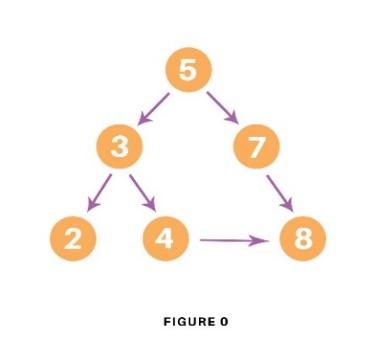
Artificial Intelligence Using Python

1. Write a Program to Implement Breadth First Search using Python.
2. Write a Program to Implement Depth First Search using Python
3. Write a Program to implement a tower of Hanoi using python.
4. Write a Program to implement a simple chatbot using python.
5. Write a Program to Implement a Linear Regression using Python.
6. Write a Program to implement Hangman Game using python.
7. Write a Program to implement the Time series using python.
8. Write a Program to implement the K means Clustering using python.
9. Write a Program to implement the Principle Component Analysis(PCA) using python.
10. Write a Program to implement the Support Vector Machine(SVM) using python.

**Pgm1:BFS ALGORITHM**



import matplotlib.pyplot as plt

import networkx as nx

graph = {  
 '5' : ['3','7'],  
 '3' : ['2', '4'],  
 '7' : ['8'],  
 '2' : [],  
 '4' : ['8'],  
 '8' : []  
}  
  
visited = [] *# List for visited nodes.*queue = [] *#Initialize a queue*def bfs(visited, graph, node): *#function for BFS* visited.append(node)  
 queue.append(node)  
  
 while queue: *# Creating loop to visit each node* m = queue.pop(0)  
 print (m, end = " ")  
 for neighbour in graph[m]:  
 if neighbour not in visited:  
 visited.append(neighbour)  
 queue.append(neighbour)  
*# Driver Code*print("Following is the Breadth-First Search")  
bfs(visited, graph, '5') *# function calling*

# Visualizing the graph

G = nx.Graph(graph)

pos = nx.spring\_layout(G) # Positions for all nodes

nx.draw(G, pos, with\_labels=True, node\_size=1000, node\_color="lightblue", font\_size=12, font\_weight="bold")

plt.title("Graph Visualization")

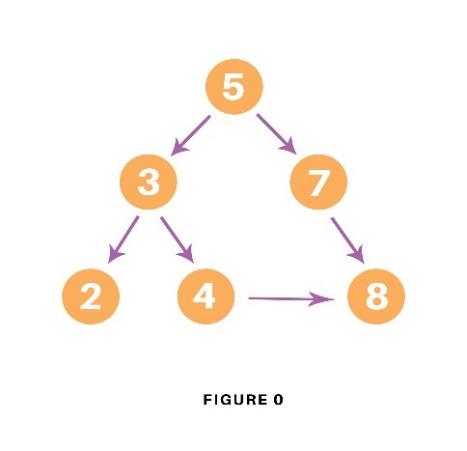
plt.show()

**Output of Program:**

Following is the Breadth-First Search

5 3 7 2 4 8

**Pgm 2: DFS**



import matplotlib.pyplot as plt

import networkx as nx

*# Using a Python dictionary to act as an adjacency list*graph = {  
 '5' : ['3','7'],  
 '3' : ['2', '4'],  
 '7' : ['8'],  
 '2' : [],  
 '4' : ['8'],  
 '8' : []  
}  
  
visited = set() *# Set to keep track of visited nodes of graph.*def dfs(visited, graph, node): *#function for dfs* if node not in visited:  
 print (node)  
 visited.add(node)  
 for neighbour in graph[node]:  
 dfs(visited, graph, neighbour)  
  
*# Driver Code*print("Following is the Depth-First Search")  
dfs(visited, graph, '5')

# Visualizing the graph

G = nx.Graph(graph)

pos = nx.spring\_layout(G) # Positions for all nodes

nx.draw(G, pos, with\_labels=True, node\_size=1000, node\_color="lightblue", font\_size=12, font\_weight="bold")

plt.title("Graph Visualization")

plt.show()

**Output of Program:**

**Following is the Depth-First Search**

**5**

**3**

**2**

**4**

**8**

**7**

**Pgm 3:TOWER OF HANOI**

*# Creating a recursive function*def tower\_of\_hanoi(disks, source, auxiliary, target):  
 if (disks == 1):  
 print('Move disk 1 from rod {} to rod {}.'.format(source, target))  
 return  
 *# function call itself* tower\_of\_hanoi(disks - 1, source, target, auxiliary)  
 print('Move disk {} from rod {} to rod {}.'.format(disks, source, target))  
 tower\_of\_hanoi(disks - 1, auxiliary, source, target)  
disks = int(input('Enter the number of disks: '))  
*# We are referring source as A, auxiliary as B, and target as C*tower\_of\_hanoi(disks, 'A', 'B', 'C') *# Calling the function*

**Output of Program :**

Enter the number of disks: 3

Move disk 1 from rod A to rod C.

Move disk 2 from rod A to rod B.

Move disk 1 from rod C to rod B.

Move disk 3 from rod A to rod C.

Move disk 1 from rod B to rod A.

Move disk 2 from rod B to rod C.

Move disk 1 from rod A to rod C.

**Pgm 4: AI Simple Chat Bot**

print("How are you?")  
print("Are you working?")  
print("What is your name?")  
print("what did you do yesterday?")  
print("Quit")

while True:  
 question = input("Enter one question from above list:")  
 question = question.lower()  
 if question in ['hi']:  
 print("Hello")  
 elif question in ['how are you?','how do you do?']:  
 print("I am fine")  
 elif question in ['are you working?','are you doing any job?']:  
 print("yes. I'am working in KLU")  
 elif question in ['what is your name?']:  
 print("My name is Emilia")  
 name=input("Enter your name?")  
 print("Nice name and Nice meeting you", name)  
 elif question in ['what did you do yesterday?']:  
 print("I saw Bahubali 5 times")  
 elif question in ['quit']:  
 break  
 else:  
 print("I don't understand what you said")

**Output Program :**

**How are you?**

**Are you working?**

**What is your name?**

**what did you do yesterday?**

**Quit**

**Enter one question from above list:Hi**

**Hello**

**Enter one question from above list:How are you?**

**I am fine**

**Enter one question from above list:WHat is your Name?**

**My name is Emilia**

**Enter one question from above list:what did you do yesterday?**

**I saw Bahubali 5 times**

**Enter one question from above list:time**

**I don't understand what you said**

**Enter one question from above list:quit**

**Pgm 5: Linear regression**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

# Sample data

X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1) # Feature matrix (reshape to a column vector)

y = np.array([2, 3, 4, 5, 6]) # Target values

# Create and fit the linear regression model

model = LinearRegression()

model.fit(X, y)

# Predictions

y\_pred = model.predict(X)

# Plotting the data and the linear regression line

plt.scatter(X, y, color='blue', label='Actual data')

plt.plot(X, y\_pred, color='red', label='Linear regression line')

plt.title('Linear Regression')

plt.xlabel('X')

plt.ylabel('y')

plt.legend()

plt.show()

# Printing coefficients

print('Intercept:', model.intercept\_)

print('Slope:', model.coef\_[0])

**Pgm 6: Implementation of Hangman Game in Python**

import time

from time import sleep

name = input("Enter Your Name:")

print( "Hello" + name)

print("Get ready!!")

print ("")

time.sleep(1)

print ("Let us play Hangman!!")

time.sleep(0.5)

word = "Flower"

wrd = ''

chance = 10

while chance > 0:

    failed = 0

    for char in word:

        if char in wrd:

            print (char)

        else:

            print( "\_")

            failed += 1

    if failed == 0:

        print( "You Won!!Congratulations!!" )

        break

    guess = input("Guess a Letter:")

    wrd = wrd+guess

    if guess not in word:

        chance -= 1

        print ("Wrong Guess! Try Again")

        print ("You have", + chance, 'more turn' )

        if chance == 0:

            print ("You Lose! Better Luck Next Time" )

**Output Program :**

Enter Your Name:sixtc c

Hellosixtc c

Get ready!!

Let us play Hangman!!

\_

\_

\_

\_

\_

\_

Guess a Letter:e

\_

\_

\_

\_

e

\_

Guess a Letter:l

\_

l

\_

\_

e

\_

Guess a Letter:o

\_

l

o

\_

e

\_

Guess a Letter:w

\_

l

o

w

e

\_

Guess a Letter:r

\_

l

o

w

e

r

Guess a Letter:f

Wrong Guess! Try Again

You have 9 more turn

\_

l

o

w

e

r

Guess a Letter:f

Wrong Guess! Try Again

You have 8 more turn

\_

l

o

w

e

r

Guess a Letter:F

F

l

o

w

e

r

You Won!!Congratulations!!

**Pgm 7: Write a Program to implement the Time series using python.**

import numpy as np

import matplotlib.pyplot as plt

# Generate some random time series data

np.random.seed(0)

time\_series = np.cumsum(np.random.randn(100)) + 50

# Define the moving average function

def moving\_average(data, window\_size):

return np.convolve(data, np.ones(window\_size)/window\_size, mode='valid')

# Compute the moving average

window\_size = 5

ma = moving\_average(time\_series, window\_size)

# Plot the original time series and the moving average

plt.figure(figsize=(10, 5))

plt.plot(time\_series, label='Original Time Series')

plt.plot(range(window\_size-1, len(ma)+window\_size-1), ma, label='Moving Average', color='red')

plt.xlabel("Time")

plt.ylabel("Value")

plt.title("Simple Moving Average")

plt.legend()

plt.show()

**Pgm 8: Write a Program to K Means Clustering using python.**

**#k means clustering**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_blobs

from sklearn.cluster import KMeans

# Generate some example data

X, \_ = make\_blobs(n\_samples=300, centers=4, cluster\_std=0.60, random\_state=0)

# Create a K-Means model

kmeans = KMeans(n\_clusters=4)

kmeans.fit(X)

# Predict the cluster for each data point

y\_kmeans = kmeans.predict(X)

# Plot the results

plt.scatter(X[:, 0], X[:, 1], c=y\_kmeans, s=50, cmap='viridis')

centers = kmeans.cluster\_centers\_

plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.75)

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.title("K-Means Clustering")

plt.show()

**Pgm 9: Write a Program to Principal component Analysis using python.**

**# Principal Component Analysis (PCA) for Dimensionality Reduction**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.decomposition import PCA

# Load the Iris dataset

iris = load\_iris()

X, y = iris.data, iris.target

# Apply PCA to reduce the data to 2 dimensions

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X)

# Plot the PCA results

plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=y, cmap='viridis')

plt.xlabel("First principal component")

plt.ylabel("Second principal component")

plt.title("PCA of Iris Dataset")

plt.show()

**Pgm 10: Write a Program to Support Vector Machine using python.**

**#. Support Vector Machine (SVM) for Classification**

import numpy as np

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, classification\_report

# Load the Iris dataset

iris = load\_iris()

X, y = iris.data, iris.target

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train the SVM model

model = SVC(kernel='linear')

model.fit(X\_train, y\_train)

# Make predictions

y\_pred = model.predict(X\_test)

# Calculate the accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy}")

# Print the classification report

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))