



배열, 리스트, 스택, 큐

:: 배열(array) vs 리스트(list)

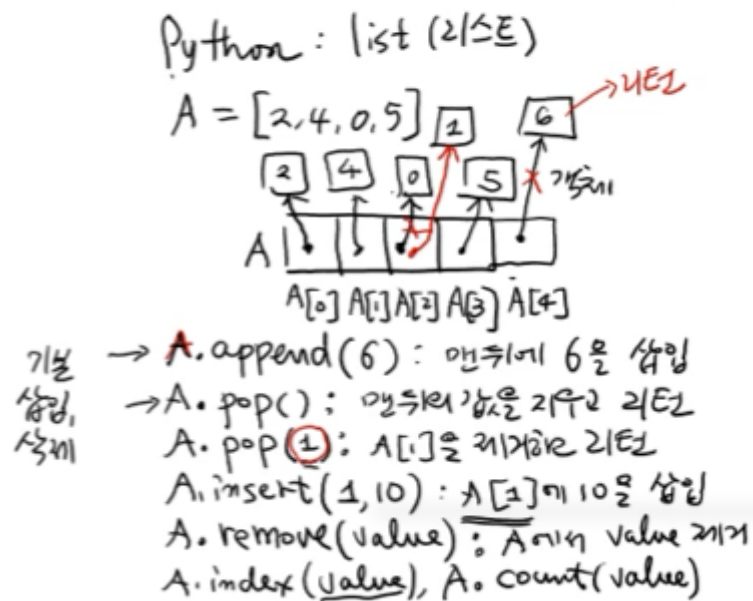
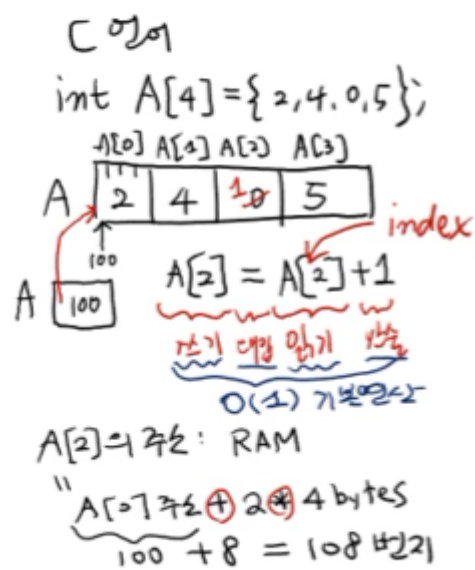
- 가장 기본적인 **순차적인** 자료구조 [매우 기본, 중요]
- 리스트 같은 경우 python

:: 리스트(list) - Python

- Python 리스트는 C언어 배열과 다르게 리스트 셀에는 데이터가 아닌 주소가 저장된다. (모든 셀의 크기가 같기 때문에 index에 의해 O(1) 시간 접근 가능)

배열(array) vs. 리스트(list) ← python

- 가장 기본적인 순차적인(sequential) 자료구조 [매우 기본, 중요]

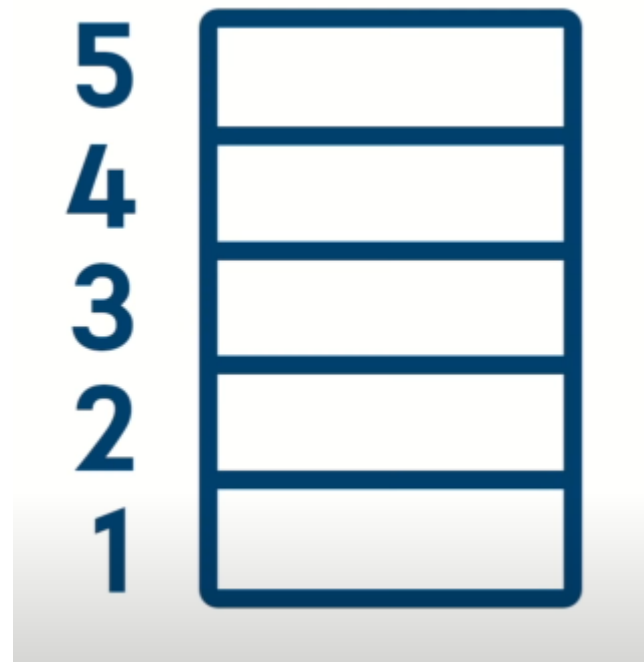


:: 순차적 자료구조

- 배열, 리스트
 - index로 임의의 원소를 접근
 - 삽입(append, insert)
 - 삭제(pop, remove)
- Stack, Queue, dequeue
 - 제한된 접근(삽입, 삭제) 허용

stack : LIFO(Last in First out) - (팬케이크)

- '스택'은 배열이 수직으로 쌓임
- 요소를 추가하거나 삭제 할때 첫번째부터 함
- 마지막으로 쌓아올린 팬케이크가 가장 먼저 나감



:: 스택 (Stack)

삽입 : push / 삭제 : pop

- push, pop, top, len, isEmpty 함수 모두 $O(1)$ 시간 연산이다.

```
# stack_queue.py 에 저장
class Stack:
    def __init__(self):
        self.items = [] # 데이터 저장을 위한 리스트 준비
    def push(self, val):
        self.items.append(val)
    def pop(self):
        try: # pop할 아이템이 없으면
            return self.items.pop()
        except IndexError: # IndexError 발생
            print("Stack is empty")
    def top(self):
        try:
            return self.items[-1]
        except IndexError:
            print("Stack is empty")
    def __len__(self): # len()로 호출하면 stack의 item 수 반환
        return len(self.items)
    def isEmpty(self):
        return self.__len__() == 0

# for test
S = Stack()
S.push(10)
S.push(2)
print(S.top())
print(S.pop())
print(len(S))
print(S.isEmpty())
```

스택이므로 처음 push 10, 그 다음 2를 푸쉬하면

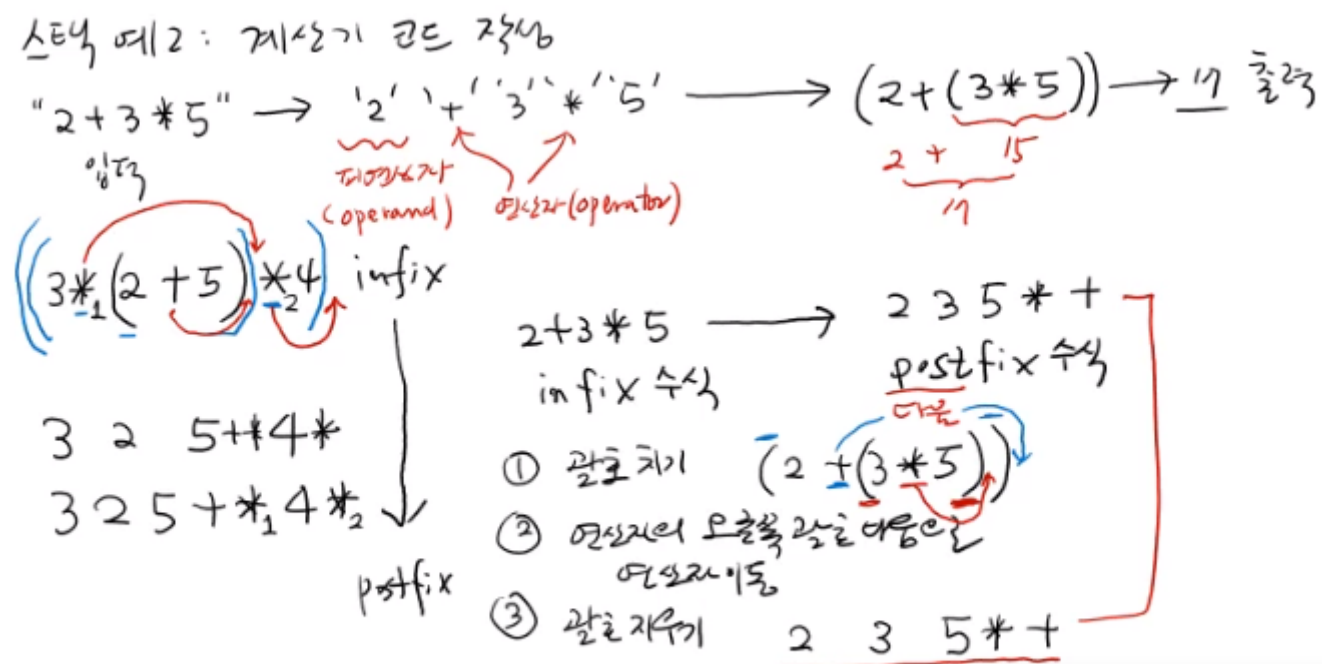
수직으로 제일 아래에 10 그 위에 2가 들어가게 된다.

2 이 모양이므로 top()함수를 통해 제일 위에 있는 값은 2

10 pop을 하면 제일 위에 있는 값이 삭제되므로 삭제되는 값은 2

len함수를 통해 길이를 출력하는데 pop으로 인해 2가 지워졌으므로 10만 남아있어서 len 길이는 1, 그리고 10이 남아있으므로 isEmpty는 False

스택 활용 - 계산기 코드



infix 수식을 postfix 수식으로 바꾸는 방법

$A + B * C \rightarrow A B C * +$

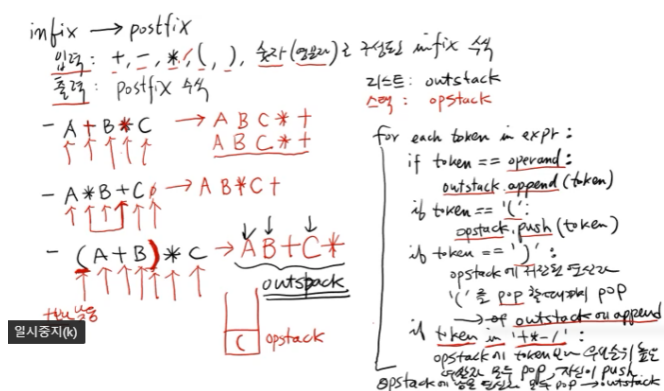
$A * B + C \rightarrow A B * C +$

$(A + B) * C \rightarrow A B + C *$

1. 피연산자의 순서는 그대로
2. 우선순위가 높은 연산자는 먼저 스택에서 나옴.

리스트: outstack

스택: opstack

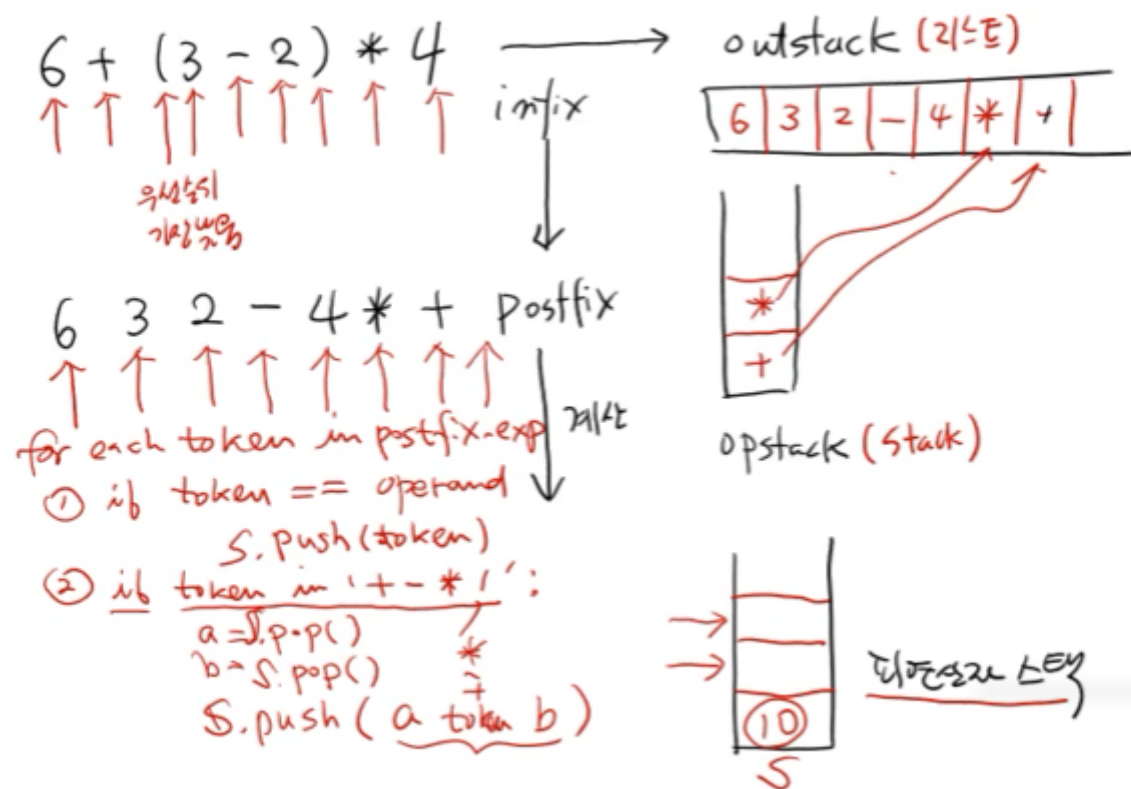


```

for each token in expr:
    if token == operand:
        outstack.append(token)
    if token == '(':
        opstack.push(token)
    if token == ')':
        opstack에 저장된 연산자
        '('를 pop할 때까지 pop
        -> outstack에 append
    if token in '+*-/':
        opstack에 token보다 우선순위 높은 연산자 모두 pop 후
        자신이 opstack에 push

- opstack에 남은 연산자 모두 pop -> outstack에 append
  
```

operand : 피연산자



$6 + (3 - 2) * 4$ 를 infix에서 postfix로 바꾸는 법까지 할줄 알 것이다.

여기서 계산을 하는 방법을 보자.

1. 피연산자스택을 하나 새로 만들어준다.
2. 그리고 만약 token이 operand(피연산자)일 경우 스택에 push해준다.
이럴경우 처음 6, 3, 2가 push가 쪽 들어갈 것이다.
3. 만약 token이 연산자'+-*/'일 경우
a와 b 두 개를 pop해준다.
이것이 무엇이나
현재 피연자 스택에 들어가있는 값은 6, 3, 2이다.
이 다음은 '-'로 연산자이다.
-가 들어올경우 a, b.
즉, 3, 2가 pop된다. 3과 2가 마이너스(-)로 pop이 되어 1이 된다
4. 그 다음 그 값을 push한다.
피연산자 스택에는 처음 6과 3, 2가 pop된 1이 남는다.

이것을 반복하면 그 다음 값 4가 피연산자 스택에 들어온다.

그러면 6, 1, 4가 된다.

여기서 * 연산자가 들어오면서 1과 4가 * 연산자로 pop되며 4가 된다.

그리고 마지막에 + 연산자가 들어오면서 남아있는 6과 4가 +로 pop되어 10이 나오고 계산과 일치하게 된다.

스택활용예1: 괄호짝맞추기

```
# stack_queue.py 에 저장
class Stack:
    def __init__(self):
        self.items = [] # 데이터 저장을 위한 리스트 준비
    def push(self, val):
        self.items.append(val)
    def pop(self):
        try: # pop할 아이템이 없으면
            return self.items.pop()
        except IndexError: # IndexError 발생
            print("Stack is empty")
    def top(self):
        try:
            return self.items[-1]
        except IndexError:
            print("Stack is empty")
```

```

def __len__(self): # len()로 호출하면 stack의 item 수 반환
    return len(self.items)
def isEmpty(self):
    return self.__len__() == 0

# pseudo code
def parChecker(parSeq):
    S = Stack()
    for symbol in parSeq:
        if symbol == "(":
            S.push(symbol)
        else:
            if S.isEmpty():
                return False
            else:
                S.pop()

    if S.isEmpty():
        return True
    else:
        return False
S = input()
ispar = parChecker(S)
print(ispar)

```

스택활용예2: Infix-to-Postfix

```

'''
Infix to postfix
'''
class Stack:
    def __init__(self):
        self.items = []

    def push(self, val):
        self.items.append(val)

    def pop(self):
        try:
            return self.items.pop()
        except IndexError:
            print("Stack is empty")

    def top(self):
        try:
            return self.items[-1]
        except IndexError:
            print("Stack is empty")

    def __len__(self):
        return len(self.items)

    def isEmpty(self):
        return self.__len__() == 0

def infix_to_postfix(infix):
    opstack = Stack()
    outstack = []
    token_list = infix.split(' ')
    prec = {'(':1, '+':2, '-':2, '*':3, '/':3, '^':4}

    for token in token_list:
        if token == '(':
            opstack.push(token)

        elif token == ')':
            while True:
                infix = opstack.pop()
                if infix == '(':
                    break
                outstack.append(infix)

        elif token in '+-/*^':
            while(not opstack.isEmpty()) and (prec[opstack.top()] >= prec[token]):
                outstack.append(opstack.pop())
            opstack.push(token)

        else: # operand일 때
            outstack.append(token)

    while not(opstack.isEmpty()):

```

```

        outstack.append(opstack.pop())
    return " ".join(outstack)

infix_expr = input()
postfix_expr = infix_to_postfix(infix_expr)
print(postfix_expr)

```

스택활용예3: Postfix 계산

```

class Stack:
    def __init__(self):
        self.items = [] # 데이터 저장을 위한 리스트 준비
    def push(self, val):
        self.items.append(val)
    def pop(self):
        try: # pop할 아이템이 없으면
            return self.items.pop()
        except IndexError: # IndexError 발생
            print("Stack is empty")
    def top(self):
        try:
            return self.items[-1]
        except IndexError:
            print("Stack is empty")
    def __len__(self): # len()로 호출하면 stack의 item 수 반환
        return len(self.items)
    def isEmpty(self):
        return self.__len__() == 0

def compute_postfix(postfix):
    operand = Stack()
    token_list = postfix.split()
    operators = ['*', '/', '+', '-', '^'] # operator 리스트 만들.

    for token in token_list:
        if token not in operators: # 연산자가 아닐때. 즉 숫자일때 (0123456789)
            operand.push(int(token)) # int형변환

        else: #연산자일 경우
            n1 = operand.pop() # 해당 연산자 앞의 두 개의 피연산자에 대한 연산이므로, 앞서 스택에 들어갔던 피연산자 두 개를 pop 하여 다시 꺼내 연산을 처리
            n2 = operand.pop()
            result = calc(token, n2, n1)
            operand.push(result) # 그 결과값을 다시 스택에 넣는다.
    return operand.pop()

def calc(i, op1, op2): #계산 함수
    if i == '+':
        return op1 + op2
    elif i == '-':
        return op1 - op2
    elif i == '*':
        return op1 * op2
    elif i == '/':
        return op1 / op2
    else:
        return op1 ** op2

postfix_eval = input()
print("%.4f" %(compute_postfix(postfix_eval)))

```

스택활용예4: 윈도우계산기완성

```

from tkinter import Tk, Label, Button, Entry, StringVar
from functools import partial

class Stack:
    def __init__(self):
        self.items = []

    def push(self, val):
        self.items.append(val)

```

```

def pop(self):
    try:
        return self.items.pop()
    except IndexError:
        print("Stack is empty")

def top(self):
    try:
        return self.items[-1]
    except IndexError:
        print("Stack is empty")

def __len__(self):
    return len(self.items)

def isEmpty(self):
    return self.__len__() == 0

def infix_to_postfix(infix):
    opstack = Stack()
    outstack = []
    token_list = infix.split(' ')
    prec = {'(':1, '+':2, '-':2, '*':3, '/':3, '^':4}

    for token in token_list:
        if token == '(':
            opstack.push(token)

        elif token == ')':
            while True:
                infix = opstack.pop()
                if infix == '(':
                    break
            outstack.append(infix)

        elif token in '+-/*^':
            while(not opstack.isEmpty()) and (prec[opstack.top()] >= prec[token]):
                outstack.append(opstack.pop())
            opstack.push(token)

        else: # operand일 때
            outstack.append(token)

    while not(opstack.isEmpty()):
        outstack.append(opstack.pop())
    return " ".join(outstack)

def compute_postfix(postfix):
    operand = Stack()
    token_list = postfix.split()
    operators = ['*', '/', '+', '-', '^'] # operator 리스트 만들.

    for token in token_list:
        if token not in operators: # 연산자가 아닐때. 즉 숫자일때 (0123456789)
            operand.push(int(token)) # int형변환

        else: #연산자일 경우
            n1 = operand.pop() # 해당 연산자 앞의 두 개의 피연산자에 대한 연산이므로, 앞서 스택에 들어갔던 피연산자 두 개를 pop 하여 다시 꺼내 연산을 처리
            n2 = operand.pop()
            result = calc(token, n2, n1)
            operand.push(result) # 그 결과값을 다시 스택에 넣는다.
    return operand.pop()

def do_something():
    value = compute_postfix(infix_to_postfix(expr.get()))
    total.set("{0:.4f}".format(value))
    return

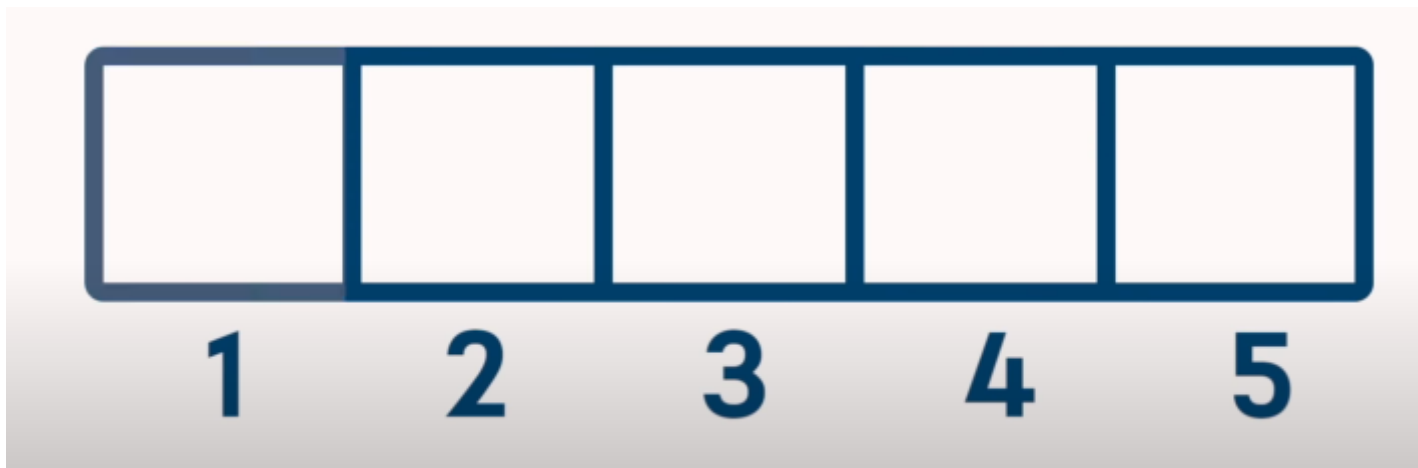
root = Tk()
root.title("My Calculator")
expr = StringVar()
title_label = Label(root, text="My Calcualtor").grid(row=0, columnspan=2)
input_exam = Label(root, text="Space between terms: ( 3 + 2 ) * 8").grid(row=1, columnspan=2)
exp_entry = Entry(root, textvariable=expr).grid(row=2, column=0)
total_label = Label(root, text="TOTAL").grid(row=3, column=0)
total = StringVar()
total.set('0')
value_label = Label(root, textvariable=total, width=20).grid(row=3, column=1)
equal_btn = Button(root, text=' = ', width=20, command=do_something).grid(row=2, column=1)
root.mainloop()
root.destroy()

```

:: 큐(queue)

- 큐는 enqueue(스택의 push 연산에 대응)로 값을 삽입, dequeue(스택의 pop 연산에 대응)로 가장 처음에 저장된 값을 삭제하는 기본적인 자료구조이다.

queue : FIFO(First in First out) - (버스 줄)



- 줄을 제일 먼저 기다린 사람이 제일 먼저 탑승
가장 늦게 온 사람이 가장 마지막에 탑승
가장 먼저 '큐에' 입장한 요소가 가장 먼저 '큐'에서 나감
새로운 '큐' 맨 뒤에 추가 되고 맨 앞에 있는 요소만 읽거나 삭제될 수 있음.

3. linked list(연결 리스트)

- index로 접근할 수 x

```
# stack_queue.py 에 저장
class Queue:
    def __init__(self):
        self.items = [] # 데이터 저장을 위한 리스트 준비
    def enqueue(self, val):
        self.items.append(val)
    def dequeue(self):
        try: # pop할 아이템이 없으면
            return self.items.front # 어떤 함수를 호출해야 할까?
        except IndexError: # IndexError 발생
            print("Queue is empty")
    def front(self):
        try:
            return self.items[_____] # 어떤 값이 와야할까?
        except IndexError:
            print("Queue is empty")
    def __len__(self): # len()로 호출하면 stack의 item 수 반환
        return len(self.items)
    def isEmpty(self):
        return len(self)
```

- enqueue, dequeue, front, len, isEmpty 함수 모두 $O(1)$ 시간 연산
 - enqueue, dequeue, front, len, isEmpty 함수 모두 $O(1)$ 시간에 가능한가?
 - 어떤 연산이 상수시간에 수행되지 않는가? 최악의 경우의 수행시간과 그 이유는 무엇인가?
 - dequeue에서 가장 왼쪽 값 `items[0]`이 삭제되면서 그 오른쪽의 값들이 한 칸씩 왼쪽으로 이동하는 시간이 필요하기 때문
- 모든 연산을 상수시간에 수행되도록 하려면?
 - dequeue에서의 값이 이동하지 않고 현재 큐의 가장 왼쪽의 값이 저장된 `index`를 직접 관리한다.

- 즉, 현재 시점에서 가장 오래된에 삽입된 값의 index를 기억하고, dequeue되면 index 값을 1 증가시킨다.

- def deque에서 # 어떤 함수를 호출해야 할까?

return None

```
def deque(self):
    if (self.front_index == len(self.item)): #큐가 비어있음
        print("Queue is empty!")
        return None
    else:
        x = self.item[self.front_index]
        self.front_index += 1
        return x
```

- def front에서 # 어떤 값이 와야할까?

```
def front(self):
    return self.item[self.front_index]
```

class Queue:

```
class Queue:
    def __init__(self):
        self.item = []
        self.front_index = 0

    def enqueue(self, val):
        self.item.append(val)

    def dequeue(self):
        if (self.front_index == len(self.item)): # 큐가 비어있음
            print("Queue is empty!")
            return None
        else:
            x = self.item[self.front_index]
            self.front_index += 1
            return x

    def isEmpty(self):
        if (self.front_index == len(self.item)):
            print("Queue is empty!")
        else:
            return len(self.item) - self.front_index

    def front(self): # 가장 왼쪽에 저장된 값을 삭제하지 않고 리턴
        return self.item[self.front_index]

    def len(self):
        return len(self.item) - self.front_index
```

:: front_index를 마련해 dequeue시간을 상수시간으로 관리하는 방법

```
# stack_queue.py 에 저장
class Queue:
    def __init__(self):
        self.items = [] # 데이터 저장을 위한 리스트 준비
        self.front_index = 0 # 다음 dequeue될 값의 인덱스 저장

    def enqueue(self, val):
        self.items.append(val)

    def dequeue(self):
        if len(self.items) == 0 or self.front_index == len(self.items):
            print("Queue is empty")
        else:
            x = self.items[self.front_index]
            self.front_index += 1
            return x

    def front(self):
```

```

    if len(self.items) == 0 or self.front_index == len(self.items):
        print("Queue is empty")
    else:
        return self.items[self.front_index]

def __len__(self): # len()로 호출하면 stack의 item 수 반환
    return len(self.items)-self.front_index # why?

def isEmpty(self):
    return len(self)

Q = Queue()
Q.enqueue(10)
Q.enqueue(4)
print(Q.dequeue())
print(Q.front())
print(Q.dequeue())
print(Q.front())

```

실행 결과:

10

4

4

Queue is empty

None

:: 디큐(dequeue)

1. 왼쪽과 오른쪽에서 모두 삽입과 삭제가 가능한 큐 - 두 가지 버전의 push와 pop을 연산을 구현하면 되고, 나머지 연산은 Stack, Queue 클래스와 유사하게 구현한다.
2. Python에서는 collections라는 모듈에 deque란 클래스로 dequeue가 이미 구현됨
 - a. 오른쪽 push = append, 왼쪽 push = appendleft
 - b. 오른쪽 pop = pop, 왼쪽 pop = popleft

- **Deque의 사용 예:** Palindrome 검사 코드

- Palindrome은 왼쪽부터 읽어도 오른쪽부터 읽어도 같은 문자열을 말한다.
- 방법 1: 문자열 s와 s를 reverse한 문자열이 같다면 palindrome임
 - reversed(s)는 문자열 s를 거꾸로한 iterable 객체임
 - >>> s == ".join(reversed(s))
- 방법 2: collections 패키지의 deque 모듈을 사용함
 - deque에 문자열을 저장한 후 양쪽에서 하나씩 빼면서 (pop과 popleft 이용) 같은지를 비교하는 것을 반복
 - 다르다면 palindrome이 아님
 - Pseudo 코드:

```

from collections import deque
def check_palindrome(s):
    dq = deque(s)
    palindrome = True
    while len(dq) > 1:
        if dq.popleft() != dq.pop():
            palindrome = False
    return palindrome

```

Palindrome Check

```

class deque:
    def __init__(self):
        self.items = []

    def __init__(self, s):
        self.items = []

    def append(self, c):
        self.items.append(c)

    def appendleft(self, c):
        self.items.appendleft(c)

    def pop(self):
        self.items.pop()

    def popleft(self):
        self.items.popleft()

    def __len__(self):
        return len(self.items)

    def right(self):
        return self.items[0]

    def left(self):
        return self.items[-1]

from collections import deque

def check_palindrome(s):
    dq = deque(s)
    palindrome = True

    while len(dq) > 1:
        if dq.popleft() != dq.pop():
            palindrome = False

    return palindrome

n = input()
print(check_palindrome(n))

```