

EX - 09 STACKS AND QUEUES

T SADAKOPA RAMAKRISHNAN | 3122225002109 | IT - B

stack.py:

class Stack:

def __init__(self):

'''

Constructor to initialize a list

'''

self.item = []

def isempty(self):

'''

Method to check if stack is empty or not.

'''

return len(self.item) == 0

def push(self, ele):

'''

Method to append elements to a stack.

'''

self.item.append(ele)

def pop(self):

'''

Method to remove elements from stack.

'''

return self.item.pop()

def __len__(self):

'''

Method to return length of the stack.

'''

return len(self.item)

```

def __str__(self):
    """
        Overriding to print string value of the object.
    """
    return str(self.item)

def topelement(self):
    """
        Method that returns the top value of the stack.
    """
    return self.item[-1]

```

queue.py:

```

class Queue:
    def __init__(self):
        """
            Constructor to initialise variables.
        """
        self.item = []
        self.front = 0
        self.rear = 0

    def isempty(self):
        """
            Method to check if Queue is empty
        """
        if self.front == self.rear:
            return True
        else:
            return False

    def enqueue(self, ele):
        """
            Method to add elements to the queue.
        """
        self.item.insert(self.rear, ele)

```

```

        self.rear += 1

def dequeue(self):
    """
        Methods to remove elements from the queue.
    """
    return self.item.pop(0)

def __len__(self):
    """
        Method to return length of the queue.
    """
    return self.rear - self.front

def __str__(self):
    """
        Overiding to return the string value of the object
    """
    return str(self.item)

```

circularqueue.py:

```
import ctypes
```

```

class CircularQueue:
    def __init__(self, cap):
        """
            Constructor to initialise variables
        """
        self.cap = cap
        self.item = self.makearray(cap)
        self.rear = self.front = 0

    def makearray(self, cap):
        """
            Method to create compact arrays
        """
        temp = (cap * ctypes.py_object)()

```

```

    return temp

def next(self, pos):
    """
        Method to go next in the data structure.
    """
    return ( pos + 1 ) % self.cap

def isEmpty(self):
    """
        Method to check if circular queue is empty.
    """
    return (self.front == self.rear)

def isFull(self):
    """
        Method to check if circular queue is full.
    """
    return ((self.rear + 1) % self.cap == self.front)

def enqueue(self, ele):
    """
        Method to add elements to the circular queue.
    """
    if (self.isFull()):
        print("Queue is Full. No more orders accepted.")
    else:
        self.item[self.rear] = ele
        self.rear = self.next(self.rear)
        print("Order placed Successfully")

def dequeue(self):
    """
        Method to remove elements from circular queue.
    """
    if self.isEmpty():
        print("No more orders to serve")

```

```

    else:
        order = self.item[self.front]
        self.front = self.next(self.front)
        print("Serving order", order)

class Full(Exception):
    pass

class Empty(Exception):
    pass

```

Q1) Design and implement stack and queue data structure as wrapper around the Python List. Using stack and queue check whether a given number is a palindrome

Code:

```

# -*- coding: utf-8 -*-
"""

```

```

This module checks if the entered string is a palindrome
or not using both stack and queue. This is a part
of the exercises given under the course UIT2201 (Programming
and Data Structures) .

```

```

In this source code I've executed my own logic and may
contain bugs.

```

```

The source code has followed good coding practices.

```

```

Your comments and suggestions are welcome.

```

```

Created on Wed May 31 2023

```

```

Revised on Tue June 6 2023

```

```

Original Author: T. Sadakopa Ramakrishnan
<sadakopa2210221@ssn.edu.in>
"""

```

```

from stack import Stack
from queue import Queue

def ispalindrome():
    S = Stack()
    Q = Queue()

    num = input("Enter something to check if its palindrome: ")

    for i in range(len(num)):
        S.push(num[i])
        Q.enqueue(num[i])

    while True:
        if S.pop() != Q.dequeue():
            print("Not a palindrome")
            break
        else:
            print("Palindrome")
            break

ispalindrome()

```

Output:

```

Enter something to check if its palindrome: 121
Palindrome

```

```

Enter something to check if its palindrome: sadas
Palindrome

```

Q2) Design and implement data structure to maintain two stacks in a single array. All the basic stack operations should include an argument to select one of the stacks. For example, 'push(0, item)' should push item into the first stack, while 'push(1, item)' should push item into the second stack. Your stack methods should not raise stack full exception, unless every array cell is used.

Code:

```
# -*- coding: utf-8 -*-  
"""
```

This module provides a data structure to maintain two stacks in a single array . This is a part of the exercises given under the course UIT2201 (Programming and Data Structures) .

In this source code I've executed my own logic and may contain bugs.

The source code has followed good coding practices.

Your comments and suggestions are welcome.

Created on Wed May 31 2023

Revised on Tue June 6 2023

Original Author: T. Sadakopa Ramakrishnan
<sadakopa2210221@ssn.edu.in>
"""

```
import ctypes
```

```
class Stack:
```

```
    def __init__(self, cap):
```

```
        '''
```

```
            Constructor to initialise variables.
```

```
        '''
```

```
        self.capacity = cap
```

```
        self.top1 = 0
```

```
        self.top2 = self.capacity - 1
```

```
        self.item = self.makearray(self.capacity)
```

```
    def makearray(self, cap):
```

```
        '''
```

```

        Method to create compact arrays
'''
temp = (cap * ctypes.py_object)()
return temp

def isempty(self):
'''
        Method to check if the data structure is empty
or not.
'''
    if self.top1 == 0 & self.top2 == self.capacity - 1:
        return True
    else:
        return False

def push(self,n, ele):
'''
        Method to push into stack depending on the
position 1 or 0.
'''
    if n == 0:
        self.item[self.top1] = ele
        self.top1 += 1
    elif n == 1:
        self.item[self.top2] = ele
        self.top2 -= 1
    else:
        raise ValueError("ENter only 0 or 1")

def pop(self,n):
'''
        Method to pop from stack depending the position
1 or 0.
'''
    if n == 0:
        self.top1 -= 1
        return self.item[self.top1]

```



```

        elif n == 1:
            self.top2 += 1
            return self.item[self.top2]
        else:
            raise ValueError("ENter only 0 or 1")

    def __str__(self):
        '''
            Overriding to print string value of the object
        '''
        return str([self.item[i] for i in range(self.top1)])
\
        +str([self.item[i] for i in range(self.capacity-1,
self.top2, -1)])

    def len(self,n):
        '''
            Returns the length of the data structure
        '''
        if n == 0:
            print(self.top1)
        elif n == 1:
            print(self.capacity - self.top2-1)
        else:
            raise ValueError("enter 1 or 0")

    def isfull(self):
        '''
            Method to check if the data structure is full or
not.
        '''
        if self.top1 > self.top2:
            return True
        else:
            return False

if __name__ == "__main__":
    S = Stack(5)

```

```

S.push(0, 1)
S.push(0, 3)
S.push(0, 4)

S.push(1, 5)
S.push(1, 4)

S.len(1)
S.len(0)

print(S)

print(S.isfull())

```

Output:

```

2
3
[1, 3, 4][5, 4]
True

```

Q3)A food delivery system accepts a maximum of M orders. Orders are served in first come first basis. Orders once placed cannot be cancelled. Write a Python code to simulate the system using circular queues.

Code:

```

# -*- coding: utf-8 -*-
"""

```

```

This module provides a food delivery system accepts a
maximum

```

of M orders and uses circular queue concept.This is a part of the exercises given under the course UIT2201 (Programming and Data Structures).

In this source code I've executed my own logic and may contain bugs.

The source code has followed good coding practices.

Your comments and suggestions are welcome.

Created on Wed May 31 2023

Revised on Tue June 6 2023

Original Author: T. Sadakopa Ramakrishnan

<sadakopa2210221@ssn.edu.in>

"""

```
from circularqueue import CircularQueue
```

```
number_of_orders = int(input("Enter the number of orders:"))
```

```
food_delivery = CircularQueue(number_of_orders)
```

```
for i in range(number_of_orders):  
    food_delivery.enqueue(f"Order {i+1}")
```

```
for i in range(number_of_orders):  
    food_delivery.dequeue()
```

Output:

```
Enter the number of orders: 5
Order placed Successfully
Order placed Successfully
Order placed Successfully
Order placed Successfully
Queue is Full. No more orders accepted.
Serving order Order 1
Serving order Order 2
Serving order Order 3
Serving order Order 4
No more orders to serve
```

Q4) Design and implement data structure for a queue like abstraction, referred to as PQueue, that internally maintains two arrays — one for a high priority queue and the other for the low priority queue. Enqueue operation will mention the priority of the 'item' to be added to the PQueue — for example, 'enqueue(0, item)' will add the 'item' to the high priority queue, while 'enqueue(1, item)' will add it to the low priority queue. 'dequeue()' operation will dequeue from the high priority queue. If the high priority queue is empty, then first item from the low priority queue will be dequeued.

Code:

```
class PQueue:
    def __init__(self):
        '''
            Constructor to initialise two lists.
        '''
        self.hpq = []
        self.lpq = []

    def enqueue(self, priority, ele):
        '''
            Adding elements to the 2 queues depending on the
            priority value.
```

```

'''
if priority == 0:
    self.hpq.append(ele)
elif priority == 1:
    self.lpq.append(ele)
else:
    raise ValueError("Invalid Priority.")

def dequeue(self):
    '''
        Deleting elements from the queue depending on
the priority value.
    '''
    if self.hpq:
        return self.hpq.pop(0)
    elif self.lpq:
        return self.lpq.pop(0)
    else:
        raise IndexError("Pqueue is empty. Can't dequeue
further.")

def isempty(self):
    '''
        Checking if the high priority queue and low
priority queue are empty
    '''
    return len(self.hpq) == 0 and len(self.lpq) == 0

if __name__ == "__main__":
    pq = PQueue()

    pq.enqueue(0, "Sada")
    pq.enqueue(1, "Kopa")

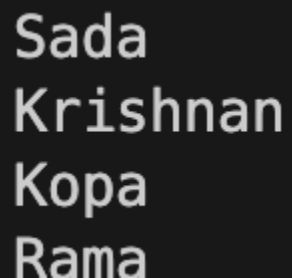
    print(pq.dequeue())

    pq.enqueue(1, "Rama")
    pq.enqueue(0, "Krishnan")

```

```
print(pq.dequeue())  
print(pq.dequeue())  
print(pq.dequeue())
```

Output:



```
Sada  
Krishnan  
Kopa  
Rama
```