

Day-13 : Stack & Queue

Problem statement: Implement a stack using an array.

```
class Stack:

    def __init__(self):
        self.stack = []

    def push(self, item):
        self.stack.append(item)

    def pop(self):
        if not self.is_empty():
            return self.stack.pop()
        else:
            return None

    def is_empty(self):
        return len(self.stack) == 0

    def print_stack(self):
        if not self.is_empty():
            print("Stack:")
            for item in reversed(self.stack):
                print(item)
        else:
            print("Stack is empty")

stack = Stack()

stack.push(10)
stack.push(20)
stack.push(30)
```

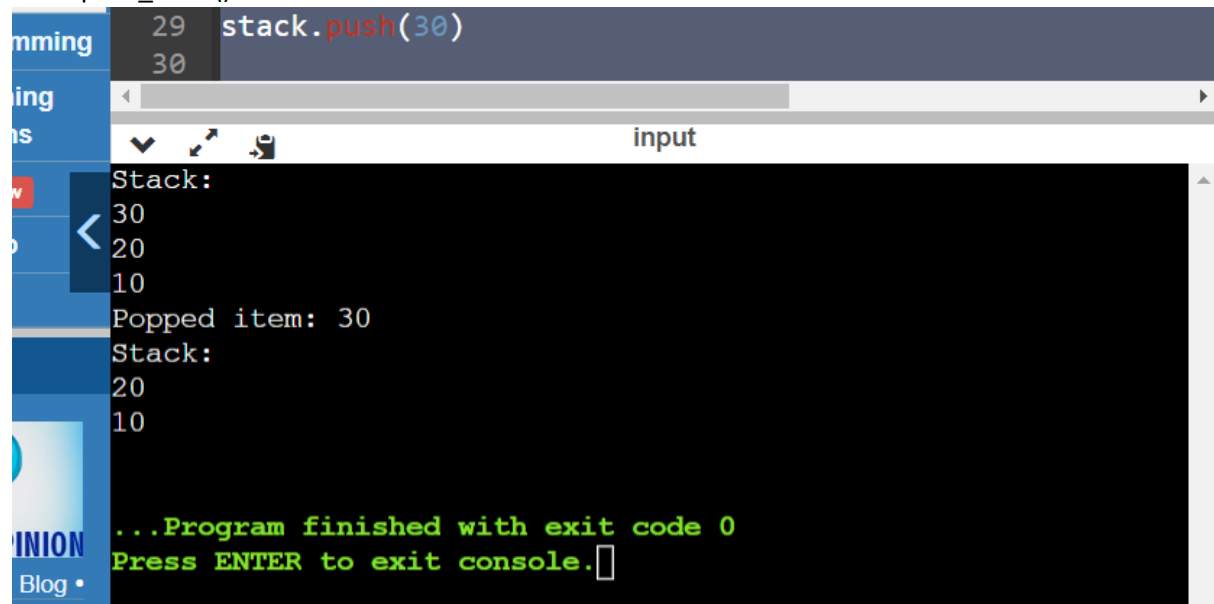
```
stack.print_stack()
```

```
popped_item = stack.pop()
```

```
if popped_item is not None:
```

```
    print("Popped item:", popped_item)
```

```
stack.print_stack()
```



```
29 stack.push(30)
30
Stack:
30
20
10
Popped item: 30
Stack:
20
10
...Program finished with exit code 0
Press ENTER to exit console.
```

Problem Statement: Implement Queue Data Structure using Array with all functions like pop, push, top, size, etc.

```
class Queue:
```

```
    def __init__(self):
```

```
        self.queue = []
```

```
    def is_empty(self):
```

```
        return len(self.queue) == 0
```

```
    def size(self):
```

```
        return len(self.queue)
```

```
def enqueue(self, item):  
    self.queue.append(item)
```

```
def dequeue(self):  
    if self.is_empty():  
        return None  
    return self.queue.pop(0)
```

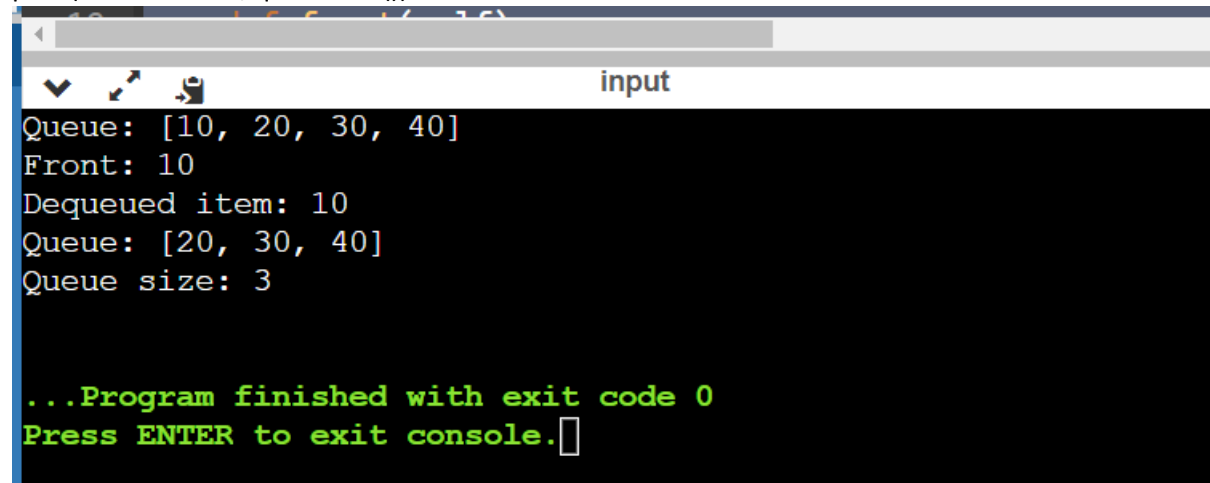
```
def front(self):  
    if self.is_empty():  
        return None  
    return self.queue[0]
```

```
def print_queue(self):  
    if self.is_empty():  
        print("Queue is empty")  
    else:  
        print("Queue:", self.queue)
```

```
queue = Queue()  
queue.enqueue(10)  
queue.enqueue(20)  
queue.enqueue(30)  
queue.enqueue(40)
```

```
queue.print_queue()  
print("Front:", queue.front())  
print("Dequeued item:", queue.dequeue())  
queue.print_queue()
```

```
print("Queue size:", queue.size())
```



```
input
Queue: [10, 20, 30, 40]
Front: 10
Dequeued item: 10
Queue: [20, 30, 40]
Queue size: 3

...Program finished with exit code 0
Press ENTER to exit console.
```

Problem Statement: Implement a **Stack** using a single **Queue**.

```
class Stack:
```

```
    def __init__(self):
```

```
        self.queue = []
```

```
    def push(self, value):
```

```
        self.queue.append(value)
```

```
    def pop(self):
```

```
        if not self.is_empty():
```

```
            size = len(self.queue)
```

```
            for _ in range(size - 1):
```

```
                self.queue.append(self.queue.pop(0))
```

```
            return self.queue.pop(0)
```

```
    def top(self):
```

```
        if not self.is_empty():
```

```
            size = len(self.queue)
```

```
            for _ in range(size - 1):
```

```
                self.queue.append(self.queue.pop(0))
```

```
    return self.queue[0]
```

```
def size(self):
```

```
    return len(self.queue)
```

```
def is_empty(self):
```

```
    return len(self.queue) == 0
```

```
def print_stack(self):
```

```
    print("Stack:", self.queue)
```

```
stack = Stack()
```

```
stack.push(1)
```

```
stack.push(2)
```

```
stack.push(3)
```

```
stack.print_stack()
```

```
print("Size:", stack.size())
```

```
print("Top:", stack.top())
```

```
print("Pop:", stack.pop())
```

```
stack.print_stack()
```

```
34 stack.push(3)
35 stack.print_stack()
36 print("Size:", stack.size())
37 print("Top:", stack.top())
38 print("Pop:", stack.pop())
39 stack.print_stack()
40
```

input

```
Stack: [1, 2, 3]
Size: 3
Top: 3
Pop: 2
Stack: [3, 1]

...Program finished with exit code 0
Press ENTER to exit console.
```

Problem Statement: Given a Stack having some elements stored in it. Can you implement a Queue using the given Stack? **Using two Stacks where push operation is $O(N)$.**

```
class QueueUsingStack:
```

```
    def __init__(self):
```

```
        self.stack1 = []
```

```
        self.stack2 = []
```

```
    def enqueue(self, value):
```

```
        self.stack1.append(value)
```

```
    def dequeue(self):
```

```
        if not self.stack1 and not self.stack2:
```

```
raise Exception("Queue is empty (underflow)")
```

```
if not self.stack2:
```

```
    while self.stack1:
```

```
        self.stack2.append(self.stack1.pop())
```

```
    return self.stack2.pop()
```

```
queue = QueueUsingStack()
```

```
queue.enqueue(10)
```

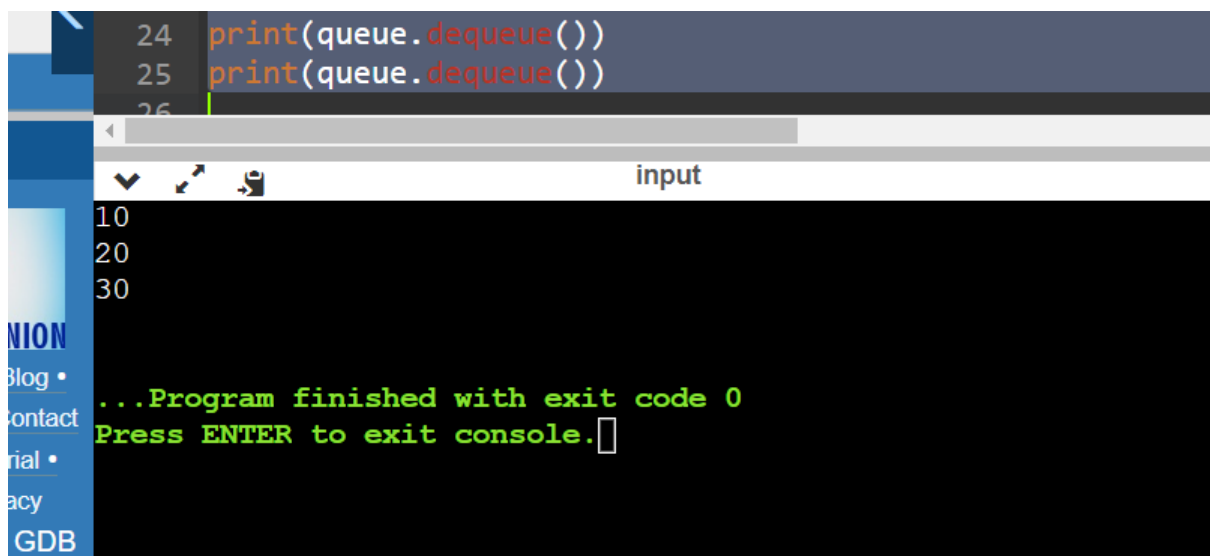
```
queue.enqueue(20)
```

```
queue.enqueue(30)
```

```
print(queue.dequeue())
```

```
print(queue.dequeue())
```

```
print(queue.dequeue())
```



The screenshot shows a code editor with three lines of Python code: `print(queue.dequeue())` on line 24, `print(queue.dequeue())` on line 25, and `print(queue.dequeue())` on line 26. Below the code editor is a terminal window titled "input". The terminal displays the output of the program: `10`, `20`, and `30` on separate lines. At the bottom of the terminal, it says `...Program finished with exit code 0` and `Press ENTER to exit console.` with a cursor. On the left side of the terminal, there is a sidebar with a blue header "NION" and a list of links: "Blog", "Contact", "Tutorial", "Privacy", and "GDB".

Problem Statement: Check Balanced Parentheses. Given string str containing just the characters '(', ')', '{', '}', '[' and ']', check if the input string is valid and return true if the string is balanced otherwise return false.

Note: string str is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

```
def is_balanced_parentheses(string):
```

```
    stack = []
```

```
    opening_brackets = ['(', '[', '{']
```

```
    closing_brackets = [')', ']', '}']
```

```
    bracket_map = {'(': ')', '[': ']', '{': '}'}
```

```
    for char in string:
```

```
        if char in opening_brackets:
```

```
            stack.append(char)
```

```
        elif char in closing_brackets:
```

```
            if len(stack) == 0:
```

```
                return False
```

```
            if stack[-1] == bracket_map[char]:
```

```
                stack.pop()
```

```
        else:
```

```
            return False
```



```
return len(stack) == 0
```

```
input_string = "([]{})"
```

```
print(input_string)
```

```
print(is_balanced_parentheses(input_string))
```

```
input_string = "([])"
```

```
print(input_string)
```

```
print(is_balanced_parentheses(input_string))
```

```
input_string = "((()))"
```

```
print(input_string)
```

```
print(is_balanced_parentheses(input_string))
```

```
new
Up
in
30
31 input_string = "((()))"
32 print(input_string)
33 print(is_balanced_parentheses(input_string))
34

input
([[]{}])
True
([])
False
((()))
False
...Program finished with exit code 0
Press ENTER to exit console.
```

Problem Statement: Given a circular integer array **A**, return the next greater element for every element in A. The next greater element for an element x is the first element greater than x that we come across while traversing the array in a clockwise manner. If it doesn't exist, return -1 for this element.

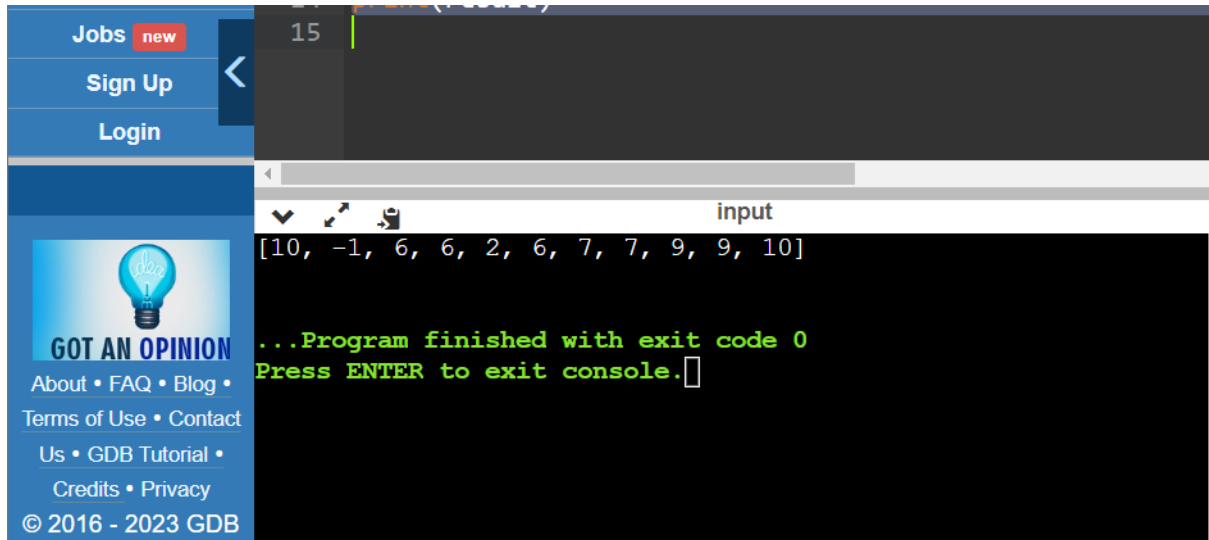
```
def next_greater_elements(N, A):
    stack = []
    result = [-1] * N
    for _ in range(2):
        for i in range(N):
            while stack and A[i] > A[stack[-1]]:
                result[stack.pop()] = A[i]
            stack.append(i)
    return result
```

N = 11

```
A = [3, 10, 4, 2, 1, 2, 6, 1, 7, 2, 9]
```

```
result = next_greater_elements(N, A)
```

```
print(result)
```



The screenshot shows a web application interface. On the left is a blue sidebar with navigation links: 'Jobs' (with a 'new' badge), 'Sign Up', 'Login', 'About', 'FAQ', 'Blog', 'Terms of Use', 'Contact', 'Us', 'GDB Tutorial', 'Credits', 'Privacy', and '© 2016 - 2023 GDB'. The main area is a dark-themed terminal window. At the top, it shows '15' and a cursor. Below that, it displays 'input' and a list of numbers: '[10, -1, 6, 6, 2, 6, 7, 7, 9, 9, 10]'. The terminal output shows '...Program finished with exit code 0' and 'Press ENTER to exit console.' with a cursor.