```
#Best first search(1)

tree ={
    'A':[['B',3],['C',2]],
    'B':[['A',5],['C',2],['D',2],['E',3]],
    'C':[['A',5],['B',3],['F',2],['G',4]],
    'D':[['H',1],['I',99]],
    'F': [['J',99]],
    'G':[['K',99],['L',3]]
}

Start='A'
Goal='E'
Closed = []
SUCCESS=True
FAILURE=False
State=FAILURE

def MOVEGEN(N):
      New_list=list()
      if N in tree.keys():
            New_list=tree[N]

      return New_list

def GOALTEST(N):
      if N == Goal:
            return True
      else:
            return False

def APPEND(L1,L2):
      New_list=list(L1)+list(L2)
      return New_list

def SORT(L):
      L.sort(key = lambda x: x[1])
      return L

def BestFirstSearch():
      OPEN=[[Start,5]]
      CLOSED=[]
      global State
      global Closed

      while (len(OPEN) != 0) and (State != SUCCESS):
            print("\n\n------------\n\n")
            N= OPEN[0]
            print("N=",N)
            del OPEN[0] #delete the node we picked

            if GOALTEST(N[0])==True:
                  State = SUCCESS
                  CLOSED = APPEND(CLOSED,[N])
                  print("CLOSED=",CLOSED)
            else:
                  CLOSED = APPEND(CLOSED,[N])
                  print("CLOSED=",CLOSED)
                  CHILD = MOVEGEN(N[0])
                  print("CHILD=",CHILD)
                  for val in CLOSED:
                        if val in CHILD:
                              CHILD.remove(val)
                  for val in OPEN:
                        if val in CHILD:
                              CHILD.remove(val)
                  OPEN = APPEND(CHILD,OPEN) #append movegen elements to OPEN
                  print("Unsorted OPEN=",OPEN)
                  SORT(OPEN)
                  print("Sorted OPEN=",OPEN)

      Closed=CLOSED
      return State

result=BestFirstSearch() #call search algorithm
print(Closed,"\n\n")
print(result)

###########################
#my best first search code#
###########################
from queue import PriorityQueue
v = 14
graph = [[] for i in range(v)]
# Function For Implementing Best First Search
# Gives output path having lowest cost
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]
        # Displaying the path having lowest cost
        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()


# Function for adding edges to graph
def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))
# The nodes shown in above example(by alphabets) are
# implemented using integers addedge(x,y,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 = 9
best_first_search(source, target, v)
#o/p: 0 1 3 2 8 9
###################################################################################
#############################

#Breadth first search(2)

tree={
    'A':['B','C','D'],
    'B':['E'],
    'C':['F','G'],
    'D':['H'],
    'E':['I','J'],
    'F':[],
    'G':[],
    'H':['K'],
    'I':[],
    'J':[],
    'K':[]
}

#create visited list and queue list
visited=[]
queue=[]

def bfs(visited,queue,node):
    visited.append(node)
    queue.append(node)

    while queue:
        m=queue.pop(0)
        print(m,end=" ")

        for neighbour in tree[m]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)

#main
print("Following is breadth first Search :\n")
bfs(visited,queue,"A")
###################################################################################
##################################

#Depth first search(3)


#generate tree using dictionary
tree={
    'A':['B','C','D'],
    'B':['E'],
    'C':['F','G'],
    'D':['H'],
    'E':['I','J'],
    'F':[],
    'G':[],
    'H':['K'],
    'I':[],
    'J':[],
    'K':[]
}

#create a set for storing visited node
visited=set()

#define a function dfs

def dfs(tree,visited,node):
    if node not in visited:                     #check if node is visited or not
        print(node)
        visited.add(node)                       #add node to visited
        for neighbour in tree[node]:            #traverse neighbour
            dfs(tree,visited,neighbour)         #call dfs recursively


#main
print("Folllowing is depth first search :\n")
dfs(tree,visited,'A')
###################################################################################
####################################

#Demonstration of simple machine learning program for line y=2x-1 (4)


#given list of values of x
X=[-1,-2,-3,-4,-5,6,7,8,9,10]
Y=[]

#generating y values from given equation
for number in X:
    Y.append(2*number-1)
print(Y)

import tensorflow as tf
import numpy as np
from tensorflow import keras

model1=tf.keras.Sequential([keras.layers.Dense(units=1,input_shape=[1])])
model1.compile(optimizer='sgd',loss='mean_squared_error')
print(model1.summary())

x=np.array(X)
y=np.array(Y)
model1.fit(x,y, epochs=500)
print(model1.predict([10]))
###################################################################################
##################################
```

```
#Matrix operations (5)

import tensorflow as tf

matrix1=[[4,2,1],[6,4,2],[6,8,9]]
matrix2=[[1,3,1],[1,2,9],[9,1,7]]

#matrix addition
sum=tf.add(matrix1,matrix2)
print("\nAddition of matrix1 and matrix2 :\n",sum)

#matrix subtraction
difference=tf.subtract(matrix1,matrix2)
print("\nDifference between matrix1 and matrix2 :\n",difference)

#matrix multiplication
product=tf.matmul(matrix1,matrix2)
print("\nMatrix multiplication :\n",product)

#matrix division
division=tf.divide(matrix1,matrix2)
print("\nMatrix division :\n",division)

#transpose
transpose=tf.transpose(matrix1)
print("\nTranspose of matrix1 :\n",transpose)

###################################################################################
##########################################

#Program for chatbot(git) (6)

from nltk.chat.util import Chat, reflections

pairs = [
    [
        r"my name is (.*)",
        ["Hello %1 . How can help you ?\n",]
    ],
    [
        r"I am (.*)",
        ["Hello %1 . How can help you ?\n",]
    ],
    [
        r"hi|hey|hello",
        ["Hello there.\n", "Hey :)\n",]
    ],
    [
        r"what is your name ?",
        ["I am GIDEON\n",]
    ],
    [
        r"nice",
        [":)\n","tell me more...\n"]
    ],
    [
        r"how are you ?",
        ["I am a Computer Program, I don't feel anything..\nWhat about you ?\n",]
    ],
    [
        r"sorry (.*)",
        ["It's OK !\n","Not a problem.\n",]
    ],
    [
        r"I am fine",
        ["Good. Anything else ?\n",]
    ],
    [
        r"(.*)good",
        [":)\n"]
    ],
    [
        r"i'm (.*) doing good",
        ["Pleasing for ears.\n","Nice :)\n",]
    ],
    [
        r"(.*) age?",
        ["I have been built recently.\n",]
    ],
    [
        r"what (.*) want ?",
        ["Haha... You are cool. be nice ;)\n",]
    ],
    [
        r"(.*) created ?",
        ["It's a secret ;)\n",]
    ],
    [
        r"(.*) (location|city)(.*)?i",
        ['You are in Kolhapur\n',]
    ],
    [
        r"how is weather in (.*)?",
        ["It's Cloudy.\n","It's Hot in here.\n","About to freeze...\n","Bright and 
humid...\n"]
    ],
    [
        r"(.*) work in (.*)",
        ["Yes. %1 is a good company.\n",]
    ],
    [
        r"(.*)raining(.*)?",
        ["No. It's clear in %1\n","Drizzling...\n"]
    ],
    [
        r"how (.*) health(.*)",
        ["My creator made me immortal ;)\n",]
    ],
    [
        r"(.*) (sports|game) ?",
        ["I like to play Computer games... :)\n"]
    ],
    [
        r"(.*) (sportsperson|player) ?",
        ["I follow Shroud Gaming.\n"]
    ],
    [
        r"(.*) (moviestar|actor)?",
        ["Ashok Saraf\n","Rowan Atkinson\n","Raj Kapoor :{\n","\n","Irfan Khan\
n","Robert Downey Jr.\n"]
    ],
    [
        r"suggest me some courses?",
        ["You can Search on google. It can suggest you plenty of the courses...\n"]
    ],
    [
        r"quit",
        ["Nice to meet you !\n","Bye !\n"]
    ],
    [
        r"(.*)",
        ["I don't understand what are you trying to say! Please say it again...\n"]
    ],

]

def chat():
    print("\n\nVoila ! I am Online...\nHii fellas! I am GEDION. What's Up??? :)\n")
    chat = Chat(pairs, reflections)
    chat.converse()

chat()
######
#mine#
######
# Simple Chatbot
# you can open notepad or chrome using open notepad and open chrome command

import random
import os
import webbrowser

name = " Bot"
weather = "rainy"
mood = "Happy"
responses = {
    "what's your name?": [
        "They call me {0}".format(name),
        "I usually go by {0}".format(name),
        "My name is the {0}".format(name)],
    "what's today's weather?": [
        "The weather is {0}".format(weather),
        "It's {0} today".format(weather),
        "Let me check, it looks {0} today".format(weather)],
    "Are you a robot?": [
        "What do you think?",
        "Maybe yes, maybe no!",
        "Yes, I am a robot with human feelings.", ],
    "hi": ["Hello "],
    "how are you?": [
        "I am feeling {0}".format(mood),
        "{0}! How about you?".format(mood),
        "I am {0}! How about yourself?".format(mood), ],
    "": [
        "Hey! Are you there?",
        "What do you mean by saying nothing?",
        "Sometimes saying nothing tells a lot :)", ],
    "notepad": ["please wait"],
    "Google Chrome": ["please wait"],
    "default": [
        "Sorry i can't get it"]
}


def op(message):
    if (message == "Google Chrome"):
        webbrowser.open('"C:\Program Files (x86)\Google\Chrome\Application\
chrome.exe"')
    else:
        os.system(message)


def respond(message):
    if message in responses:
        bot_message = random.choice(responses[message])
    else:
        bot_message = random.choice(responses["default"])
    return bot_message


def related(x_text):
    if "name" in x_text:
        y_text = "what's your name?"
    elif "weather" in x_text:
        y_text = "what's today's weather?"
    elif "robot" in x_text:
        y_text = "are you a robot?"
    elif "how are" in x_text:
        y_text = "how are you?"
    elif "open notepad" in x_text:
        y_text = "notepad"
        op(y_text)
    elif "open chrome" in x_text:
        y_text = "Google Chrome"
        op(y_text)
    elif "hi" in x_text:
        y_text = "hi"
    else:
        y_text = ""
    return y_text


def send_message(message):
    print(user_template.format(message))
    response = respond(message)
    print(bot_template.format(response))


print("BOT: What do you want me to call you?")
user_name = input()

bot_template = "BOT : {0}"
user_template = user_name + " : {0}"

while 1:
    my_input = input()
    my_input = my_input.lower()
    related_text = related(my_input)
    send_message(related_text)
    if my_input == "exit" or my_input == "stop":
        break
###################################################################################
############################################3

Installation of tensorflow: (7)

I. With PIP

    1. Windows
            syntax:
                    python3 -m pip install tensorflow

            # Verify install:
                    python3 -c "import tensorflow as tf;
print(tf.config.list_physical_devices('GPU'))"
    2. MacOS
            syntax: python3 -m pip install tensorflow

            # Verify install:
                    python3 -c "import tensorflow as tf;
print(tf.reduce_sum(tf.random.normal([1000, 1000])))"
    3. Linux
            syntax: python3 -m pip install tensorflow

            # Verify install:
                    python3 -c "import tensorflow as tf;
print(tf.reduce_sum(tf.random.normal([1000, 1000])))"

II. With Anaconda

            syntax:
                    conda install tensorflow
```