AI\calendar - [2,12].py

```python
# Q)  Write a python program to generate Calendar for the given month and year? [2, 12]
import calendar

year = int(input("Enter the year: "))
month = int(input("Enter the month: "))

cal = calendar.TextCalendar(calendar.SUNDAY)

month_cal = cal.formatmonth(year,month)

print(month_cal)

_____________________________________________________________________________
# Q)  Write a python program to remove punctuations from the given string? [3, 21]
import string

str = input("Enter string: ")

no_punc = ""

for char in str:
    if char not in string.punctuation:
        no_punc += char

print(no_punc)

_____________________________________________________________________________
# Q) Write a python program to sort the sentence in alphabetical order? [14, 24]
sentence = input("Enter the sentence : ")
words = sentence.split()
words.sort()
print("Sentence After Sorting The Words : ")
print(" ".join(words))

_____________________________________________________________________________
# Q) Write a python program to remove stop words for a given passage from a text file using NLTK?. [6]
import nltk
from nltk.corpus import stopwords

file = open("sample.txt","r")
text = file.read()
file.close()

tokens = nltk.word_tokenize(text)

stop_words = set(stopwords.words('english'))
filtered_tokens = []

for w in tokens:
    if w.lower() not in stop_words:
        filtered_tokens.append(w)

filtered_text = " ".join(filtered_tokens)

print(filtered_text)

```

AI\Lemmatization [5].py

```python
# Q)  Write a python program to implement Lemmatization using NLTK [5]

import nltk
from nltk.stem import WordNetLemmatizer

# Ensure necessary resources are available
nltk.download('punkt_tab')
nltk.download('punkt')
nltk.download('wordnet')

text = "studies studying studied"

# Tokenize the text
tokenized_text = nltk.word_tokenize(text)
print("Tokenized text:", tokenized_text)

# Initialize the lemmatizer
lemmatizer = WordNetLemmatizer()

# Lemmatize tokens
print("Lemmatized tokens:")
for token in tokenized_text:
    print(lemmatizer.lemmatize(token))



_____
# Q) Write a Program to Implement Tower of Hanoi using Python [16,23]

def tower_of_hanoi(n, source, target, auxiliary):
    if n == 1:
        print(f"Move disk 1 from {source} to {target}")
        return
    tower_of_hanoi(n - 1, source, auxiliary, target)
    print(f"Move disk {n} from {source} to {target}")
    tower_of_hanoi(n - 1, auxiliary, target, source)

if __name__ == "__main__":
    # Define the number of disks and the tower names
    num_disks = int(input("Enter the number of disks: "))
    source_tower = "Source"
    target_tower = "Target"
    auxiliary_tower = "Auxiliary"

    # Run Tower of Hanoi algorithm
    tower_of_hanoi(num_disks, source_tower, target_tower, auxiliary_tower)

```

AI\ChatBot - [7, 10, 20, 22, 25].py

```python
# Q) Build a bot which provides all the information related to you in college [7,10,20,22,25]

import random

def chatbot():
    print("Welcome to the College Information Chatbot! Type 'exit' to end.")

    responses = {
        "hi": "Hello! How can I assist you?",
        "application deadline": "The application deadline is August 31.",
        "admission requirements": "Admission requirements include SSC, HSG 2 TY Marksheet.",
        "application status": "Application results will be available in September.",
        "scholarships": "Various scholarships, grants, and work-study options are available.",
        "bye": "Goodbye!"
    }

    default_responses = [
        "I'm not sure about that. Can you ask something else?",
        "I don't have information on that topic. Is there anything else I can help with?",
        "That's beyond my knowledge. Can I assist you with something else?"
    ]

    while True:
        user_input = input("You: ").lower().strip()

        if user_input == 'exit':
            print("Goodbye!")
            break
        elif user_input in responses:
            print(f"Bot: {responses[user_input]}")
        else:
            print(f"Bot: {random.choice(default_responses)}")

if __name__ == "__main__":
    chatbot()

```

AI\BFS - [4,5,6].py

```python
# Q) Write a Python program to implement Breadth First Search algorithm. Refer the following graph as an Input for the program. [4,5,6]

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

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

        if node == target:
            print(f"\nTarget node '{target}' found.")
            return

        for neighbor in graph[node]:
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append(neighbor)

    print(f"\nTarget node '{target}' not found.")

graph = {
    '1': ['2', '3'],
    '2': ['4', '5'],
    '3': ['6', '7'],
    '4': ['8'],
    '5': ['8'],
    '6': ['8'],
    '7': ['8'],
    '8':[]
}

start = '1'
target = '8'

print("Breadth-First Search Traversal:")
bfs(graph, start, target)

```

AI\DFS [2,3].py

```python
# Q) Write a Python program to implement Depth First Search algorithm. Refer the following graph as an Input for the program. [2,3]

def dfs(graph, start, visited,target):
    if start not in visited:
        print(start, end=" ")
        visited.add(start)
        if start == target:
            print(f"Target found {target}")
            return True

        for neighbor in graph[start]:
            if dfs(graph, neighbor, visited,target):
                return True

    return False


# Example usage
graph = {
    '1': ['2', '3'],
    '2': ['4'],
    '3': ['2'],
    '4': ['5','6'],
    '5': ['3','7'],
    '6': [],
    '7': ['6'],
}
visited = set()
start = '1'
target = '7'

print("Depth-First Search Traversal:")
dfs(graph, start,visited, target)

```

AI\Alpha-Beta [22].py

```python
# Q) Write a Program to Implement Alpha-Beta Pruning  using Python [22]
import math

def alphaBeta(depth, nodeIndex,alpha, beta, maxPlayer, values, target):
    if depth == target:
        return values[nodeIndex]

    if maxPlayer:
        best = -math.inf
        for i in range(2):
            val = alphaBeta(depth + 1, nodeIndex * 2 + i,alpha, beta, False, values, target)
            best = max(best, val)
            alpha = max(alpha, best)
            if beta <= alpha:
                break
        return best
    else:
        best = math.inf
        for i in range(2):
            val = alphaBeta(depth + 1, nodeIndex * 2 + i,alpha, beta, True, values, target)
            best = min(best, val)
            beta = min(beta, best)
            if beta <= alpha:
                break
        return best

# Test the algorithm
values = [-1, 4, 2, 6, -3, -5, 0, 7]
print(f"The optimal value is: {alphaBeta(0, 0, -math.inf, math.inf, True, values, 3)}")
```

```python
# Q)  Write a Python program to implement Mini-Max Algorithm. [13 20]

import math

def minmax(depth, nodeIndex, maxPlayer, values, target):
    if depth == target:
        return values[nodeIndex]

    if maxPlayer:
        best = -math.inf
        for i in range(2):
            val = minmax(depth + 1, nodeIndex * 2 + i, False, values, target)
            best = max(best, val)
        return best
    else:
        best = math.inf
        for i in range(2):
            val = minmax(depth + 1, nodeIndex * 2 + i, True, values, target)
            best = min(best, val)
        return best

# Test the algorithm
values = [-1, 4, 2, 6, -3, -5, 0, 7]
print(f"The optimal value is: {minmax(0, 0, True, values, 3)}")
```

AI\Two + Two [10,21,23,24].py

```python
from itertools import permutations


def solve_cryptarithmetic(equation):
    unique_letters = set("".join(equation))

    if len(unique_letters) > 10:
        return "Invalid equation. More than 10 unique letters."

    # Convert the list of unique letters to a sorted list for consistent ordering
    unique_letters = sorted(unique_letters)

    for perm in permutations(range(10), len(unique_letters)):
        mapping = dict(zip(unique_letters, perm))

        # Skip leading zero cases
        if any(mapping[equation[i][0]] == 0 for i in range(len(equation))):
            continue

        # Calculate the values of each word based on the current mapping
        left_sum = sum(mapping[letter] * 10**i for i, letter in enumerate(equation[0][::-1]))
        right_sum = sum(mapping[letter] * 10**i for i, letter in enumerate(equation[1][::-1]))
        result_sum = sum(mapping[letter] * 10**i for i, letter in enumerate(equation[2][::-1]))

        # Check if the equation holds
        if left_sum + right_sum == result_sum:
            return {letter: mapping[letter] for letter in unique_letters}

    return "No solution found"

if __name__ == "__main__":
    equation = ["TWO", "TWO", "FOUR"]
    solution = solve_cryptarithmetic(equation)

    if isinstance(solution, dict):
        print("Solution found:")
        for letter, value in solution.items():
            print(f"{letter}: {value}")
    else:
        print(solution)

```

AI\N - Queens [12,13, 14, 25].py

```python
# Q)  Write a Python program to simulate n-Queens problem. [12, 13, 14, 25]

N = 4

def print_board(board):
    for i in range(N):
        for j in range(N):
            print(board[i][j], end=" ")
        print()

def is_safe(board, row, col):
    # Check the current row on the left side
    for i in range(col):
        if board[row][i] == 1:
            return False

    # Check upper diagonal on the left side
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    # Check lower diagonal on the left side
    for i, j in zip(range(row, N), range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    return True

def solveNqueen(board, col):
    if col >= N:
        return True

    for i in range(N):
        if is_safe(board, i, col):
            board[i][col] = 1  # Place queen
            if solveNqueen(board, col + 1):  # Recur to place rest
                return True
            board[i][col] = 0  # Backtrack if placing queen doesn't work

    return False

board = [[0] * N for _ in range(N)]

if not solveNqueen(board, 0):
    print("Solution does not exist")
else:
    print_board(board)

```

AI\tic-tac-toe [7,8,16].py

```python
# Q) Write a Python program to solve tic-tac-toe problem.  [7,8,16]

def print_board(board):
    for i in range(3):
        print(" | ".join(board[i*3:(i+1)*3]))
        if i < 2:
            print("-------")

def check_winner(board, player):
    for i in range(3):
        if all(board[i*3+j] == player for j in range(3)) or \
           all(board[i+j*3] == player for j in range(3)):
            return True
    if all(board[i] == player for i in [0, 4, 8]) or \
       all(board[i] == player for i in [2, 4, 6]):
        return True
    return False

def is_full(board):
    return all(cell != " " for row in board for cell in row)

def play_game():
    board = [" " for _ in range(9)]
    current_player = "X"

    while True:
        print_board(board)

        move = int(input(f"Player {current_player}'s turn. Enter move (1-9): ")) - 1

        if move < 0 or move > 8:
            print("Invalid input. Enter a number between 1 and 9.")
            continue

        if board[move] != " ":
            print("That cell is already occupied. Try again.")
            continue

        board[move] = current_player

        if check_winner(board, current_player):
            print_board(board)
            print(f"Player {current_player} wins!")
            break

        if is_full(board):
            print_board(board)
            print("It's a tie!")
            break

        current_player = "O" if current_player == "X" else "X"

if __name__ == "__main__":
    play_game()
```