvisited neighbours and push them into queue.

Now, Queue becomes empty, So, terminate these process of iteration.

Code:

Breadth-First Search.py

```python
def bfs(graph, start):
    visited = set()
    queue = [start]
    visited.add(start)

    while queue:
        vertex = queue.pop(0)
        print(vertex, end=' ')
        for neighbor in graph[vertex]:
            if neighbor not in visited:
                queue.append(neighbor)
                visited.add(neighbor)
if __name__== '__main__':
    graph = {0: [1, 2], 1: [2], 2: [3], 3: [1, 2]}
    print("Following is the breadth-first traversal:")
    bfs(graph,0)
```

Output:

```
ograms/Python/Python310/python.exe "c:/Users/Dell/OneDrive/Documents/Rahul Kewat/python file/
Breadth-First Search.py"
Following is the breadth-first traversal:
0 1 2 3
```

Practical: 10

Aim: Write a Program to implement DFS algorithm.

Writeup:

Description:

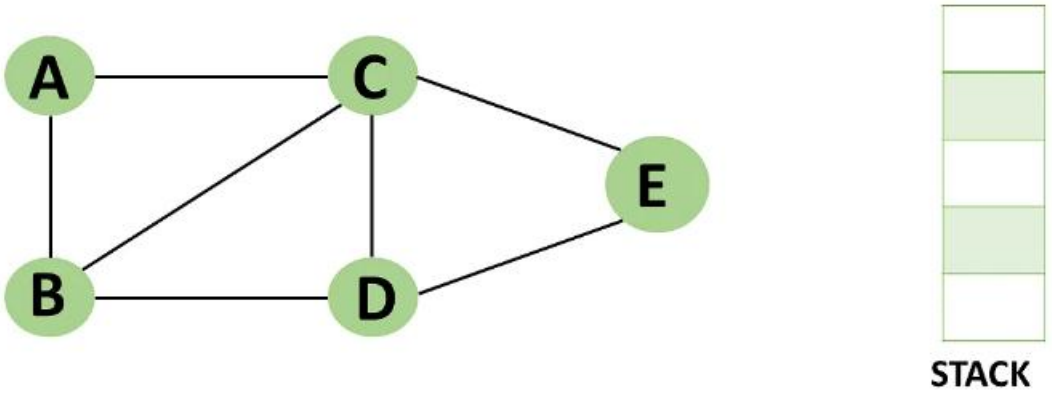
Depth-First Search (DFS) is a fundamental graph traversal algorithm used to explore and navigate through the nodes and edges of a graph or tree data structure. It starts at a designated "root" node and explores as far as possible along each branch before backtracking.

The basic idea behind DFS can be summarized as follows

1. Start at the root node.
2. Explore a neighboring node connected to the current node. If there are multiple neighboring nodes, choose one.
3. Continue the process recursively by moving to the neighboring node and repeating step 2.
4. If there are no unvisited neighboring nodes, backtrack to the previous node and explore other unvisited branches.
5. Repeat steps 2-4 until all nodes have been visited or until you've found the target node if you are searching for a specific node in the graph.

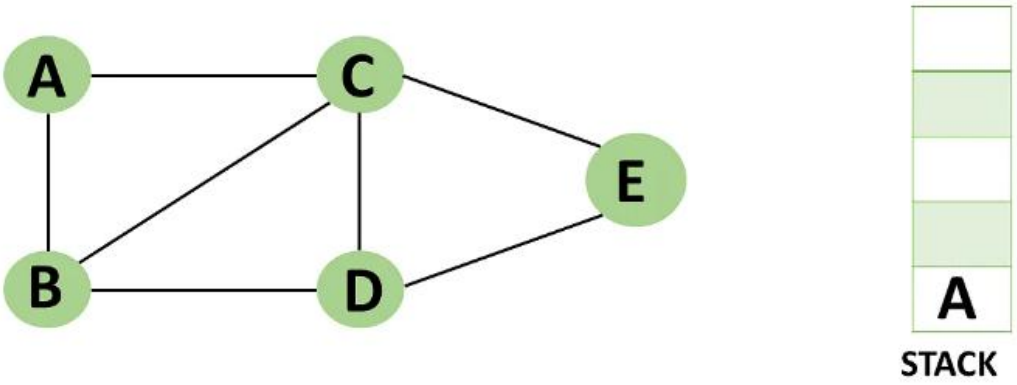
DFS is often implemented using a stack (or the call stack in a recursive implementation) to keep track of nodes to be explored. When using recursion, each recursive call corresponds to a step deeper into the graph, and the backtracking occurs automatically when a branch is fully explored.

Consider the following graph as an example of how to use the dfs algorithm.



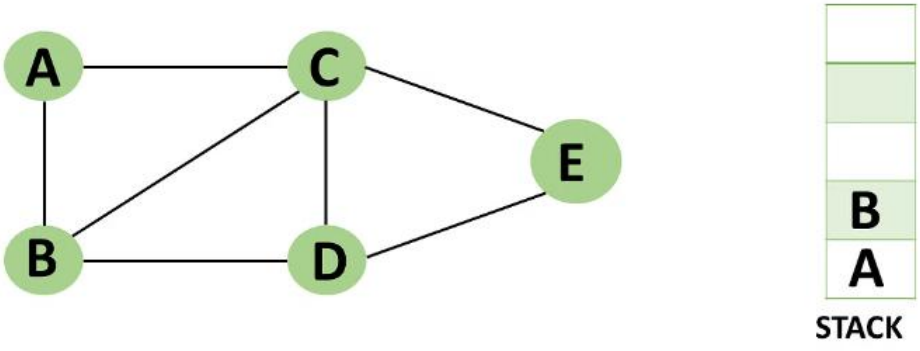
Step 1: Mark vertex A as a visited source node by selecting it as a source node.

- You should push vertex A to the top of the stack.



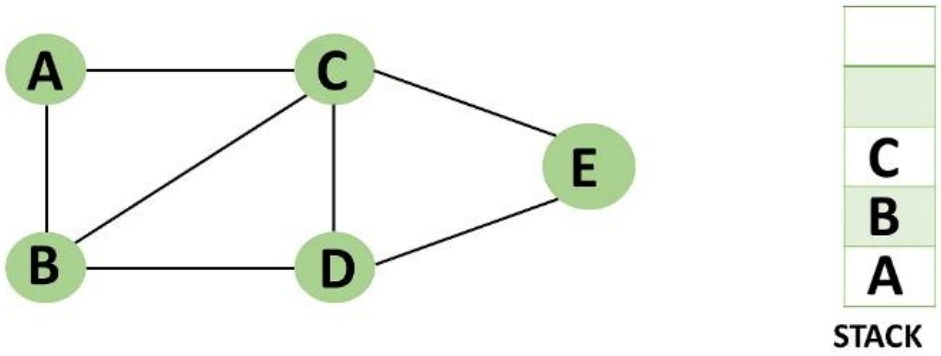
Step 2: Any nearby unvisited vertex of vertex A, say B, should be visited.

- You should push vertex B to the top of the stack.



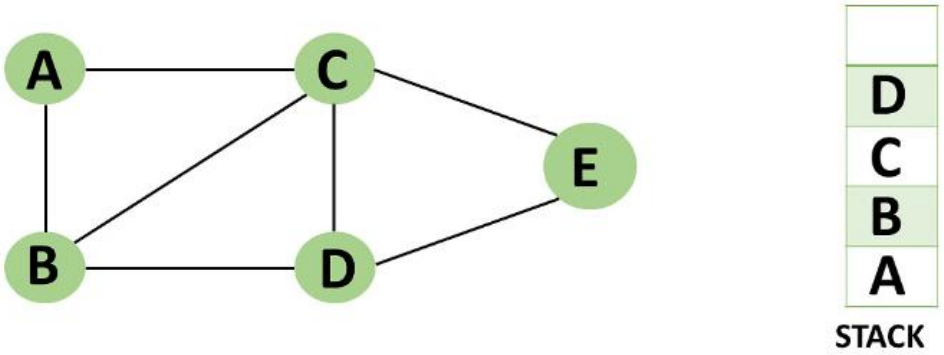
Step 3: From vertex C and D, visit any adjacent unvisited vertices of vertex B. Imagine you have chosen vertex C, and you want to make C a visited vertex.

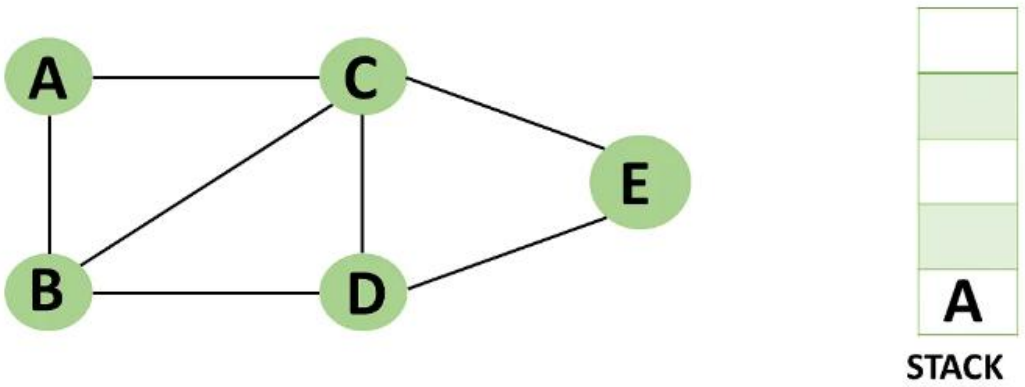
- Vertex C is pushed to the top of the stack.



Step 4: You can visit any nearby unvisited vertices of vertex C, you need to select vertex D and designate it as a visited vertex.

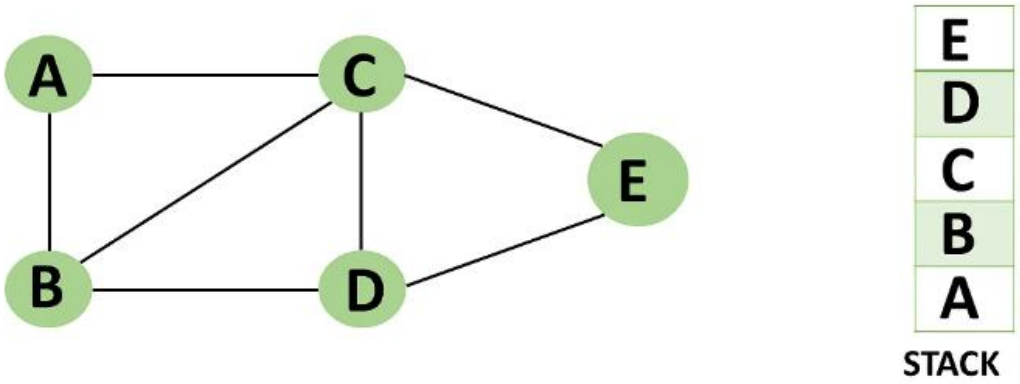
- Vertex D is pushed to the top of the stack.



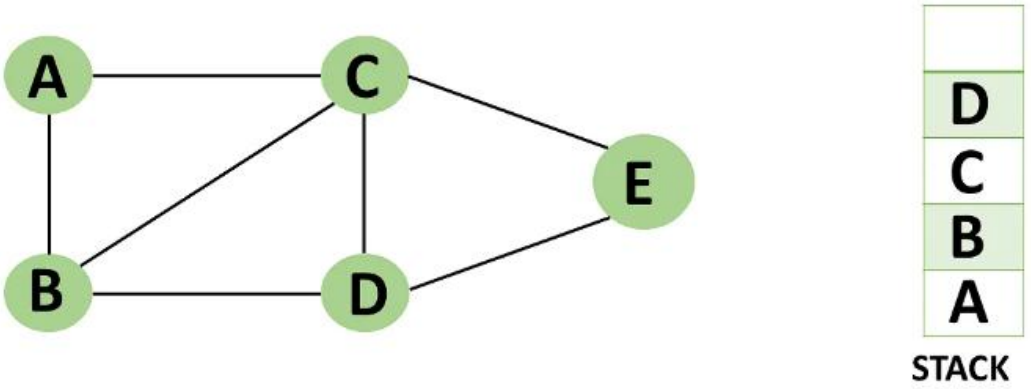


Step 5: Vertex E is the lone unvisited adjacent vertex of vertex D, thus marking it as visited.

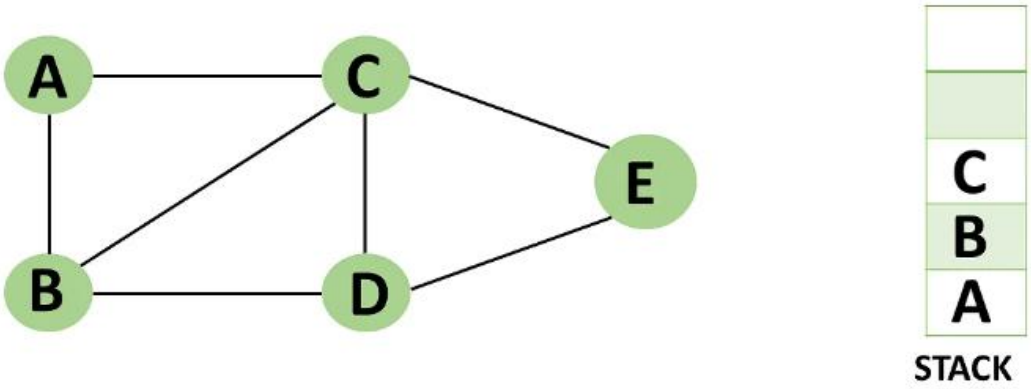
- Vertex E should be pushed to the top of the stack.



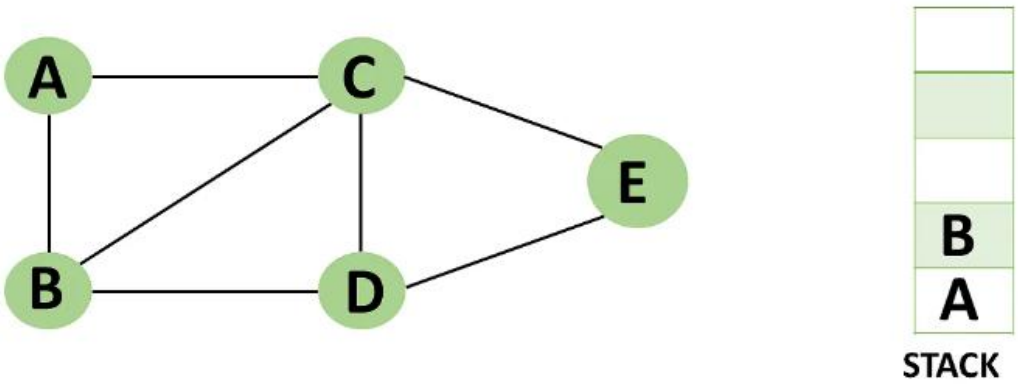
Step 6: Vertex E's nearby vertices, namely vertex C and D have been visited, pop vertex E from the stack.



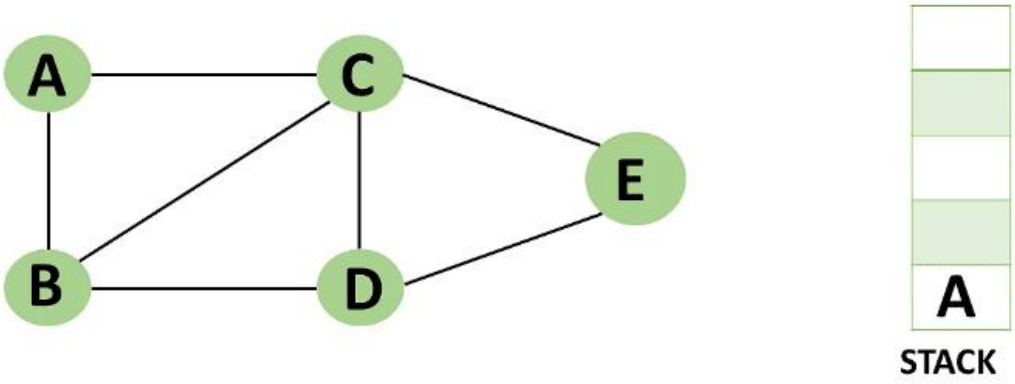
Step 7: Now that all of vertex D's nearby vertices, namely vertex B and C, have been visited, pop vertex D from the stack.



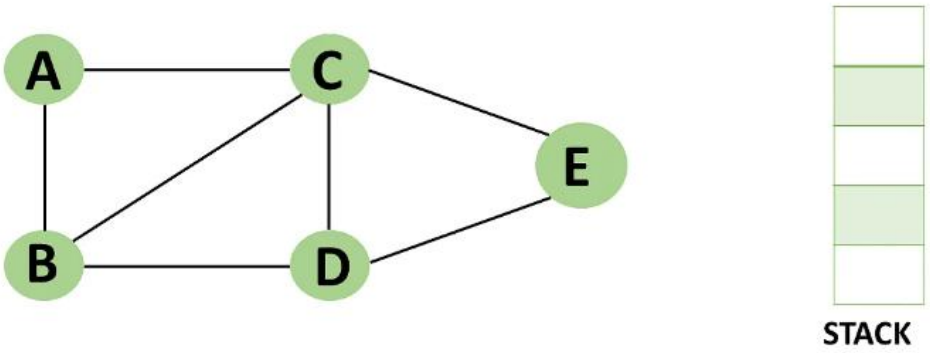
Step 8: Similarly, vertex C's adjacent vertices have already been visited; therefore, pop it from the stack.



Step 9: There is no more unvisited adjacent vertex of b, thus pop it from the stack.



Step 10: All of the nearby vertices of Vertex A, B, and C, have already been visited, so pop vertex A from the stack as well.

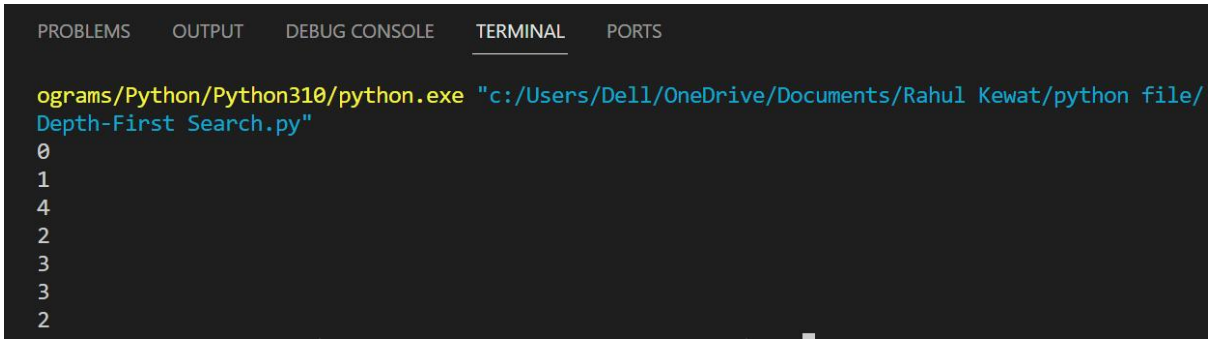


Code:

Depth-First Search.py

```python
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start)

    for next in graph[start] - visited:
        dfs(graph,next,visited)
    return visited
graph = {'0':set(['1','2']),
         '1':set(['0','3','4']),
         '2':set(['0']),
         '3':set(['1']),
         '4':set(['2','3'])}
dfs(graph,'0')
```

Output:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
ograms/Python/Python310/python.exe "c:/Users/Dell/OneDrive/Documents/Rahul Kewat/python file/
Depth-First Search.py"
0
1
4
2
3
3
2