Data Structures and Algorithms – Assignment 2



Name: Ashish Prem Roll No: AM.EN.U4CSE19309

**Stack & Queue**

1. Write a Python program to implement stack and its operations using array (use python list).

from sys import maxsize

def createStack():

stack = []

return stack

def isEmpty(stack):

return len(stack) == 0

def push(stack, item):

stack.append(item)

print(item + " pushed to stack ")

def pop(stack):

if (isEmpty(stack)):

return str(-maxsize - 1)

return stack.pop()

def peek(stack):

if (isEmpty(stack)):

return str(-maxsize - 1)

return stack[len(stack) - 1]

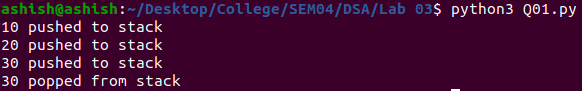
stack = createStack()

push(stack, str(10))

push(stack, str(20))

push(stack, str(30))

print(pop(stack) + " popped from stack")



1. Write a Python program to implement stack and its operations using linked list.

stack = []

def push():

element = input("Enter the element: ")

stack.append(element)

print(stack)

def pop\_element():

if not stack:

print("stack is empty")

else:

e = stack.pop()

print("removed element:",e)

print(stack)

while True:

print("Select the operation 1.push 2.pop 3.quit")

choice = int(input())

if choice == 1:

push()

elif choice == 2:

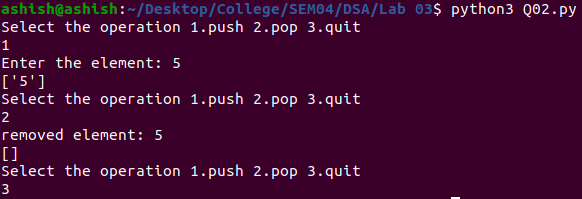
pop\_element()

elif choice == 3:

break

else:

print("Enter the correct option")



1. Write a Python program to implement queue and its operations using array (use python list).

class Queue:

def \_\_init\_\_(self, capacity):

*self*.front = *self*.size = 0

*self*.rear = capacity - 1

*self*.Q = [None] \* capacity

*self*.capacity = capacity

def isFull(self):

return *self*.size == *self*.capacity

def isEmpty(self):

return *self*.size == 0

def EnQueue(self, item):

if *self*.isFull():

print("Full")

return

*self*.rear = (*self*.rear + 1) % (*self*.capacity)

*self*.Q[*self*.rear] = item

*self*.size = *self*.size + 1

print("% s enqueued to queue" % str(item))

def DeQueue(self):

if *self*.isEmpty():

print("Empty")

return

print("% s dequeued from queue" % str(*self*.Q[*self*.front]))

*self*.front = (*self*.front + 1) % (*self*.capacity)

*self*.size = *self*.size - 1

def que\_front(self):

if *self*.isEmpty():

print("Queue is empty")

print("Front item is", *self*.Q[*self*.front])

def que\_rear(self):

if *self*.isEmpty():

print("Queue is empty")

print("Rear item is", *self*.Q[*self*.rear])

if \_\_name\_\_ == '\_\_main\_\_':

queue = Queue(60)

queue.EnQueue(20)

queue.EnQueue(40)

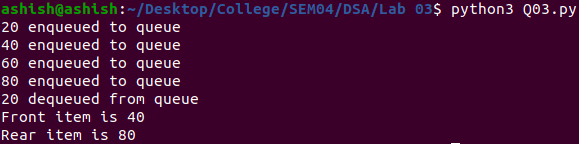
queue.EnQueue(60)

queue.EnQueue(80)

queue.DeQueue()

queue.que\_front()

queue.que\_rear()



1. Write a Python program to implement queue and its operations using linked list.

queue = []

def enqueue():

element = input("Enter the element: ")

queue.append(element)

print(queue)

def dequeue():

if not queue:

print("queue is empty")

else:

e = queue.pop()

print("removed element:", e)

print(queue)

while True:

print("Select the operation 1.push 2.pop 3.quit 4.show")

choice = int(input())

if choice == 1:

enqueue()

elif choice == 2:

dequeue()

elif choice == 3:

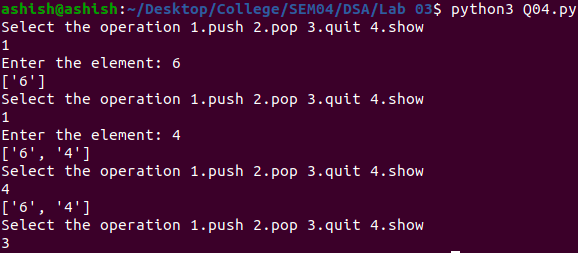
break

elif choice == 4:

print(queue)

else:

print("Enter the correct option")



1. Write a Python program to implement circular queue.

class CircularQueue:

def \_\_init\_\_(self, size):

*self*.size = size

*self*.queue = [None] \* size

*self*.front = *self*.rear = -1

def enqueue(self, item):

if ((*self*.rear + 1) % *self*.size == *self*.front):

print("Queue Is Full\n")

elif (*self*.front == -1):

*self*.front = *self*.rear = 0

*self*.queue[*self*.rear] = item

else:

*self*.rear = (*self*.rear + 1) % *self*.size

*self*.queue[*self*.rear] = item

def dequeue(self):

if (*self*.front == -1):

print("Queue Is Empty\n")

elif (*self*.front == *self*.rear):

value = *self*.queue[*self*.front]

*self*.front = *self*.rear = -1

return value

else:

value = *self*.queue[*self*.front]

*self*.front = (*self*.front + 1) % *self*.size

return value

def print(self):

if (*self*.front == -1):

print("Queue Empty")

elif (*self*.rear > *self*.front):

for i in range(*self*.front, *self*.rear + 1):

print(*self*.queue[i], end=" ")

print()

else:

for i in range(0, *self*.rear + 1):

print(*self*.queue[i], end=" ")

for i in range(*self*.front, *self*.size):

print(*self*.queue[i], end=" ")

print()

q = CircularQueue(5)

print("\nEnqueue:")

q.enqueue(1)

q.enqueue(2)

q.enqueue(3)

q.enqueue(4)

q.enqueue(5)

q.print()

q.dequeue()

q.dequeue()

print("\nAfter Dequeue:")

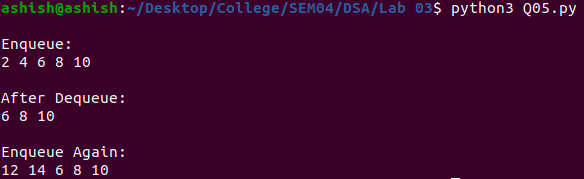
q.print()

print("\nEnqueue Again:")

q.enqueue(6)

q.enqueue(7)

q.print()



1. Write a Python program to implement deque.

import collections

de = collections.deque([1,2,3])

de.append(4)

print ("The deque after appending at right is : ")

print (de)

de.appendleft(6)

print ("The deque after appending at left is : ")

print (de)

de.pop()

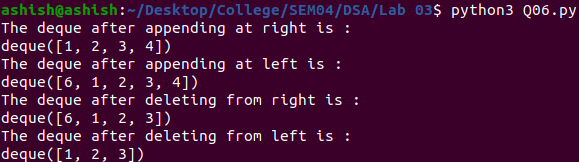
print ("The deque after deleting from right is : ")

print (de)

de.popleft()

print ("The deque after deleting from left is : ")

print (de)



1. Write a Python program to reverse a stack.

def stack():

stack = []

return stack

def isEmpty(stack):

return len(stack) == 0

def length(stack):

return len(stack)

def push(stack, item):

stack.append(item)

print("pushed item:", item)

def pop(stack):

if (isEmpty(stack)):

return "stack is empty"

return stack.pop()

def reverse(string):

n = len(string)

s = stack()

for i in range(0, n, 1):

push(s, string[i])

string = ""

for i in range(0, n, 1):

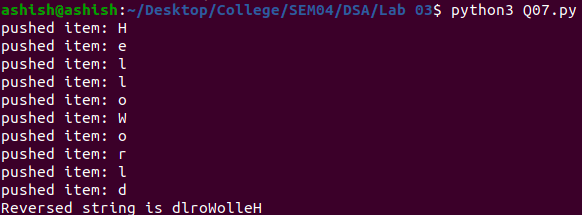
string += pop(s)

return string

string = "HelloWorld"

string = reverse(string)

print("Reversed string is " + string)



8. Write a Python program to reverse a queue.

class Queue:

def \_\_init\_\_(node):

node.data = []

def Empty(node):

return node.data == []

def enQueue(node, data):

node.data.insert(0,data)

def deQueue(node):

return node.data.pop()

def Reverse():

if(Q.Empty()):

return

temp = Q.data[-1]

Q.deQueue()

Reverse()

Q.enQueue(temp)

Q = Queue()

Q.enQueue(100)

Q.enQueue(80)

Q.enQueue(60)

Q.enQueue(40)

Q.enQueue(20)

print(Q.data)

Reverse()

print(Q.data)

