코테용 코드 모음집

스택

class Stack:

    def \_\_init\_\_(self):

        self.s = []

    def push(self, item):

        self.s.append(item)

    def pop(self):

        if self.isEmpty() == False:

            return self.s.pop(-1)

        else:

            return None

    def peek(self):

        if self.isEmpty() == False:

            return self.s[-1]

        else:

            return None

    def isEmpty(self):

        if len(self.s) > 0:

            return False

        else:

            return True

    def size(self):

        return len(self.s)

    def print(self):

        print(self.s)

큐

class Queue:

    def \_\_init\_\_(self):

        self.q = []

    def enQueue(self, item):

        self.q.append(item)

    def deQueue(self):

        if self.isEmpty() == False:

            return self.q.pop(0)

    def size(self):

        return len(self.q)

    def isEmpty(self):

        if len(self.q) > 0:

            return False

        else:

            return True

    def peek(self):

        if self.isEmpty() == False:

            return self.q[0]

    def delete(self, item):

        if item in self.q:

            self.q.remove(item)

        else:

            print("해당 아이템이 존재하지 않습니다.")

원형 큐

Class C\_queue:

Def \_\_init\_\_(self,):

Int rear = 0

Int front = 0

Int maxsize = 0

Int queue = []

Def isempty():

If rear == front:

Return ture

Else:

Return false

Def isfull():

If (rear + 1) % maxsize == front:

Return true

Else :

Return false

Def enqueue( n):

If (isfull)

Print(queue is full)

Else

Rear = (rear+1) % maxsize

Queue[rear].append(n)

Def dequeue()

If isempty():

Return -1

Else

Front = (front+1)%maxsize

Return queue[front]

문자열 고유한지 확인 – 아스키코드일 경우

Def is unique(str):

Str\_len = len(str)

If str\_len > 128:

Return true

Check = [FALSE for I in range(0, 127)]

For I in range(0, str\_len):

Val = ord(str[i])

If Check[val] = true

Return false

Else :

Check[val] = true

Return ture

문자열이 고유한지 확인 – 유니코드의 경우

Def isunique(str):

D = {}

For I in range(0, len(str)):

C = ord(str[i])

If (len(D) != 0 and D.has\_key(C):

Return false

Else :

D[C] = True

Return true

퀵소트

def quicsort(L):

  if len(L) <= 1:

    return L

    p = L[len(L) // 2]

    lesser\_L, equal\_L, greater\_L = [], [], []

    for num in L:

      if num < p:

        leser\_arr.append(num)

      elif num > p:

        greater\_arr.append(num)

      else :

        equal\_arr.append(num)

    return quicsort(lesser\_L) + equal\_L + quicsort(greater\_L)

회문 판독

#회문여부 판독

def ispermutationofpalindrome(s):

  for i in range(len(word) // 2):      # 0부터 문자열 길이의 절반만큼 반복

    if word[i] != word[-1 - i]:      # 왼쪽 문자와 오른쪽 문자를 비교하여 문자가 다르면

        is\_palindrome = False        # 회문이 아님

        break

더 간단하게

print(word == word[::-1])

n-gram

text = 'this is python script'

words = text.split()                 # 공백을 기준으로 문자열을 분리하여 리스트로 만듦

for i in range(len(words) - 1):      # 2-gram이므로 리스트의 마지막에서 요소 한 개 앞까지만 반복함

    print(words[i], words[i + 1])    # 현재 문자열과 그다음 문자열 출력

조합, 순열

def combinations(arr,r):

for i in range(len(arr)): // 함수에서 지금할 일

if r == 1: // 종료조건

yield [arr[i]]

else:

for next in combinations(arr[i+1:],r-1): // 다음에 할 일

yield [arr[i]] + next

중복조합

def combinations\_with\_replacement(arr,r):

for i in range(len(arr)):

if r == 1:

yield [arr[i]]

else:

for next in combinations\_with\_replacement(arr[i:],r-1):

yield [arr[i]] + next

중복순열

def product(arr,r):

for i in range(len(arr)):

if r == 1:

yield [arr[i]]

else:

for next in product(arr,r-1):

yield [arr[i]] + next