

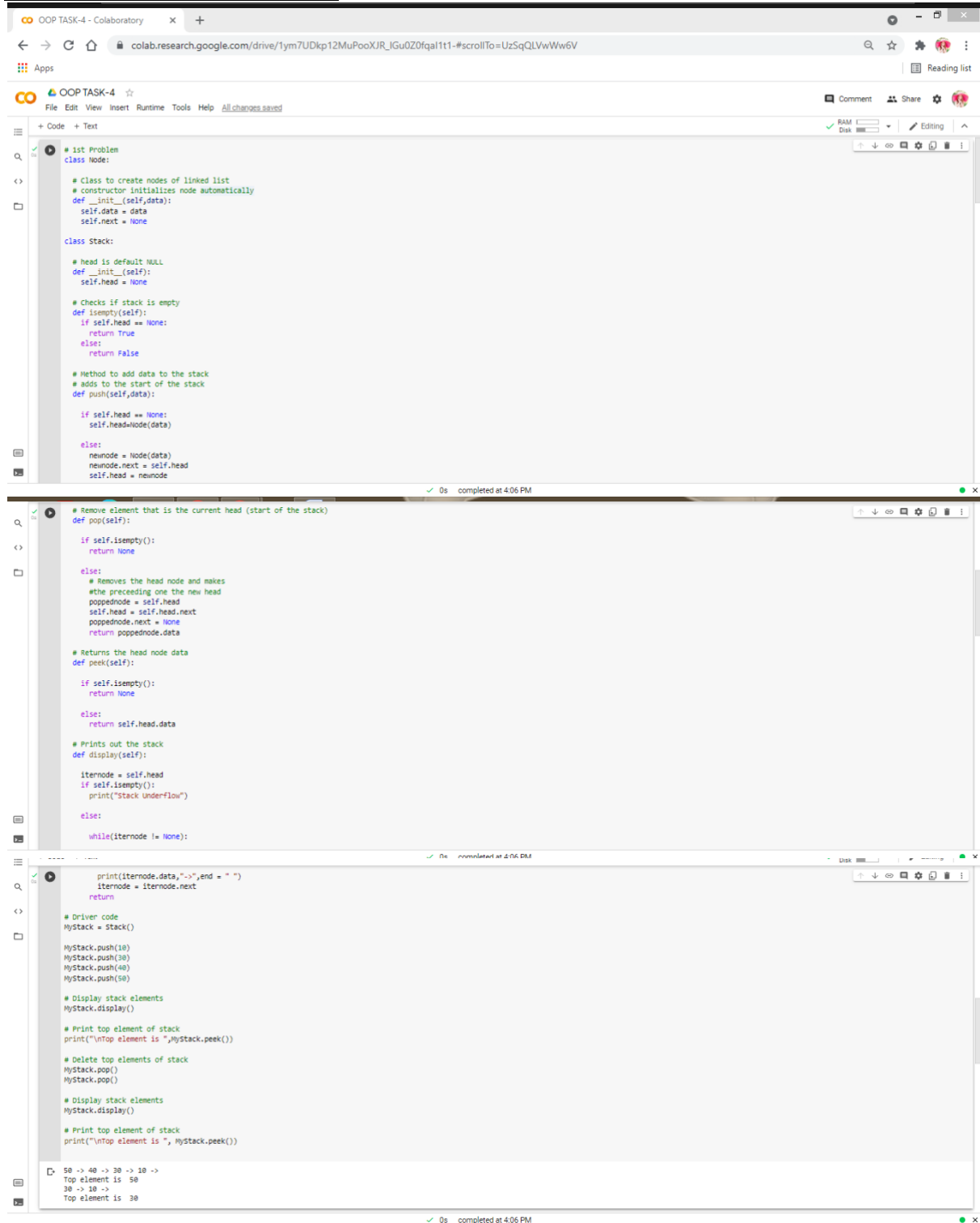
Object Oriented Programming

Lab Assignment –4

Name: - E.Vaishnavi

RegNo: 20BCS044

Problem-1 Solution:-



```
# 1st Problem
class Node:
    # Class to create nodes of linked list
    # constructor initializes node automatically
    def __init__(self, data):
        self.data = data
        self.next = None

class Stack:
    # head is default NULL
    def __init__(self):
        self.head = None

    # Checks if stack is empty
    def isempty(self):
        if self.head == None:
            return True
        else:
            return False

    # Method to add data to the stack
    # adds to the start of the stack
    def push(self, data):
        if self.head == None:
            self.head = Node(data)
        else:
            newnode = Node(data)
            newnode.next = self.head
            self.head = newnode

    # Remove element that is the current head (start of the stack)
    def pop(self):
        if self.isempty():
            return None
        else:
            # Removes the head node and makes
            # the preceding one the new head
            poppednode = self.head
            self.head = self.head.next
            poppednode.next = None
            return poppednode.data

    # Returns the head node data
    def peek(self):
        if self.isempty():
            return None
        else:
            return self.head.data

    # Prints out the stack
    def display(self):
        iternode = self.head
        if self.isempty():
            print("Stack Underflow")
        else:
            while(iternode != None):
                print(iternode.data, "--> ", end = " ")
                iternode = iternode.next
            return

# Driver code
MyStack = Stack()

MyStack.push(10)
MyStack.push(30)
MyStack.push(40)
MyStack.push(50)

# Display stack elements
MyStack.display()

# Print top element of stack
print("Top element is ", MyStack.peek())

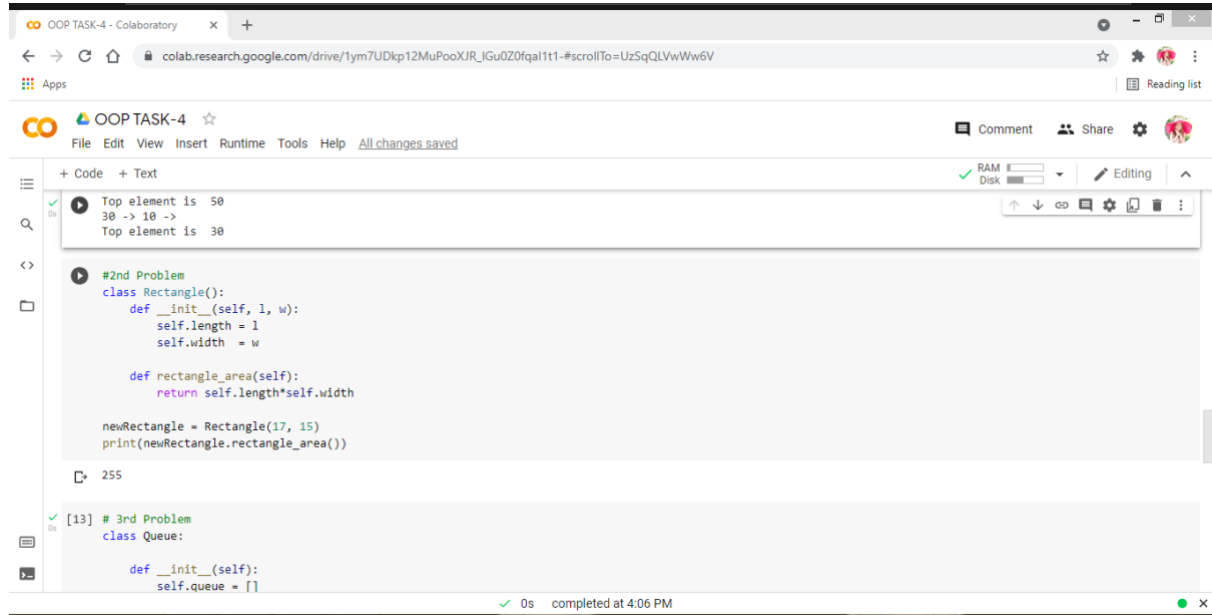
# Delete top elements of stack
MyStack.pop()
MyStack.pop()

# Display stack elements
MyStack.display()

# Print top element of stack
print("Top element is ", MyStack.peek())
```

50 --> 40 --> 30 --> 10 -->
Top element is 50
30 --> 10 -->
Top element is 30

Problem-2 Solution:-



A screenshot of a Google Colaboratory notebook titled "OOP TASK-4". The notebook shows the solution to Problem 2, which involves a `Rectangle` class. The code defines the class with an `__init__` method that takes length and width, and a `rectangle_area` method that returns the area. An instance `newRectangle` is created with length 17 and width 15, and its area is printed, resulting in 255. The output shows the execution of the code and the final result.

```
#2nd Problem
class Rectangle():
    def __init__(self, l, w):
        self.length = l
        self.width = w

    def rectangle_area(self):
        return self.length*self.width

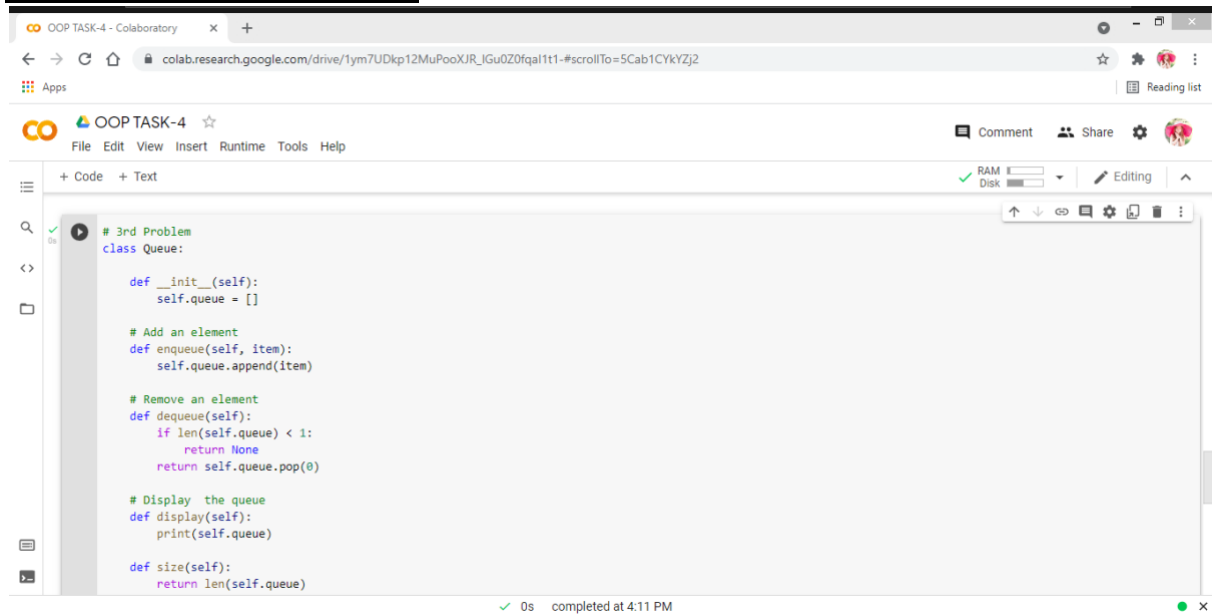
newRectangle = Rectangle(17, 15)
print(newRectangle.rectangle_area())
```

255

```
[13] # 3rd Problem
class Queue:
    def __init__(self):
        self.queue = []
```

0s completed at 4:06 PM

Problem-3 Solution:-



A screenshot of a Google Colaboratory notebook titled "OOP TASK-4". The notebook shows the solution to Problem 3, which involves a `Queue` class. The code defines the class with an `__init__` method, an `enqueue` method to add elements, a `dequeue` method to remove elements, a `display` method to show the queue, and a `size` method to return the length of the queue. The output shows the execution of the code and the final result.

```
# 3rd Problem
class Queue:
    def __init__(self):
        self.queue = []

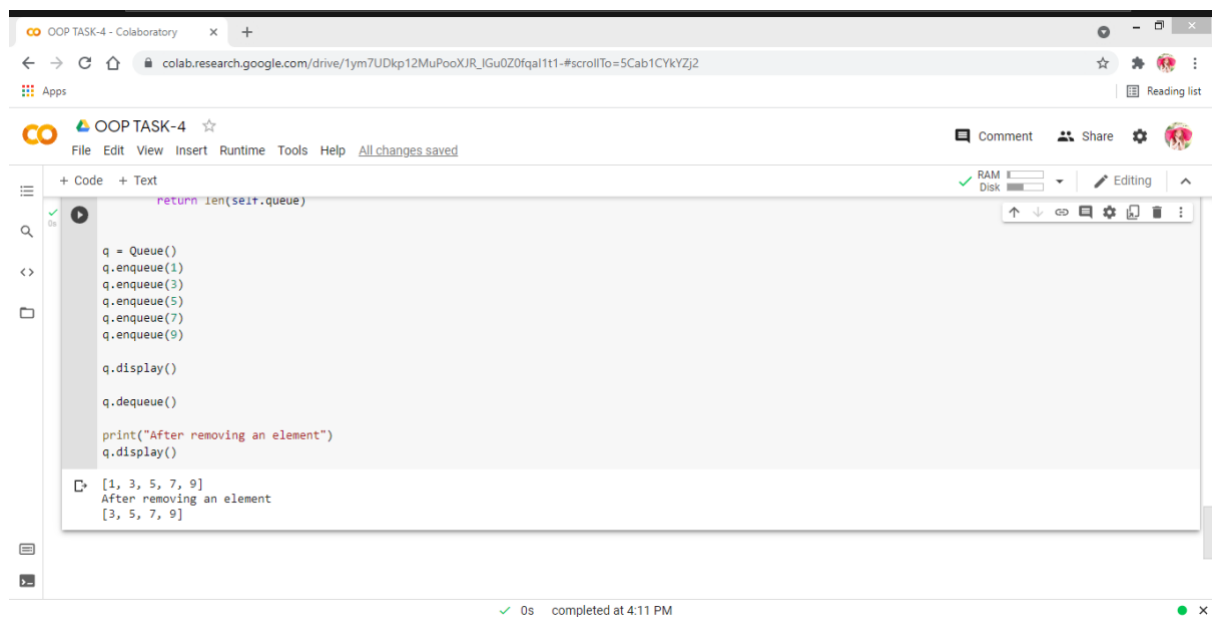
    # Add an element
    def enqueue(self, item):
        self.queue.append(item)

    # Remove an element
    def dequeue(self):
        if len(self.queue) < 1:
            return None
        return self.queue.pop(0)

    # Display the queue
    def display(self):
        print(self.queue)

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

0s completed at 4:11 PM



A screenshot of a Google Colaboratory notebook titled "OOP TASK-4". The notebook shows the solution to Problem 3, which involves a `Queue` class. The code defines the class with an `__init__` method, an `enqueue` method to add elements, a `dequeue` method to remove elements, a `display` method to show the queue, and a `size` method to return the length of the queue. The output shows the execution of the code and the final result.

```
return len(self.queue)

q = Queue()
q.enqueue(1)
q.enqueue(3)
q.enqueue(5)
q.enqueue(7)
q.enqueue(9)

q.display()

q.dequeue()

print("After removing an element")
q.display()
```

[1, 3, 5, 7, 9]
After removing an element
[3, 5, 7, 9]

0s completed at 4:11 PM