Slip 1-1 - Python program that demonstrates the hill climbing algorithm to find the maximum of a mathematical function.(For example f(x) = -x^2 + 4x) [10Marks]

```
import random

# Define the objective function (you can replace this with your own)
def objective_function(x):
    return -(x**2 + 4*x)  #Example: maximize x^2 + 4x (negate for max.)

def hill_climbing(max_iterations, step_size):
  current_solution = random.uniform(-10, 10)  # returns random floating
number between -10 & 10
  print(current_solution)
  current_value = objective_function(current_solution)

  for _ in range(max_iterations):
    neighbor = current_solution + random.uniform(-step_size, step_size)
    neighbor_value = objective_function(neighbor)

    if neighbor_value>current_value:
      current_solution = neighbor
      current_value = neighbor_value
  return current_solution, current_value

if __name__ == "__main__":
  max_iterations = 1000  # Maximum number of iterations
  step_size = 0.1  # Step size for making small changes

  final_solution, final_value = hill_climbing(max_iterations,
step_size)

  print("Final Solution:", final_solution)
  print("Objective Value at Final Solution:", final_value)

# https://www.mathway.com/Algebra --> refer for answer
```

```
Output : 9.049461082711886
Final Solution: -1.9998411314064843
Objective Value at Final Solution: 3.99999997476077
```

Slip 1-2 : Write a Python program to implement Depth First Search algorithm. Refer the following graph as an Input for the program. [Initial node=1,Goal node=8] [20 Marks]

```
def isEmpty(s):
    return s==[]

def isFull(s):
    return len(s)==5

def push(s,ele):
    s.append(ele)
```

```
def pop(s):
    return s.pop()

def peep(s):
    return s[-1]

def dfs(start,nodes):
    s=[]
    visited=[]
    for i in range(nodes+1):
        visited.append(0)
    push(s,start)
    print start
    while (not isEmpty(s)) :
        node = peep(s)
        visited[node]=1
        for x,y in mydict.items():
            if x==node :
                i=0
                for v in y:
                    i+=1
                    if visited[v]==0 :
                        break
                if (i==len(y)):
                    if isEmpty(s):
                        break
                    node=pop(s)
                if visited[v]==0:
                    push(s,v)
                    print v
                    visited[v]=1

nodes = int(raw_input("No of vertices"))
print nodes
mydict = dict()
for i in range (nodes):
    print "Vertices adjacent to",i+1,":"
    mydict[i+1]=input()
print mydict
start=1
dfs(start,nodes)
```

--- X ---

Slip 2-1 : Q.1) Write a python program to generate Calendar for the given month and year?.

[10 Marks]

```
# Import the 'calendar' module
import calendar
# Prompt the user to input the year and month
y = int(input("Input the year : "))
m = int(input("Input the month : "))
# Print the calendar for the specified year and month
print(calendar.month(y, m))
```

slip 2-2 : Q.2)Write a Python program to implement Depth First Search algorithm. Refer the following graph as an Input for the program.[Initial node=1,Goal node=7]. [20 Marks]

Refer to slip 1-2

-- X --

slip 3-1 : Q.1) Write a python program to remove punctuations from the given string [10 marks]

```
# define punctuation
punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''

my_str = "Hello!!!, he said ---and went."

# To take input from the user
# my_str = input("Enter a string: ")

# remove punctuation from the string
no_punct = ""
for char in my_str:
   if char not in punctuations:
       no_punct = no_punct + char
print(no_punct)
```

slip 3-2 : Q.2)Write a Python program to implement Depth First Search algorithm. Refer the following graph as an Input for the program.[Initial node=2,Goal node=7]. [20 Marks]

Refer to slip 1-2

-- X --

slip 4-1 : Q.1)Write a program to implement Hangman game using python. [10 Marks]

Description: Hangman is a classic word-guessing game. The user should guess the word correctly by entering alphabets of the user choice. The Program will get input as single alphabet from the user and it will matchmaking with the alphabets in the original

```
word = ("secret")          #here we set the secret. You can select any
word to play with.
guesses = ''               #creates an variable with an empty value
turns = 10                 #determine the number of turns

while turns > 0:           #check if the turns are more than zero
    failed = 0             # make a counter that starts with zero
    for char in word:      # for every character in secret_word
      if char in guesses: # see if the character is in players guess
        print (char,end=""),  # print then out the character
      else:
        print ("_",end=""),   # if not found, print a dash
        failed += 1           # and increase failed counter with one
    if failed == 0:           # if failed = zero  print You Won
      print ("You won")
      break                   # exit the script
    guess = input("guess a character:") # ask the user go guess a char
    guesses += guess                # set the players guess to guesses
    if guess not in word: # if the guess is not found in secret word
```

```
      turns -= 1                   # turns counter decreases with 1
      print ("Wrong")                            # print wrong
    print ("You have", + turns, 'more guesses' ) # remaining turn
    if turns == 0:                          # if the turns are equal to zero
      print ("You Lose"  )                  # print "You Lose"
```

slip 4-2 : Q.2) Write a Python program to implement Breadth First Search algorithm. Refer the following graph as an Input for the program.[Initial node=1,Goal node=8] [20 Marks]

```
def isEmpty(s):
    return s==[]

def addq(s,ele):
    s.insert(0,ele)

def remove(s):
    return s.pop()

def bfs(start,nodes):
    q=[]
    visited=[]
    for i in range(nodes+1):
        visited.append(0)
    addq(q,start)
    while not(isEmpty(q)):
        node = remove(q)
        visited[node]=1
        print node
        for x,y in mydict.items():
            if x==node :
                for v in y:
                    if visited[v]==0 :
                        addq(q,v)
                        visited[v]=1

nodes = int(raw_input("No of vertices"))
print nodes
mydict = dict()
for i in range (nodes):
    print "Vertices adjacent to",i+1,":"
    mydict[i+1]=input()
print mydict
start=1
bfs(start,nodes)
```

-- X --

slip 5-1 : Q.1) Write a python program to implement Lemmatization using NLTK [10 Marks]

```
import nltk
from nltk.stem import   WordNetLemmatizer

wordnet_lemmatizer = WordNetLemmatizer()

text = "studies studying cries cry "
nltk.download('punkt')
nltk.download('wordnet')
tokenization = nltk.word_tokenize(text)
for w in tokenization:
  print("Lemma for {} is {}".format(w,
wordnet_lemmatizer.lemmatize(w)))
```

slip 5-2 : Q.2) Write a Python program to implement Breadth First Search algorithm. Refer the following graph as an Input for the program.[Initial node=1,Goal node=8] [20 Marks]

Refer to slip 4-2

-- X --

slip 6-1 : Q.1) Write a python program to remove stop words for a given passage from a text file using NLTK?. [10 Marks]

```
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')
nltk.download('punkt')
data = text = open("input6.txt").read().lower()
stopWords = set(stopwords.words('english'))
words = word_tokenize(data)
wordsFiltered = [w for w in words if w not in stopWords]

print(wordsFiltered)
```

slip 6-2 : Q.2) Write a Python program to implement Breadth First Search algorithm. Refer the following graph as an Input for the program.[Initial node=1,Goal node=8]. [20Marks]

Refer to slip 4-2

-- X --

slip 7-1 : Q.1)Write a python program implement tic-tac-toe using alpha beeta pruning [10 Marks]

```
pip install easyAI
from easyAI import TwoPlayerGame
from easyAI.Player import Human_Player


class TicTacToe(TwoPlayerGame):
    """The board positions are numbered as follows:
    1 2 3
    4 5 6
    7 8 9
    """

    def __init__(self, players):
        self.players = players
        self.board = [0 for i in range(9)]
        self.current_player = 1  # player 1 starts.
    def possible_moves(self):
        return [i + 1 for i, e in enumerate(self.board) if e == 0]

    def make_move(self, move):
        self.board[int(move) - 1] = self.current_player

    def unmake_move(self, move):  # optional method (speeds up the AI)
        self.board[int(move) - 1] = 0

    def lose(self):
        """ Has the opponent "three in line ?" """
        return any(
            [
                all([(self.board[c - 1] == self.opponent_index) for c
in line])
                for line in [
                    [1, 2, 3],
                    [4, 5, 6],
                    [7, 8, 9],  # horiz.
                    [1, 4, 7],
                    [2, 5, 8],
                    [3, 6, 9],  # vertical
                    [1, 5, 9],
                    [3, 5, 7],
                ]
            ]
        )  # diagonal

    def is_over(self):
        return (self.possible_moves() == []) or self.lose()

    def show(self):
        print(
            "\n"
            + "\n".join(
                [
                    " ".join([[".", "O", "X"][self.board[3 * j + i]]
for i in range(3)])
                    for j in range(3)
                ]
            )
        )

    def scoring(self):
        return -100 if self.lose() else 0


if __name__ == "__main__":

    from easyAI import AI_Player, Negamax
    ai_algo = Negamax(6)
    TicTacToe([Human_Player(), AI_Player(ai_algo)]).play()
```

Slip 7-2 : Q.2) Write a Python program to implement Simple Chatbot. **[20Marks]**

```
name = "bot286"
monsoon = "rainy"
mood = "Smiley"
resp = {
"hi":[
    "hello",
    "hey"
],
"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(monsoon),
"It's {0} today".format(monsoon)],

"how are you?": [
"I am feeling {0}".format(mood),
"{0}! How about you?".format(mood),
"I am {0}! How about yourself?".format(mood), ],

"hello": [
 "Hey! hi",
"hello buddy?",
 ],

"bye": [
"bye c u soon"]
}

import random
def res(message):
  if message in resp:
    bot286_message = random.choice(resp[message])
  else:
    bot286_message = random.choice(resp["default"])
  return bot286_message

def relate(xtext):
  if "name" in xtext:
    ytext = "what's your name?"
  elif "hello" in xtext:
    ytext = "hi"
  elif "weather" in xtext:
    ytext = "what's today's weather?"
  elif "how are" in xtext:
    ytext = "how are you?"
  else:
    ytext = "bye"
  return ytext

def send_message(message):
  #print((message))
  response = res(message)
  print((response))

while(1):
  my_input = input()
  my_input = my_input.lower()
  related_text = relate(my_input)
  send_message(related_text)
  if my_input == "quit":
    break
```

-- X --

Slip 8-1 : Q.1) Write a Python program to accept a string. Find and print the number of upper case alphabets and lower case alphabets. [10 Marks]

```
Str = input("Enter a string: ")
lower=0
upper=0
for i in Str:
      if(i.islower()):
            lower+=1
      else:
            upper+=1
print("The number of lowercase characters is:",lower)
print("The number of uppercase characters is:",upper)
```

slip 8-2 : Q.2) Write a Python program to solve tic-tac-toe problem. [20 Marks]

Refer to slip 7-1

-- X --

Slip 9-1 : Q.1) Write python program to solve 8 puzzle problem using A* algorithm [10 marks]

Will replace this by tower of Hanoi

Slip 9-2 : Q.2) Write a Python program to solve water jug problem. 2 jugs with capacity 5 gallon and 7 gallon are given with unlimited water supply respectively. The target to achieve is 4 gallon of water in second jug. [20 Marks]

```
bfs_queue = [(0, 0, None, 0)] # 4_jug, 3_jug, parent state, rule
applied
```

```
visited_states = [] #all the states which are not to be considered for
exploration
output_path = [] #to print the path
print ("Enter Jug 1 Capacity, Jug 2 Capacity, Target (space
separated)\n : "),
jug1, jug2, target = map(int, input().split())
print ("\nInitial State. ->  (0, 0)")
print ("Goal State. -> (%d, N) or (M, %d) \n"%(target, target))
# Production rules
def success(M, N) :
    global target
    return M == target or N == target

def fill_M(M, N) :
    global jug1
    if M < jug1 :
        M = jug1
    return M, N, 1

def fill_N(M, N) :
    global jug2
    if N < jug2 :
        N = jug2
    return M, N, 2

def empty_M(M, N) :
    if M > 0 :
        M = 0
    return M, N, 3

def empty_N(M, N) :
    if N > 0 :
        N = 0
    return M, N, 4

def pour_N_to_M(M, N) :
    global jug1
    if N > 0 and M + N >= jug1 :
        M = jug1
        N = N - (jug1 - M)
    return M, N, 5

def pour_M_to_N(M, N) :
    global jug2
    if M > 0 and M + N >= jug2 :
        M = M - (jug2 - N)
        N = jug2
    return M, N, 6

def pour_all_N_to_M(M, N) :
    global jug1
    if M + N <= jug1 and N > 0 :
        M = M + N
        N = 0
    return M, N, 7

def pour_all_M_to_N(M, N) :
    global jug2
    if M + N <= jug2 and M > 0 :
        N = N + M
        M = 0
    return M, N, 8

#-----------------------------------
# chekcing if the state has been visited
def state_visited(x, y) :
    for val in visited_states :
        if val[0] == x and val[1] ==  y :
            return True
    return False

if __name__ == '__main__' :

    rules = [fill_M, fill_N, empty_M, empty_N,
    pour_N_to_M, pour_M_to_N, pour_all_N_to_M, pour_all_M_to_N]
#function list

    rule_text = ["1. Fill the {}-gallon jug.".format(jug1), "2. Fill
the {}-gallon jug.".format(jug2),
    "3. Empty {}-gallon jug.".format(jug1), "4. Empty {}-gallon
jug.".format(jug2),
    "5. Pour water from {}-gallon to {}-gallon jug.".format(jug2,
jug1),
    "6. Pour water from {}-gallon to {}-gallon jug.".format(jug1,
jug2),
    "7. Pour all the water from {}-gallon to {}-gallon
jug.".format(jug2, jug1),
    "8. Pour all the water from {}-gallon to {}-gallon
jug.".format(jug1, jug2)]
    done = False

    #Approach using breadth first search
    while bfs_queue :
        M, N, parent, rule_no = bfs_queue.pop(0)
        visited_states.append((M, N, parent, rule_no))
        for func in rules : #applying all the rules to the current
state
            x, y, rule_no = func(M, N)
            if x < 0 or y < 0 :
                continue
            if success(x, y) :
                done = True
                visited_states.append((x, y, (M, N), rule_no))
                break
            if not state_visited(x, y) : #appending the newly formed
states in the queue
                bfs_queue.append((x, y, (M, N), rule_no))


        if done :
            print ("\nGOAL Reached!")
            break
    #end_while
    if not done :
        print ("No Solution Exists!")
        print ("Terminated.")
        exit(0)


    index = len(visited_states) - 1
    output_path.append((visited_states[index][:2], None))
    parent = visited_states[index][2]
    rule = visited_states[index][3]
    # finding the path from bottom to top
    while parent :
        output_path.append((parent, rule))
        for val in visited_states :
            if val[0] == parent[0] and val[1] ==  parent[1] :
                parent = val[2]
                rule = val[3]
                break

    op = list(reversed(output_path))
    #printing the states and the rules applied.
    for i in range(0, len(op)) :
        if op[i][1] is not None :
            print (op[i][0], " -> ", rule_text[op[i][1] - 1])
    print (op[len(op) - 1][0], " ->  Goal State.")
    print ("Terminated.")
```

-- X --

Slip 10-1 : Q.1) Write Python program to implement crypt arithmetic problem

TWO+TWO=FOUR **[10 Marks]**

```
from itertools import permutations

def solve_cryptarithmetic(puzzle):
  unique_chars = set("".join(puzzle))
  if len(unique_chars) > 10:
    print("Too many unique characters for a valid puzzle.")
    return

  for perm in permutations('0123456789', len(unique_chars)):
    char_to_digit = dict(zip(unique_chars, perm))
    if char_to_digit[puzzle[0][0]] == '0' or
char_to_digit[puzzle[1][0]] == '0' or char_to_digit[puzzle[2][0]] ==
'0':
      continue

    expression = "".join([char_to_digit[char] for char in puzzle[0]]) +
"+" + \
    "".join([char_to_digit[char] for char in puzzle[1]]) + "==" + \
    "".join([char_to_digit[char] for char in puzzle[2]])

    if eval(expression):
      print(f"Solution found: {expression}")

# Example puzzle: TWO + TWO = FOUR
puzzle = ["TWO", "TWO", "FOUR"]
solve_cryptarithmetic(puzzle)
```

slip 10-2:Q.2) Write a Python program to implement Simple Chatbot.

[20 Marks]

Refer to slip 7-2

-- X --

Slip 11-1 : Q.1) Write a python program using mean end analysis algorithmproblem of transforming a string of lowercase letters into another string. [10 Marks]

Will replace this by hangman

Slip 11-2 : Q.2) Write a Python program to solve water jug problem. Two jugs with capacity 4 gallon and 3 gallon are given with unlimited water supply respectively. The target is to achieve 2 gallon of water in second jug. [20 Marks]

Refer to slip 9-2

-- X --

Slip 12-1 : Q.1) Write a python program to generate Calendar for the given month and year?.

[10Marks]

Refer to slip 2-1

Slip 12-2 : Q.2)Write a Python program to simulate 4-Queens problem. [20Marks]

```
# Taking number of queens as input from user
print ("Enter the number of queens")
N = int(input())

# here we create a chessboard
# NxN matrix with all elements set to 0
board = [[0]*N for _ in range(N)] # _ indiactes loop variable is
irrelevant

def attack(i, j):
    #checking vertically and horizontally
    for k in range(0,N):
        if board[i][k]==1 or board[k][j]==1:
            return True
    #checking diagonally
    for k in range(0,N):
        for l in range(0,N):
            if (k+l==i+j) or (k-l==i-j):
                if board[k][l]==1:
                    return True
    return False

def N_queens(n):
    if n==0:
        return True
    for i in range(0,N):
        for j in range(0,N):
            if (not(attack(i,j))) and (board[i][j]!=1):
                board[i][j] = 1
                if N_queens(n-1)==True:
                    return True
                board[i][j] = 0

    return False
```

`N_queens(N)`   # **pass value of N as 4**

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

-- X --

slip 13-1 : Q.1Write a Python program to implement Mini-Max Algorithm. [10 Marks]

```
import math

def minimax (curDepth, nodeIndex, maxTurn, scores, targetDepth):
  # base case :targetDepth reached
  if (curDepth == targetDepth):
    return scores[nodeIndex]

  if (maxTurn):
    return max(minimax(curDepth + 1, nodeIndex * 2, False, scores,
targetDepth),
    minimax(curDepth + 1, nodeIndex * 2 + 1,       False, scores,
targetDepth))

  else:
    return min(minimax(curDepth + 1, nodeIndex * 2, True, scores,
targetDepth),
    minimax(curDepth + 1, nodeIndex * 2 + 1, True, scores,
targetDepth))

  # Driver code
scores = [3, 5, 2, 9, 12, 5, 23, 23]
treeDepth = math.log(len(scores), 2)
print("The optimal value is : ", end = "")
print(minimax(0, 0, True, scores, treeDepth))
```

slip 13-2 : Q.2) Write a Python program to simulate 8-Queens problem. [20 Marks]

Refer to slip 12-2 , just put value of N as 4 in driver program

-- X --

slip 14-1 : Q.1) Write a python program to sort the sentence in alphabetical order? [10Marks]

```
#Program to sort alphabetically words form a string provided by user

my_str = "Hello this Is an Example With cased letters"

#my_str = input("Enter a string: ")# To take input from the user
words = [word.lower() for word in my_str.split()]   # breakdown the
string into a list of words
words.sort()           # sort the list
print("The sorted words are:")     # display the sorted words
for word in words:
  print(word)
```

slip 14-2 :Q.2) Write a Python program to simulate n-Queens problem. [20Marks]

Refer to slip12-2

-- X --

Not going to keep Slip 15 in the exam

-- X --

slip 16-1 : Q.1) Write a Program to Implement Tower of Hanoi using Python [10 Marks]

```
# Recursive Python function to solve the tower of hanoi

def TowerOfHanoi(n , source, destination, auxiliary):
  if n==1:
    print ("Move disk 1 from source",source,"to
destination",destination)
    return
  TowerOfHanoi(n-1, source, auxiliary, destination)
  print ("Move disk",n,"from source",source,"to
destination",destination)
  TowerOfHanoi(n-1, auxiliary, destination, source)

# Driver code
n = 3
TowerOfHanoi(n,'A','B','C')
# A, C, B are the name of rods
```

Slip 16-2 : Q.2) Write a Python program to solve tic-tac-toe problem. [20 Marks]

Refer to slip 7-1

-- X --

Slip 17-1 : Q.1) Python program that demonstrates the hill climbing algorithm to find the maximum of a mathematical function. [10Marks]

Refer to slip 1-1

Slip 17-2 : Q.2) Write a Python program to implement A* algorithm. Refer the following graph as an Input for the program.[Start vertex is A and Goal Vertex is G] [20 Marks]

Will replace this program by minimax

-- X --

Slip 18-1 : Q.1).Write a python program to remove stop words for a given passage from a text file using NLTK?. [10Marks]

Refer to slip 6-1

Slip 18-2 : Q.2) Implement a system that performs arrangement of some set of objects in a room. Assume that you have only 5 rectangular, 4 square-shaped objects. Use A* approach for the placement of the objects in room for efficient space utilisation. Assume suitable heuristic, and dimensions of objects and rooms. (Informed Search) [20 Marks]

Will replace this program by water-jug

-- X --

Will not keep Slip 19 in the exam

-- X --

Slip 20-1 : Q.1) Build a bot which provides all the information related to you in college [10Marks]

Refer to slip 7-2 (just replace the questions by college related information

Slip 20-2 : Q.2) Write a Python program to implement Mini-Max Algorithm. [20Marks]

Refer to slip 13-1

-- X --

Slip 21-1 : Q.1)Write a python program to remove punctuations from the given string?[10 Marks]

Refer to slip 3-1

Slip 21-2 : Q.2)Write a Python program for the following Cryptarithmetic problems. [20 Marks]

GO + TO = OUT

Refer to slip 10-1

-- X --

Slip 22-1 : Q.1) Write a Program to Implement Alpha-Beta Pruning using Python [10 Marks]

Will replace this by lemmatization slip 5-1

Slip 22-2 : Q.2) Write a Python program to implement Simple Chatbot[20 Marks]

Refer to slip 7-2

-- X --

Slip 23-1 : Q.1) Write a Program to Implement Tower of Hanoi using Python. [10 Marks]

Refer to slip 16-1

Slip 23-2 : Q.2) Write a Python program for the following Cryptarithmetic problems

SEND + MORE = MONEY [20Marks]

Refer to slip 10-1

-- X --

Slip 24-1 : Q.1)Write a python program to sort the sentence in alphabetical order? [10 Marks]

Refer to slip 14-1

Slip 24-2: Q.2) Write a Python program for the following Cryptorithmetic problems

CROSS+ROADS = DANGER [20Marks]

Refer to slip 10-1

-- X --

Will not keep Slip 25 in the exam

-- X --