# Data Structures & Algorithms – Lab #3

**Aim:** Getting Familiar with Stack, Queue and Deque and Their Applications, collections Module in Python

**Topics:**

1. Implementing a Stack Using List
2. Implementing a Queue Using List
3. Implementing a Deque
4. collections Module in Python

## Lab Question

**Q1** – Implement the **Stack** data structure by defining a class for it. Consider all stack methods. Then create an object of the stack and do the following operations:

- Push these elements to the stack: 10 → 50 → 30 → 20
- Print the stack size.
- Print the element at the top of the stack.
- Pop an element from the stack.
- Pop an element from the stack.
- Print the element at the top of the stack.

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

    def isEmpty(self):
        return self.items == []

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

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

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

    def size(self):
        return len(self.items)
```

```
# Creating a Stack object
stack = Stack()

# Calling object methods
stack.push(10)
stack.push(50)
stack.push(30)
stack.push(20)
print("Number of elements in the stack:", stack.size())
print("Element at the top of the stack:", stack.peek())

# Removing the two top elements from the stack
stack.pop()
stack.pop()

print("Element at the top of the stack(After removing two elements):", stack.peek())
```

**Q2 –** Implement the **Queue** data structure by defining a class for it. Consider all queue methods. Then create an object of the queue and do the following operations:

- Enqueue these elements to the queue: 10 → 50 → 30 → 20
- Print the queue size.
- Print the front element of the queue.
- Dequeue an element from the queue.
- Dequeue an element from the queue.
- Print the front element of the queue.

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

    def isEmpty(self):
        return self.items == []

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

    def dequeue(self):
        return self.items.pop(0)

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

    def size(self):
        return len(self.items)
```

```
# Creating a Queue object
queue = Queue()

# Calling object methods
queue.enqueue(10)
queue.enqueue(50)
queue.enqueue(30)
queue.enqueue(20)
print("Number of elements in the queue:", queue.size())
print("The front element of the queue:", queue.front())

# Removing two elements from the queue
queue.dequeue()
queue.dequeue()

print("The front element of the queue(After removing two elements):", queue.front())
```

**Q3 –** Implement the **Deque** data structure using collections module. Then create an object of the deque and do the following operations:

- Add 10 to the end of the deque.
- Add 20 to the end of the deque.
- Add 50 to the beginning of the deque.
- Add 30 to the beginning of the deque.
- Print the deque size.
- Remove the last element from the deque.
- Remove the first element from the deque.
- Print the front element of the deque.

```
# Importing deque data structure from collections module
from collections import deque

# Creating an empty deque
my_deque = deque()

# Adding elements to the end of deque
my_deque.append(10)
my_deque.append(20)

# Adding elements to the beginning of deque
my_deque.appendleft(50)
my_deque.appendleft(30)

print("Deque:", my_deque)
print("Number of elements in the deque:", len(my_deque))

# Removing the last element from deque
my_deque.pop()
print("Deque after removing the last element:", my_deque)

# Removing the first element from deque
my_deque.popleft()
print("Deque after removing the first element:",my_deque)

print("Front element of the deque:", my_deque[0])
```

**Q4 –** Write a Python function **using stack** to check for balanced parentheses.

- The parentheses are balanced if:
  - Every opening parenthesis has a corresponding closing parenthesis of the same type.
  - The pairs of parentheses are properly nested.
- Example of balanced parentheses: { [ ( ) [ ] ] }
- Example of non-balanced parentheses: { [ ) ] }

```
def check_balanced_parantheses(myStr):
    stack = []
    for symbol in myStr:
        if symbol in '({[':
            stack.append(symbol)
        elif symbol in ')}]':
            if (not stack or
                (symbol == ')' and stack[-1] != '(') or
                (symbol == '}' and stack[-1] != '{') or
                (symbol == ']' and stack[-1] != '[')):
                print(False)
                break
            stack.pop()

    else:
        print(True if not stack else False)

check_balanced_parantheses('{([])}')
check_balanced_parantheses('{([)}')
```