

[파이썬 중간고사 과제]



**■ 과 목 명 : 파이썬 프로그래밍**

**■ 담당교수 : 나웅수 교수님**

**■ 제 출 일 : 2022.11.02**

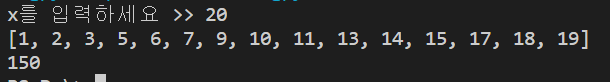
**■ 학과 : 소프트웨어**

**■ 학번 : 202202201**

**■ 성명 : 나권엽**

1. 4와 8의 배수가 아닌 수 리스트 출력

#실행화면



#핵심코드

for i in range(1,x+1):

    num\_list.append(i)

special\_num\_list = [] #4의 배수가 아닌 수를 저장할 리스트 생성

for num in num\_list:

    if num % 4 != 0:

        special\_num\_list.append(num)

        sum+=num

#구현방식

먼저 num\_list에 1부터 x까지의 수를 전부 append를 이용하여 저장한다

4와 8의 배수가 아닌 수는 4의 배수가 아닌 수와 동일하므로 4의 배수가 아닌 수를 저장할 special\_num\_list를 만든다

num\_list의 모든 수 중 4의 배수가 아니라면 이를 special\_num\_list에 넣고 4의 배수가 아닌 수의 합을 의미하는 변수 sum에 더해주고 모두 검사한 후 special\_num\_list와 sum을 출력한다.

2. atbash 암호법 구현

#실행화면

텍스트이(가) 표시된 사진

자동 생성된 설명

-암호문 입력

텍스트이(가) 표시된 사진

자동 생성된 설명

-알파벳과 숫자가 아닌 문장 입력

#핵심코드

def atbash(message):

    translated= ""

    for ch in message:

        if ch >='a' and ch<='z':

            translated=translated+chr(219-ord(ch))

        elif ch>='A' and ch<='Z':

            translated=translated+chr(155-ord(ch))

        elif ch>='0' and ch<='9':

            translated=translated+chr(105-ord(ch))

        else:

            print("입력은 알파벳과 숫자로만 입력하세요!!!")

            sys.exit()

    return translated

text=list(map(str, input("평문을 입력하세요 >> ")))

cipher=atbash(text)

#구현방식

문자열에서 문자를 차례차례 꺼내와 원래 문자와 암호화된 문자의 합이 영어 소문자의 경우

219, 대문자의 경우 155, 숫자의 경우 105인 것을 이용하여 문자를 아스키코드화 한 후 이를 뺀 값에 해당하는 문자를 변수 tranlated에 저장한다.

Text에 평문을 입력받은 후 이를 atbash 함수에 넣으면 암호화된 문장이 나오고 이를 cipher에

저장 후 출력한다

3. 로봇의 좌표 이동

#실행화면

텍스트이(가) 표시된 사진

자동 생성된 설명

#핵심코드

command = input("명령을 입력하세요 >> ").split(", ")

now=[0,0]

def Current\_position(command):

    for i in range(len(command)):

        if(command[i][0]) == 'U':

            now[1]+=int(command[i][1:])

        elif(command[i][0]) == 'D':

            now[1]-=int(command[i][1:])

        elif(command[i][0]) == 'L':

            now[0]-=int(command[i][1:])

        elif(command[i][0]) == 'R':

            now[0]+=int(command[i][1:])

        else:

            print("알 수 없는 명령이 존재합니다.")

            sys.exit()

#구현방식

Command에 명령을 입력받는 데 이때 split(“, “)을 사용하여 ,를 구분자로 잘라준다.

Command의 길이만큼 for문을 돌려 command[i][0] 즉 명령에서 상하좌우를 판단하는 알파벳을 판별하고 각 경우에 따라 좌표에 command[i][1]에 저장되어 있는 이동거리를 계산해준 후 now 좌표를 출력하면 된다.

4. 리스트 queque 구현

#실행화면

화살이(가) 표시된 사진

자동 생성된 설명

#핵심코드

    def data\_In(x):

        y=0

        y=len(queque)

        if(y<max):

            queque.append(command[1])

            print("현재 큐 현황: ", end="")

            print(queque)

        else:

            print("오버플로우입니다!!! 큐가 꽉 찼습니다!")

    def data\_pop():

        z=len(queque)

        if z==0:

            print("언더플로우입니다! 큐가 없습니다!!!")

        else:

            print("현재 큐 현황: ", end="")

            print(queque)

            print("출력된 데이터", z)

#구현방식

-Data\_ln 함수

Y에 queque의 길이를 저장한 후 이를 max값과 비교하여 더 작다면 queque에 입력값을 추가한

후 현재 큐 현황을 출력한다

그렇지 않다면 오버플로우입니다라는 문구와 함께 queque에 입력값을 추가하지 않는다

-data\_pop 함수

Z에 queque의 길이를 저장하고 먄약 z가 0이라면 queque에 어떤 값도 존재하지 않는 것이기

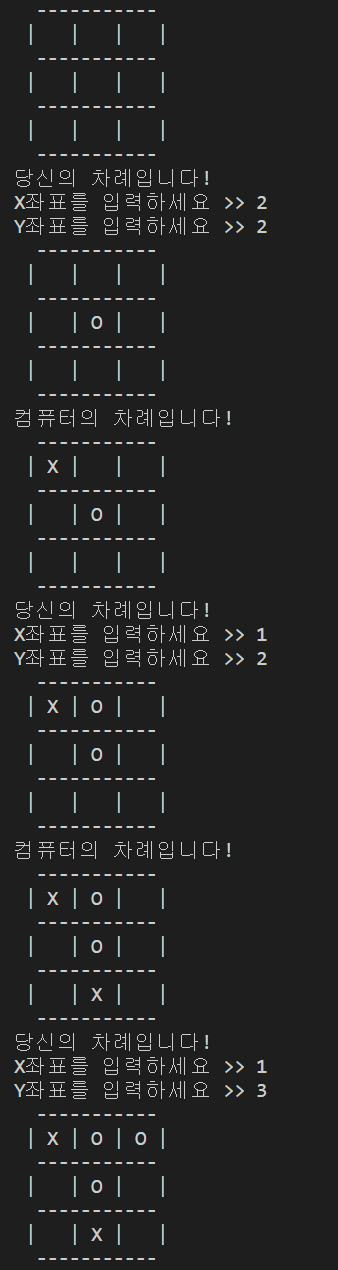
때문에 언더플로우이다.

0이 아니라면 현재 큐 현황과 함께 queque의 길이와 같은 의미를 가진 출력된 데이터의 수를

출력해준다

5. 틱택토

#실행화면

텍스트이(가) 표시된 사진

자동 생성된 설명

#보드판

def printBoard(board):

    print("  -----------")

    print(" | " + board[1] + " | " + board[2] + " | " + board[3] + " |")

    print("  -----------")

    print(" | " + board[4] + " | " + board[5] + " | " + board[6] + " |")

    print("  -----------")

    print(" | " + board[7] + " | " + board[8] + " | " + board[9] + " |")

    print("  -----------")

플레이하는 공간을 board로 지정한 후 판을 만들었습니다.

처음에는 2차원 배열로 하려 했지만 코드를 짤 때 헷갈리고 가독성을 높이기 위해 좌표값을

입력받은 후 if문을 이용하여 이를 1~9의 숫자에 각각 대입했습니다

#space is free?

def spaceIsFree(position):

    if board[position] == ' ':

        return True

    else:

        return False

보드판의 해당 위치가 공백인지 판단하여 공백이라면 True값을 아니라면 False값을 반환하여

플레이어와 컴퓨터가 놓았던 곳에 놓는 것을 방지합니다.

#승리 판별

def check\_win():

    if (board[1] == board[2] and board[1] == board[3] and board[1] != ' '):

        return True

    elif (board[4] == board[5] and board[4] == board[6] and board[4] != ' '):

        return True

    elif (board[7] == board[8] and board[7] == board[9] and board[7] != ' '):

        return True

    elif (board[1] == board[4] and board[1] == board[7] and board[1] != ' '):

        return True

    elif (board[2] == board[5] and board[2] == board[8] and board[2] != ' '):

        return True

    elif (board[3] == board[6] and board[3] == board[9] and board[3] != ' '):

        return True

    elif (board[1] == board[5] and board[1] == board[9] and board[1] != ' '):

        return True

    elif (board[7] == board[5] and board[7] == board[3] and board[7] != ' '):

        return True

    else:

        return False

두 개의 칸이 연결되어 체크되어 있는 모든 경우를 확인하여 만약 있다면 이긴 사람이 있는 것이고 아니라면 아직 승부가 결정나지 않았기 때문에 false를 반환하고 게임을 이어나간다

#무승부 판별

def check\_draw():

    for key in board.keys():

        if (board[key] == ' '):

            return False

    return True

만약 공백이 있다면 계속 승부를 이어나가고 공백이 없는데도 check\_draw함수가 호출되었다면 더 이상 둘 공간이 없기 때문에 True를 반환하여 무승부로 선언한다

#승자 판별

def insertLetter(letter, position):

    if spaceIsFree(position):

        board[position] = letter

        printBoard(board)

        if (check\_draw()):

            print("아쉽지만 무승부입니다!")

            sys.exit()

        if check\_win():

            if letter == 'X':

                print("컴퓨터의 승리입니다 분발하세요!")

                sys.exit()

            else:

                print("축하합니다 당신의 승리입니다!")

                sys.exit()

        return

마지막 차례였던 사람을 letter에 기록하고 만약 무승부였다면 무승부로 출력하고 아니라면

Letter에 저장된 변수에 따라 승자를 판별한다.

이때 판별된 경우 sys.exit()를 이용해 종료해준다.

#순서

while not check\_win():

    player\_move()

    com\_move()

승자가 판별되지 않았다면 플레이어가 반드시 선이므로 플레이어의 좌표 입력을 받는 함수를

먼저 호출하고 그후 컴퓨터의 움직임을 받는 함수를 호출한다.

# 나만의 AI 알고리즘

def minimax(board, depth, isMaximizing):

    if (checkwhichplayerwin(bot)):

        return 1

    elif (checkwhichplayerwin(player)):

        return -1

    elif (check\_draw()):

        return 0

    if (isMaximizing):

        bestscore = -2022

        for key in board.keys():

            if (board[key] == ' '):

                board[key] = bot

                score = minimax(board, depth + 1, False)

                board[key] = ' '

                if (score > bestscore):

                    bestscore = score

        return bestscore

    else:

        bestscore = 2022

        for key in board.keys():

            if (board[key] == ' '):

                board[key] = player

                score = minimax(board, depth + 1, True)

                board[key] = ' '

                if (score < bestscore):

                    bestscore = score

        return bestscore

Minimax 알고리즘을 활용하여 AI를 만들었습니다.

Minimax 알고리즘이란 나올 수 있는 모든 경우의 수를 시각화하여 트리 형태로 구상하고 경기가 끝났을 때 ai가 이겼다면 1점, 졌다면 -1점, 비겼다면 0점을 부여합니다.

모든 경우의 수를 확인한 후 bestscore라는 가장 효율성이 높은 경우의 점수와 비교하여 가장

승률이 높은 수를 선택하고 이를 저장하여 컴퓨터의 다음 수에 이용합니다.

제가 짠 코드의 경우 모든 경우를 탐색해야 한다는 단점이 있지만 만약 알파베타 가지치기라는

것을 이용하여 결과에 영향을 주지 않는 수를 제거한다면 더 빨리 최선의 수를 찾을 수 있습니다

#컴퓨터의 수

def com\_move():

    print("컴퓨터의 차례입니다!")

    bestscore = -2022

    bestmove = 0

    for key in board.keys():

        if (board[key] == ' '):

            board[key] = bot

            score = minimax(board, 0, False)

            board[key] = ' '

            if (score > bestscore):

                bestscore = score

                bestmove = key

    insertLetter(bot, bestmove)

    return

bestscore을 음수로 지정하여 혹시 모를 error를 방지하고 함수 minimax를 호출하여 현재

보드판의 상황에서 놓을 수 있는 모든 수 중 가장 score가 높은 것을 선택하고 가장 좋은 수의

좌표를 bestmove에 저장하여 이를 입력 함수에 넣은 후 return시킨다