

## Practical:8

Write a program implement stack using class and object and implement push, pop and traverse operation

Code:

```
# Python program to demonstrate  
# stack implementation using a linked list.  
# node class
```

```
class Node:
```

```
    def __init__(self, value):  
        self.value = value  
        self.next = None
```

```
class Stack:
```

```
    # Initializing a stack.  
    # Use a dummy node, which is  
    # easier for handling edge cases.  
    def __init__(self):  
        self.head = Node("head")  
        self.size = 0
```

# String representation of the stack

```
def __str__(self):  
    cur = self.head.next  
    out = ""  
    while cur:  
        out += str(cur.value) + "->"  
        cur = cur.next  
    return out[:-3]
```

# Get the current size of the stack

```
def getSize(self):  
    return self.size
```

# Check if the stack is empty

```
def isEmpty(self):  
    return self.size == 0
```

# Get the top item of the stack

```
def peek(self):  
  
    # Sanitary check to see if we  
    # are peeking an empty stack.  
    if self.isEmpty():
```

```
        raise Exception("Peeking from an empty stack")
    return self.head.next.value

# Push a value into the stack.
def push(self, value):
    node = Node(value)
    node.next = self.head.next
    self.head.next = node
    self.size += 1

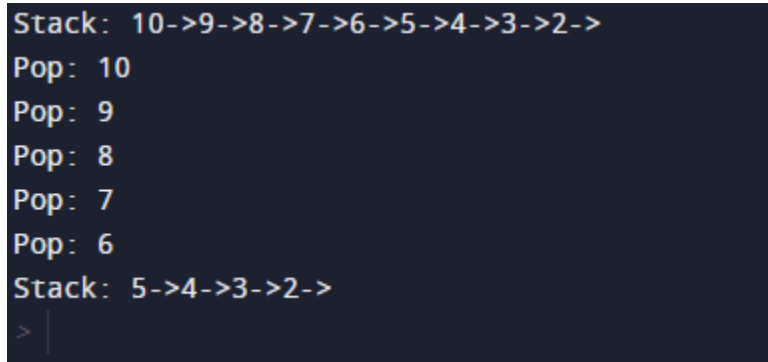
# Remove a value from the stack and return.
def pop(self):
    if self.isEmpty():
        raise Exception("Popping from an empty stack")
    remove = self.head.next
    self.head.next = self.head.next.next
    self.size -= 1
    return remove.value

# Driver Code
if __name__ == "__main__":
    stack = Stack()
    for i in range(1, 11):
```

```
        stack.push(i)
    print(f"Stack: {stack}")

    for _ in range(1, 6):
        remove = stack.pop()
        print(f"Pop: {remove}")
    print(f"Stack: {stack}")
```

Output:

A terminal window with a dark background showing the output of a Python program. The output consists of several lines: 'Stack: 10->9->8->7->6->5->4->3->2->', 'Pop: 10', 'Pop: 9', 'Pop: 8', 'Pop: 7', 'Pop: 6', 'Stack: 5->4->3->2->', and a prompt '> |' on the final line.

```
Stack: 10->9->8->7->6->5->4->3->2->
Pop: 10
Pop: 9
Pop: 8
Pop: 7
Pop: 6
Stack: 5->4->3->2->
> |
```

GitHub link: [https://github.com/SwetSoni/Python\\_Practicals](https://github.com/SwetSoni/Python_Practicals)