Object Oriented Programming, Part 3 - Stacks and Queues

(In the beginning, there was Stack Overflow)

Pragathi N pragathi.code@gmail.com
Ted Tran 42hourai@gmail.com

Summary: *This PDF is an introduction to* `stacks` *and* `queues` *in Python.*

Contents

Chapter I

Foreword

Leptodactylus fallax, commonly (and deceptively) known as the mountain chicken or giant ditch frog, is a species of frogs that is native to the Caribbean islands of Dominica and Montserrat. The population has declined 81% in the last ten years and this species is now critically endangered. In 2004 it was estimated that the population possibly was as low as 8,000 individuals. One of the main threats is human consumption.

Being deliciously chicken-tasting is dangerous.

Chapter II

Goals

- Review stacks and queues.

- Write your own stack and queue classes.

- Practice using stacks and queues to accomplish a variety of tasks

Chapter III

Introduction
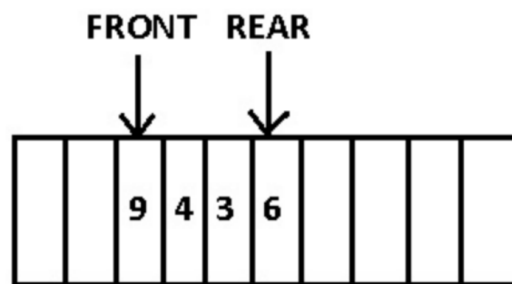
Prepare to work with stacks and queues!

These two are linear data structures that acts like a container of objects. Difference between the two data structures is in the way the elements are accessed.

`Stacks:` LIFO and `Queues:` FIFO

If you don't know what LIFO means, google it! Or bother Pragathi, her number is 1-888-447-5594. If you want someone nicer, call 605-475-6966!
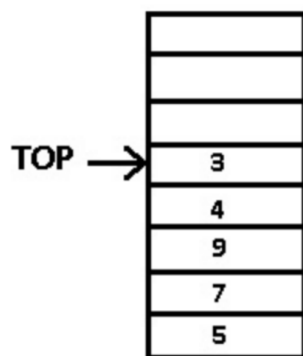
STACK

QUEUE

Chapter IV

Exercise 00 : Stack Creation

	Stack Creation
Topics to study :	
Files to turn in : `stack.py`	
Notes : `n/a`	

Make use of the Node class for the elements of the stack

```python
class Node:
    def __init__(self, value, next = None):
        self.data = value
        self.next = next
```

Implement a stack class with the following methods:

def __init__(self): Initialize the stack class with required variables

def isEmpty(self): Checks if the stack is empty or not (return None if empty)

def push(self, data): Adds an element to the stack

def pop(self): removes the top element from the stack

def peek(self): returns the value of the top element of the stack

def size(self): returns the size of the stack

def __str__(self): prints the elements of the stack

Chapter V

Exercise 01 : Basic Arithmetic

	Basic Arithmetic
Topics to study :	
Files to turn in : `math.py`	
Notes :	

Input a list and create a stack of the elements in the given list and implement the following operations on the stack. Start from the top of the stack when doing operations! Print the values in the same order as functions

- Calculate the total number of elements in the stack.

- Calculate the sum of all the elements in the stack.

- Multiply all the elements in the stack.

- Find the mean of the stack

- Lastly, find the maximum and minimum elements of the stack.

```
?> python math.py
Enter the numbers: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Total count = 10
Sum = 55
Product = 3628800
Mean = 5
Min = 1
Max = 10
```

Chapter VI

Exercise 02 : Queue Creation

	Queue Creation
Topics to study :	
Files to turn in : `queue.py`	
Notes :	

Make use of the Node class for the elements of the queue

```python
class Node:
    def __init__(self, value, next = None):
        self.data = value
        self.next = next
```

Implement a queue class with the following methods:

def __init__(self): Initialize the queue class with required variables (Front stores the front node and rear stores the last node)

def isEmpty(self): Checks if the queue is empty or not (return None if empty)

def enqueue(self, data): Inserts an element to rear of the queue

def dequeue(self): Removes an element from the front of the queue

def front(self): Returns the front value of the queue

def size(self): Returns the length of the queue

def __str__(self): prints the elements of the queue

Chapter VII

Exercise 03 : Reverse String

	Reverse String
Topics to study :	
Files to turn in : `strrev.py`	
Notes :	

Ask the user to enter the string and call the strrev method to reverse tht input string.

- Function prototyped as `def strrev(input_string)`
- "!ihtagarP" returns "Pragathi!"

> 💡 How do you read the last character as the first character (LIFO)?

```
?> python strrev.py
Enter the string to be reversed: !ihtagarP
Reversed String: Pragathi!
```

Chapter VIII

Exercise 04 : Reversing first K elements

	Reversing first K elements
Topics to study :	
Files to turn in : `revKElements.py`	
Notes :	

Given an integer k and a queue of integers, we need to reverse the order of the first k elements of the queue, leaving the other elements in the same relative order.

- Function prototyped as `def revKElements(input_string, k)`
- Create an empty stack.
- One by one dequeue items from given queue and push the dequeued items to stack.
- Enqueue the contents of stack at the back of the queue.
- Reverse the whole queue.
- input: 10,20,30,40,50,60,70,80,90,100 and k = 5 returns 50,40,30,20,10,60,70,80,90,100

```
?> python revKElements.py
Enter the list of numbers: 10,20,30,40,50,60,70,80,90,100
Enter k: 5
50,40,30,20,10,60,70,80,90,100
```

Chapter IX

Exercise 05 : Balance Parentheses

	Balance Parentheses
Topics to study :	
Files to turn in : `balanceCheck.py`	
Notes :	

Ask the user to enter the sequence and check whether the input string is a balanced parentheses.

- Function prototyped as `def isBalanced(input_string)`

- Valid sequence will consist a combination of "([{}])" or an empty string

- Function returns True for balanced sequence, False for not balanced ssequence

```
?> python balanceCheck.py
Enter the sequence:
True
Enter the sequence: [{}]
True
Enter the sequence: (((((
False
```

Chapter X

Exercise 06 : Base Converter

	Base Converter
Topics to study :	
Files to turn in : `baseConverter.py`	
Notes :	

Ask the user to enter a decimal number and a base. Call the function baseConverter to convert the given decimal number to it's equivalent number in the give base system.

- Function prototyped as `def baseConverter(decNum, base)`

- We shall take care of just positive integers and base is between 2 to 16

```
?> python baseConverter.py
Enter the decimal number: 25
Enter the base: 16
19
```

decNum is repeatedly divided by the base and pushed onto stack and then can be read from the top

Chapter XI

Bonus Part

We are very proud of you for reaching till here! Now, are you up for the final challenge?

	Evaluate Expression
Topics to study :	
Files to turn in : `evalExpr.py`	
Notes :	

Get ready to build a simple mathematical expression parser!

- Function prototyped as `def evalExpr(expr)`

- expr is the mathematical expression of the form, for example $(5 + 3) * 7 / (2 + 3)$, which is passed to the function and 11 is returned as the result

- Ensure that you handle brackets as well!

```
?> python evalExpr.py
Enter the expression: (5 + 3) * 7 / (2 + 3)
11
```