Name: Adish Shanbhag
Reg no: 251100670036

Algorithms and Data Structures for Big data Lab - Assignment 1

Stack Assignments

1. Implement unlimited size stack

Code:

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


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

    def pop(self):
        if self.is_empty():
            print("stack underflow - no elements found")
            return None
        value = self.stack.pop()
        print(value)

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

stack = Stack()
stack.push(10)
stack.push(20)
stack.push(30)
stack.pop()

print(stack.stack)
```

Output:

```
PS C:\Users\adish\OneDrive\Desktop\Python Projects> python -u "c:\Users\adish\O
neDrive\Desktop\Python Projects\assignment.py"
[10, 20, 30]
Popped element: 30
```

2. Implement limited size stack

Name: Adish Shanbhag
Reg no: 251100670036

Code:

```
class Stack:
    def __init__(self, max_size):
        self.max_size = max_size
        self.stack = []

    def push(self, item):
        if not self.is_full():
            self.stack.append(item)
        else:
           raise OverflowError("stack overflow")

    def pop(self):
        if self.is_empty():
            print("stack underflow - no elements found")
            return None
        value = self.stack.pop()
        print(value)

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

    def is_full(self):
        return len(self.stack) >= self.max_size



stack = Stack(3)
stack.push(10)
stack.push(20)
stack.push(30)
print(stack.stack)
try:
    stack.push(40)
except OverflowError as e:
    print(e)
stack.pop()
```

Output:

Name: Adish Shanbhag
Reg no: 251100670036

```
PS C:\Users\adish\OneDrive\Desktop\Python Projects> python -u "c:\Users\adish\O
neDrive\Desktop\Python Projects\assignment.py"
[10, 20, 30]
stack overflow
Popped element: 30
```

3. Reverse the content of a file using stack

Code:

```
class Stack:
    def __init__(self, max_size):
        self.max_size = max_size
        self.stack = []

    def push(self, item):
        if not self.is_full():
            self.stack.append(item)
        else:
           raise OverflowError("stack overflow")

    def pop(self):
        if self.is_empty():
            print("stack underflow - no elements found")
            return None
        value = self.stack.pop()
        return value

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

    def is_full(self):
        return len(self.stack) >= self.max_size
try:
    f = open("Adish.txt", "r")
    content = f.read()
except FileNotFoundError as e:
     print(e)


stack = Stack(len(content))
if content is None:
          raise FileNotFoundError
```

Name: Adish Shanbhag
Reg no: 251100670036

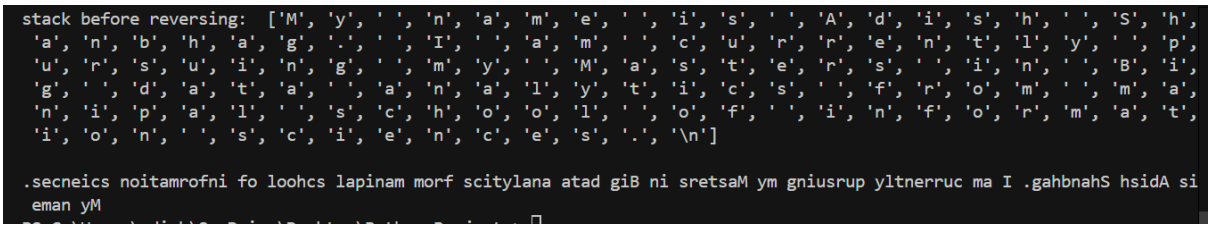
```
for letters in content:
     stack.push(letters)
     pass
print ("stack before reversing: ", format(stack.stack))

f = open("New_Adish.txt","w")
while not stack.is_empty():
     f.write(stack.pop())

f = open("New_Adish.txt", "r")
print(f.read())
```

Output:

Content of the file: “My name is Adish Shanbhag. I am currently pursuing my Masters in Big data analytics from manipal school of information sciences.”



4. Match the parenthesis using stack

Code:

```
class Stack:
    def __init__(self, max_size):
        self.max_size = max_size
        self.stack = []

    def push(self, item):
        if not self.is_full():
            self.stack.append(item)
        else:
           raise OverflowError("stack overflow")

    def pop(self):
        if self.is_empty():
            print("stack underflow - no elements found")
            return None
        value = self.stack.pop()
```

Name: Adish Shanbhag
Reg no: 251100670036

```
        return value

    def peek(self):
        if self.is_empty():
            return None
        return self.stack[-1]

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

    def is_full(self):
        return len(self.stack) >= self.max_size

    def parenthesis_balance(self, str):
        brackets = {')':'(','}':'{',']':'['}

        for bracket in str:
            if bracket in "{([":
                self.push(bracket)
            elif bracket in "})]":
                if self.peek() == brackets[bracket]:
                    self.pop()
                else:
                    return False
        if self.is_empty():
            return True
        else:
            return False



stack = Stack(5)

str = input("Enter the string to check parenthesis balancing: ")

if stack.parenthesis_balance(str):
    print("Balanced")
else:
    print("Non Balanced")
```

Output:

Name: Adish Shanbhag
Reg no: 251100670036

```
PS C:\Users\MSIS\Desktop\251100670036> & C:/Users/MSIS/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/MSIS/Desktop/251100670036/Assignment.py
Enter the string to check parenthesis balancing: )()()(
Non Balanced
PS C:\Users\MSIS\Desktop\251100670036> & C:/Users/MSIS/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/MSIS/Desktop/251100670036/Assignment.py
Enter the string to check parenthesis balancing: ({[]})
Balanced
PS C:\Users\MSIS\Desktop\251100670036> & C:/Users/MSIS/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/MSIS/Desktop/251100670036/Assignment.py
Enter the string to check parenthesis balancing: {[()()]
Non Balanced
PS C:\Users\MSIS\Desktop\251100670036>
```

5. Match the tags in HTML tag using stack

Code:

```python
import re



class Stack:
    def __init__(self): #, max_size):
        # self.max_size = max_size
        self.stack = []

    def push(self, item):
        # if not self.is_full():
            self.stack.append(item)
        # else:
        #     raise OverflowError("stack overflow")

    def pop(self):
        if self.is_empty():
            print("stack underflow - no elements found")
            return None
        value = self.stack.pop()
        return value


    def peek(self):
        if self.is_empty():
            return None
        return self.stack[-1]


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

    def is_full(self):
        return len(self.stack) >= self.max_size

    def tag_Balancer(self, Tags):
```

Name: Adish Shanbhag
Reg no: 251100670036

```
        for tag in Tags:
            if tag.find("/") == -1:
                stack.push(tag)
            else:
                temp = stack.peek()
                temp2 = tag.strip("<>")
                if temp.strip("<>") == temp2.lstrip("/"):
                    stack.pop()
                else:
                    return False
        return True


try:
    f = open("Test.html", "r")
    content = f.read()
except FileNotFoundError as e:
     print(e)
stack = Stack()

tags = re.findall(r"<[^>]+>", content)
cleaned_Tags = []
for tag in tags:
    if "DOCTYPE" in tag:
        continue
    cleaned_Tags.append(tag)

if stack.tag_Balancer(cleaned_Tags):
    print("Tags are Balanced")
else:
    print("Tags are Unbalanced")
```

Output:
Balanced Output

Name: Adish Shanbhag
Reg no: 251100670036

```
Test.html > html
  1   <!DOCTYPE html>
  2 v <html>
  3 v <head>
  4     <title>Simple Page</title>
  5   </head>
  6 v <body>
  7     <h1>Hello, World!</h1>
  8     <p>This is a minimal HTML example.</p>
  9   </body>
 10   </html>
 11
```

```
PS C:\Users\adish\OneDrive\Desktop\Python Projects> python -u "c:\Users\a
t.py"
Tags are Balanced
```

Unbalanced Output

```
Test.html > html > body > p
  1   <!DOCTYPE html>
  2   <html>
  3   <head>
  4     <title>Simple Page</title>
  5   </head>
  6   <body>
  7     <h1>Hello, World!</h1>
  8     <p>This is a minimal HTML example.</p>
  9   </html>
 10
```

```
PS C:\Users\adish\OneDrive\Desktop\Python Projects> python -
t.py"
Tags are Unbalanced
```

6. **Implement a function with signature transfer(S,T). This function transfers all elements from Stack S to Stack T. The sequence of elements in T should be the same as that of S.**

Code:

```
class Stack:
    def __init__(self): #, max_size):
        # self.max_size = max_size
        self.stack = []
    def push(self, item):
        # if not self.is_full():
            self.stack.append(item)
```

Name: Adish Shanbhag
Reg no: 251100670036

```
        # else:
        #     raise OverflowError("stack overflow")

    def pop(self):
        if self.is_empty():
            print("stack underflow - no elements found")
            return None
        value = self.stack.pop()
        return value


    def peek(self):
        if self.is_empty():
            return None
        return self.stack[-1]


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

    def is_full(self):
        return len(self.stack) >= self.max_size

    def copy_Stack(self, stack1, stack2):
        temp = stack1.pop()
        if len(stack1.stack) >0 :
            stack2 = self.copy_Stack(stack1,stack2)
        stack2.push(temp)
        return stack2

S = Stack()
T =Stack()

S.push(10)
S.push(20)
S.push(30)
S.push(40)

print("S before copying elements to T: \n", format(S.stack))
print("T before copying elements from S: \n", format(T.stack))
print("Copying...")

S.copy_Stack(S,T)
```

Name: Adish Shanbhag
Reg no: 251100670036

```
print("S after copying elements to T: \n", format(S.stack))
print("T after copying elements from S: \n", format(T.stack))
```

Output:

```
PS C:\Users\adish\OneDrive\Desktop\Python Projects> python -u "
t.py"
S before copying elements to T:
 [10, 20, 30, 40]
T before copying elements from S:
 []
Copying...
S after copying elements to T:
 []
T after copying elements from S:
 [10, 20, 30, 40]
PS C:\Users\adish\OneDrive\Desktop\Python Projects>
```

7. **Implement “Forward” and “Back” buttons of browser using Stacks. Elements need to be stored are URLs.**

Code:

```
import re

class Stack:
    def __init__(self): #, max_size):
        # self.max_size = max_size
        self.stack = []
    def push(self, item):
        # if not self.is_full():
            self.stack.append(item)
        # else:
        #     raise OverflowError("stack overflow")

    def pop(self):
        if self.is_empty():
            print("stack underflow - no elements found")
            return None
        value = self.stack.pop()
        return value


    def peek(self):
        if self.is_empty():
            return None
        return self.stack[-1]
```

Name: Adish Shanbhag
Reg no: 251100670036

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

    def is_full(self):
        return len(self.stack) >= self.max_size
    def pop_All(self):
        while(len(self.stack)>0):
            self.stack.pop()


back = Stack()
forward = Stack()
currently_Viewing = Stack()

while(1):
    print("Choose one of the options below: ")
    print("1. New Window \n2. Back \n3. Forward \n4. New URL\n5. Exit")
    choice = int(input())
    match choice:
        case 5:
            exit
        case 1:
            currently_Viewing.pop_All()
            back.pop_All()
            forward.pop_All()
            url = input("Enter the url of the website")
            currently_Viewing.push(url)
        case 2:
            if back.is_empty():
                print("No web pages opened previously!")
            else:
                if not currently_Viewing.is_empty():
                    forward.push(currently_Viewing.pop())
                currently_Viewing.push(back.pop())
        case 3:
            if forward.is_empty():
                print("This is the last page. No more pages further!")
            else:
                if not currently_Viewing.is_empty():
                    back.push(currently_Viewing.pop())
                currently_Viewing.push(forward.pop())
```

Name: Adish Shanbhag
Reg no: 251100670036

```
        case 4:
            url = input("Enter the url of the website")
            back.push(currently_Viewing.pop())
            currently_Viewing.push(url)
    if not currently_Viewing.is_empty():
        print("Currently viewing : ", format(currently_Viewing.peek()))
```

Output:

[Results]

8. **Modify Q5 such that HTML tags may contain attributes along with tag name.**

Code:

```
import re
class Stack:
    def __init__(self): #, max_size):
        # self.max_size = max_size
        self.stack = []


    def push(self, item):
        # if not self.is_full():
            self.stack.append(item)
        # else:
        #    raise OverflowError("stack overflow")

    def pop(self):
        if self.is_empty():
            print("stack underflow - no elements found")
            return None
        value = self.stack.pop()
        return value

    def peek(self):
        if self.is_empty():
            return None
        return self.stack[-1]
```

Name: Adish Shanbhag
Reg no: 251100670036

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

    def is_full(self):
        return len(self.stack) >= self.max_size

    def tag_Balancer(self, Tags):
        for tag in Tags:
            temp = tag.strip("<>").split()[0]
            if tag.find("/") == -1:
                stack.push(temp)
            else:
                temp2 = stack.peek()
                if temp2 == temp.lstrip("/"):
                    stack.pop()
                else:
                    return False
        return True




try:
    f = open("Test.html", "r")
    content = f.read()
except FileNotFoundError as e:
     print(e)
stack = Stack()


tags = re.findall(r"<[^>]+>", content)
cleaned_Tags = []
for tag in tags:
    if "DOCTYPE" in tag:
        continue
    cleaned_Tags.append(tag)


if stack.tag_Balancer(cleaned_Tags):
    print("Tags are Balanced")
else:
    print("Tags are Unbalanced")
```

Name: Adish Shanbhag
Reg no: 251100670036

Output:

Balanced output:

Test.html > html

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Simple Page</title>
</head>
<body>
  <header class="main-header">
    <h1 id="title">Welcome to My Website</h1>
  </header>

  <main>
    <section id="intro" class="section">
      <p style="color: blue;">This is a <strong>sample</strong> HTML page with attributes.</p>
    </section>

    <section id="contact" class="section">
      <a href="mailto:someone@example.com" target="_blank">Contact Us</a>
    </section>
  </main>

  <footer class="footer">
    <p>&copy; 2025 My Website</p>
  </footer>
</body>
</html>
```

```
PS C:\Users\adish\OneDrive\Desktop\Python Projects> python -u
t.py"
Tags are Balanced
```

Unbalanced output:

Test.html > html > head > body

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Simple Page</title>
<body>
  <header class="main-header">
    <h1 id="title">Welcome to My Website</h1>
  </header>

  <main>
    <section id="intro" class="section">
      <p style="color: blue;">This is a <strong>sample</strong> HTML page with attributes.</p>
    </section>

    <section id="contact" class="section">
      <a href="mailto:someone@example.com" target="_blank">Contact Us</a>
    </section>
  </main>

  <footer class="footer">
    <p>&copy; 2025 My Website</p>
  </footer>
</body>
</html>
```

```
PS C:\Users\adish\OneDrive\Desktop\Python Projects> python -u
t.py"
Tags are Unbalanced
```

Name: Adish Shanbhag
Reg no: 251100670036

Queue Assignments

1. **Implement “Simple Queue” using list data structure.**

Code:

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

    def enQueue(self, item):
        self.queue.append(item)

    def deQueue(self):
        if self.is_empty():
            print("No elements in queue to pop")
            return None
        return self.queue.pop(0)

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

queue = Queue()
queue.enQueue(10)
queue.enQueue(20)
queue.enQueue(30)
queue.enQueue(40)
print("Elements of the queue are: \n", format(queue.queue))
print("Popped element:", format(queue.deQueue()))
```

Output:

```
PS C:\Users\adish\OneDrive\Desktop\Python Projects> python -u "c
ignments.py"
Elements of the queue are:
 [10, 20, 30, 40]
Popped element: 10
```

2. **Modify Q1 such that Simple Queue can contain a limited amount of elements.**

Name: Adish Shanbhag
Reg no: 251100670036

Code:

```
class Queue:
    def __init__(self, Max_Size):
        self.Max_Size = Max_Size
        self.queue = []

    def enQueue(self, item):
        if not self.is_Full():
            self.queue.append(item)
        else:
            raise OverflowError("Queue overflow while trying to insert
{}".format(item))
    def is_Full(self):
        return len(self.queue) >= self.Max_Size
    def deQueue(self):
        if self.is_empty():
            print("No elements in queue to pop")
            return None
        return self.queue.pop(0)

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

queue = Queue(4)
queue.enQueue(10)
queue.enQueue(20)
queue.enQueue(30)
queue.enQueue(40)
print("Elements of the queue are: \n", format(queue.queue))
try:
    queue.enQueue(50)
except OverflowError as e:
    print(e)
print("Popped element:", format(queue.deQueue()))
```

Output:

```
PS C:\Users\adish\OneDrive\Desktop\Python Projects> python -u "
ignments.py"
Elements of the queue are:
 [10, 20, 30, 40]
Queue overflow while trying to insert 50
Popped element: 10
```

Name: Adish Shanbhag
Reg no: 251100670036

3. **Implement “FlexiQueue” with capacity to expand and shrink based on elements to be added or deleted.**

Code:

```
class FlexiQueue:
    def __init__(self, capacity=2):
        self.capacity = capacity
        self.queue = []

    def enQueue(self, item):
        if not self.is_Full():
            self.queue.append(item)
        else:
            print("Capacity doubled")
            self.capacity *= 2
            self.queue.append(item)    # insert AFTER doubling

    def is_Full(self):
        return len(self.queue) >= self.capacity

    def deQueue(self):
        if self.is_empty():
            print("No elements in queue to pop")
            return None

        item = self.queue.pop(0)  # remove first element

        if len(self.queue) <= self.capacity // 4 and self.capacity > 1:
            print("Size reduced to half")
            self.capacity //= 2

        return item

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


queue = FlexiQueue(4)
queue.enQueue(10)
queue.enQueue(20)
queue.enQueue(30)
```

Name: Adish Shanbhag
Reg no: 251100670036

```
queue.enQueue(40)
print("Elements of the queue are: \n", format(queue.queue))
queue.enQueue(50)
print("Popped element:", format(queue.deQueue()))
print(queue.deQueue())
print(queue.deQueue())
print(queue.queue)
```

Output:

```
PS C:\Users\MSIS\251100670036> & C:/Users/MSIS
Elements of the queue are:
 [10, 20, 30, 40]
Capacity doubled
Popped element: 10
20
Size reduced to half
30
[40, 50]
```

4. Implement Stack using two Queues

Code:

```
class StackUsingQueues:
    def __init__(self):
        self.queue1 = []
        self.queue2 = []

    def push(self, item):


        self.queue2.append(item)
        while len(self.queue1)>0:
            self.queue2.append(self.queue1.pop(0))
        self.swap_Queues()

    def pop(self):
        if self.is_empty():
            print("No elements in queue to pop")
            return None
        return self.queue1.pop(0)

    def is_empty(self):
        return len(self.queue1) == 0
```

Name: Adish Shanbhag
Reg no: 251100670036

```
    def swap_Queues(self):
            self.queue1, self.queue2  = self.queue2, self.queue1

    def top(self):
        if self.is_empty():
            print("Stack is empty")
            return None
        return self.queue1[0]

s = StackUsingQueues()
s.push(10)
s.push(20)
s.push(30)


print("Top:", s.top())
print("Pop:", s.pop())
print("Top:", s.top())
```

Output:

```
PS C:\Users\MSIS\Desktop\251100670036> & C:/Users/
SIS/Desktop/251100670036/assignment.py
Top: 30
Pop: 30
Top: 20
PS C:\Users\MSIS\Desktop\251100670036>
```

5. Implement Queue using two Stacks

Code:

```
class QueueUsingStacks:
    def __init__(self):
        self.stack1 = []
        self.stack2 = []

    def enQueue(self, item):
        while self.stack1:
            self.stack2.append(self.stack1.pop())

        self.stack1.append(item)

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

Name: Adish Shanbhag
Reg no: 251100670036

```
    def deQueue(self):
        if self.is_empty():
            print("Queue underflow - no elements found")
            return None
        return self.stack1.pop()
    def first(self):
        if self.is_empty():
            return None
        return self.stack1[-1]

    def is_empty(self):
        return len(self.stack1) == 0

s = QueueUsingStacks()
s.enQueue(10)
s.enQueue(20)
s.enQueue(30)
s.enQueue(40)
print(s.first())
s.deQueue()
print(s.first())
s.deQueue()
print(s.first())
```

Output:

```
PS C:\Users\MSIS\Desktop\251100670036> & C:/Users/MSIS/Appl
10
20
30
PS C:\Users\MSIS\Desktop\251100670036>
```

6. **Assume that we have a Queue with some elements. Write method rotate() which added the existing elements in the reverse order.**

Code:

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

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

Name: Adish Shanbhag
Reg no: 251100670036

```
    def deQueue(self):
        if self.is_empty():
            print("No elements in queue to pop")
            return None
        return self.queue.pop(0)

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

    def reverse_Queue(self):
        temp_stack = []
        while len(self.queue) > 0:
           temp_stack.append(self.queue.pop(0))
        while len(temp_stack) > 0:
            self.queue.append(temp_stack.pop())
s = Queue()
print("Before reversing: ")
s.enQueue(10)
s.enQueue(20)
s.enQueue(30)
s.enQueue(40)
print(s.queue)
print("After reversing: ")
s.reverse_Queue()
print(s.queue)
```

Output:

```
PS C:\Users\adish\OneDrive\Desktop\Python Projects> python -u
t2.py"
Before reversing:

[10, 20, 30, 40]
After reversing:

[40, 30, 20, 10]
PS C:\Users\adish\OneDrive\Desktop\Python Projects>
```

7. **Implement findMax() method, which return the maximum value of element present in the queue. After finding maximum element, queue content should be same as original.**

Name: Adish Shanbhag
Reg no: 251100670036

Code:

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

    def enQueue(self, item):
        self.queue.append(item)

    def deQueue(self):
        if self.is_empty():
            print("No elements in queue to pop")
            return None
        return self.queue.pop(0)

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

    def findMax(self):
        # method 1
        # return max(self.queue)

        # method 2
        size = len(self.queue)
        max_ele= -1
        i=0
        while(i<size):
            if self.queue[i]>max_ele:
                max_ele = self.queue[i]
            i+=1
        return max_ele



s = Queue()
s.enQueue(10)
s.enQueue(20)
s.enQueue(30)
s.enQueue(40)
print(s.queue)
print("Maximum element is: {}".format(s.findMax()))
```

Output:

Name: Adish Shanbhag
Reg no: 251100670036

```
PS C:\Users\adish\OneDrive\Desktop\Python Projects> python
t2.py"
[10, 20, 30, 40]
Maximum element is: 40
PS C:\Users\adish\OneDrive\Desktop\Python Projects>
```