Lab 4

DSA BS DS Fall 2022

Use stack for Task 1 and Task 2

Task 1 10 marks

Balanced Parenthesis

Your task is to implement a function that receives a string and returns true if the parentheses in the expression are balanced otherwise returns false.

You should only consider round brackets "()" as parentheses. Parentheses are considered balanced if each opening parenthesis has its corresponding closing parenthesis and they are properly nested.

For example:
"(a + b) * (c - d)" -> balanced "(((a + b) * (c - d)))" -> balanced

"((a + b) * (c - d)" -> not balanced (missing closing parenthesis) "(a + b) * (c - d))" -> not balanced (extra closing parenthesis)

def isBalanced(str);

Note: The expression may contains all type of parenthesis "(,{,[,],},)"

Task 2 10 marks

String words Reverse

Your task is to implement a function that receives a string and reverses each word in it using stack. You can assume that the string only consists of alphabets and spaces. The order of the words should remain same but characters within each word should get reversed.

For example:
String: "Welcome to DSA"
Modified string: "emocleW ot ASD"

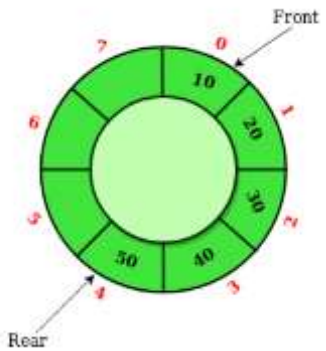
def reverseWords(str);

Task 3 10 marks

Circular Queue

A Circular Queue is an extended version of a normal queue where the last element of the queue is connected to the first element of the queue forming a circle.

The operations are performed based on FIFO (First In First Out) principle. It is also called **'Ring Buffer'**.



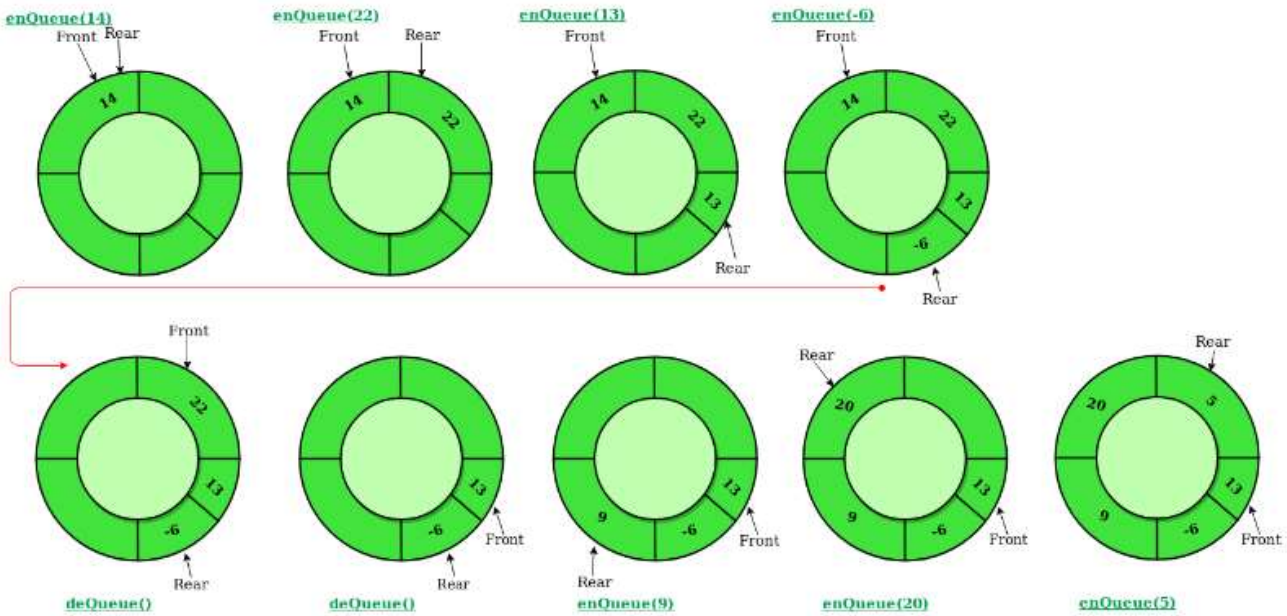
In a normal Queue, we can insert elements until queue becomes full. But once queue becomes full, we can not insert the next element even if there is a space in front of queue.

Operations on Circular Queue:

- **Front:** Get the front item from the queue.
- **Rear:** Get the last item from the queue.
- **enQueue(value)** This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at the rear position.
    - Check whether the queue is full – [i.e., the rear end is in just before the front end in a circular manner].
    - If it is full then display Queue is full.
    - If the queue is not full then, insert an element at the end of the queue.
- **deQueue()** This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from the front position.
    - Check whether the queue is Empty.
    - If it is empty then display Queue is empty.
    - If the queue is not empty, then get the last element and remove it from the queue.

Illustration of Circular Queue Operations:

Follow the below image for a better understanding of the enqueue and dequeue operations.



Working of Circular queue operations

You have to complete the following ADT

```
class CircularQueue():

    # constructor
    def __init__(self, size):  # initializing the class
        self.size = size

        # initializing queue with none
        self.queue = [None for i in range(size)]
        self.front = self.rear = -1

    def enqueue(self, data):
```

```
    def dequeue(self):
    def display(self):
    def isEmpty(self):
    def isFull(self):


# Main function
if __name__ == "__main__":

    ob = CircularQueue(5)
    ob.enqueue(14)
    ob.enqueue(22)
    ob.enqueue(13)
    ob.enqueue(-6)
    ob.display()
    print("Deleted value = ", ob.dequeue())
    print("Deleted value = ", ob.dequeue())
    ob.display()
    ob.enqueue(9)
    ob.enqueue(20)
    ob.enqueue(5)
    ob.display()
```

The output of this main should be

```
Elements in the circular queue are: 14 22 13 -6
Deleted value =  14
Deleted value =  22
Elements in the circular queue are: 13 -6
Elements in Circular Queue are: 13 -6 9 20 5
```

Task 4 — 30 marks

In this Task you are required to simulate ***Schedule***r. Scheduler is something which is responsible for assigning the CPU time to the processes.

The ***Scheduler*** should take in ***Processes*** and add them to a Queue. o Once all the ***Processes*** are read, your program would execute each.

But before executing any process, the program should print the name and other information about the process.
Now it should start executing the processes in the Queue. For each process, execution just means that process stays at the head of the Queue until time equal to its ***execution time*** has passed. The process is deleted from the list after that. At this moment your program should print that such and such process has finished execution. E.g.
Messenger.exe, 6 completed execution

You must remember that this is a simulator. This means that you'll have to define your own timer, one which increments in unit steps.

After the passage of every 10 time units, you program must stop the processing of current process and start the next process, the current process will be taken from the front of the queue to the end of the queue and the time remaining to complete process should be decremented.

Hints:

Design a class "Process" which will have following data members,

processId
processName
executionTime
And a method to display its state.

Create a circular Queue and add the process in it.

Process the “Process” at front of Queue and then remove it from front and add at the end and continue the processing. Remove the “Process” completely from queue if it has “0” “execution time” remaining.

Repeat this process until the Queue is not empty.

*You have to create the following files and will have to code all the classes listed in these files.

process.py

```
class Process:
    def __init__(self, processId, processName, processExecTime):
        self.processId = processId
        self.processName = processName
        self.processExecTime = processExecTime
```

scheduler.py

```
from queue import Queue
class Scheduler:
    def __init__(self, processArray, processArrayLength, timeQuantum):
        self.processArray = processArray
        self.processArrayLength = processArrayLength
        self.timeQuantum = timeQuantum

    def assignProcessor(self):
```

queue.py // use the task 1 queue

```
class Queue:
    def __init__(self):

    def enQueue(self):

    def deQueue(self):

    def isEmpty(self):

    def isFull(self):
```

main.py

```
from PROCESS import Process
from SCHEDULER import Scheduler

if __name__ == "__main__":
    arr = [Process(1, "notepad", 20), Process(13, "mp3player", 5), Process(4,
"bcc", 30), Process(11, "explorer", 2)]
    s = Scheduler(arr, 4, 5)
```

```
    s.assignProcessor()
```

The output of this program should be:

```
Executing process notepad for 5 units
Executing process mp3player for 5 units
Executing process bcc for 5 units
Executing process explorer for 2 units
Executing process notepad for 5 units
Executing process bcc for 5 units
Executing process notepad for 5 units
Executing process bcc for 5 units
Executing process notepad for 5 units
Executing process bcc for 5 units
Executing process bcc for 5 units
Executing process bcc for 5 units
```