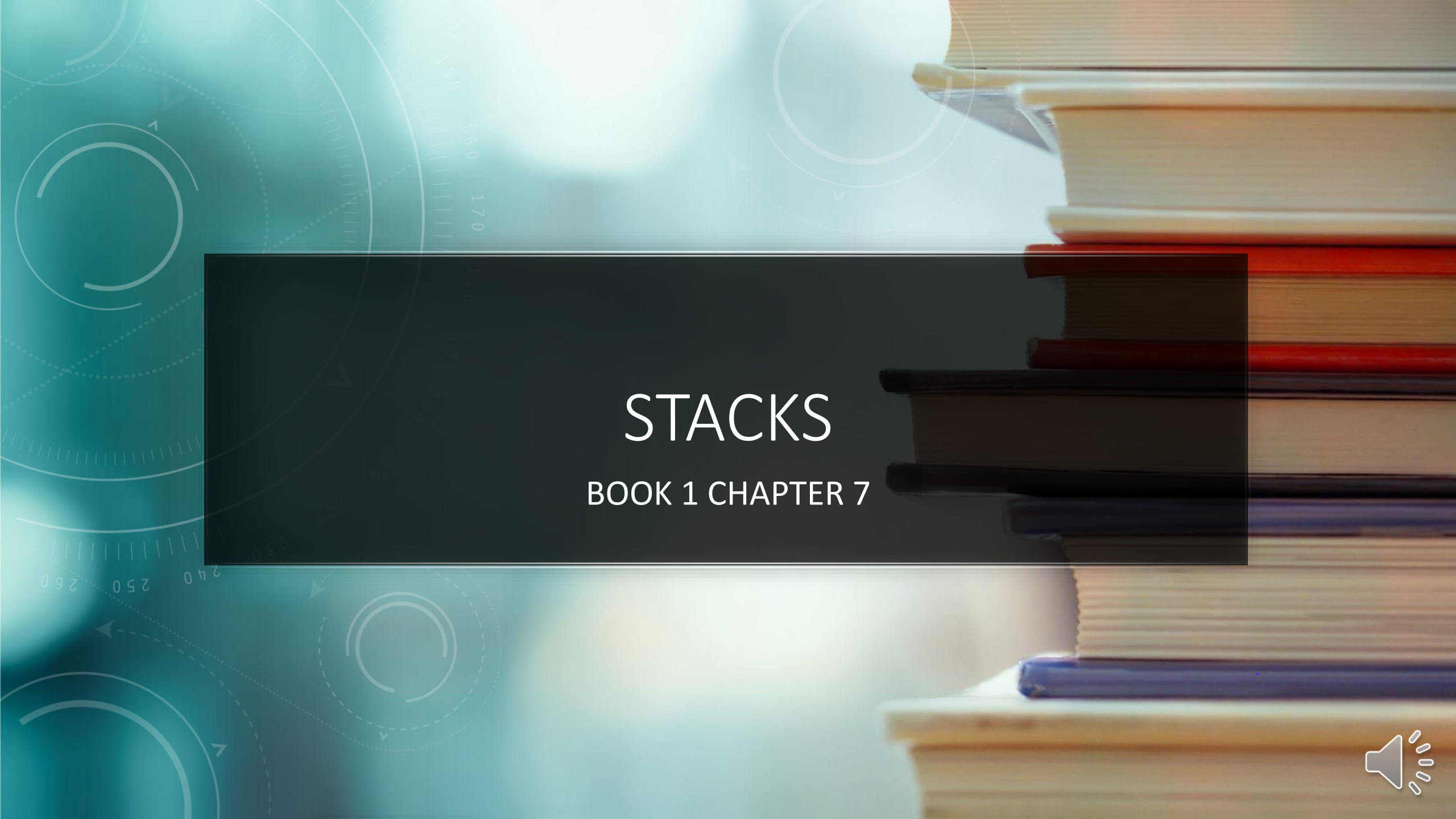
CS-218 DATA STRUCTURES AND ALGORITHMS

LECTURE 13

BY UROOJ AINUDDIN

STACKS

BOOK 1 CHAPTER 7



STACKS

A stack is used to store data such that the last item inserted is the first item removed.

The stack is a linear data structure which implements the last-in-first-out (LIFO) protocol.

New items are added, and existing items are removed from the same end, called the **top of the stack** (TOS).

ILLUSTRATING THE USE OF A STACK

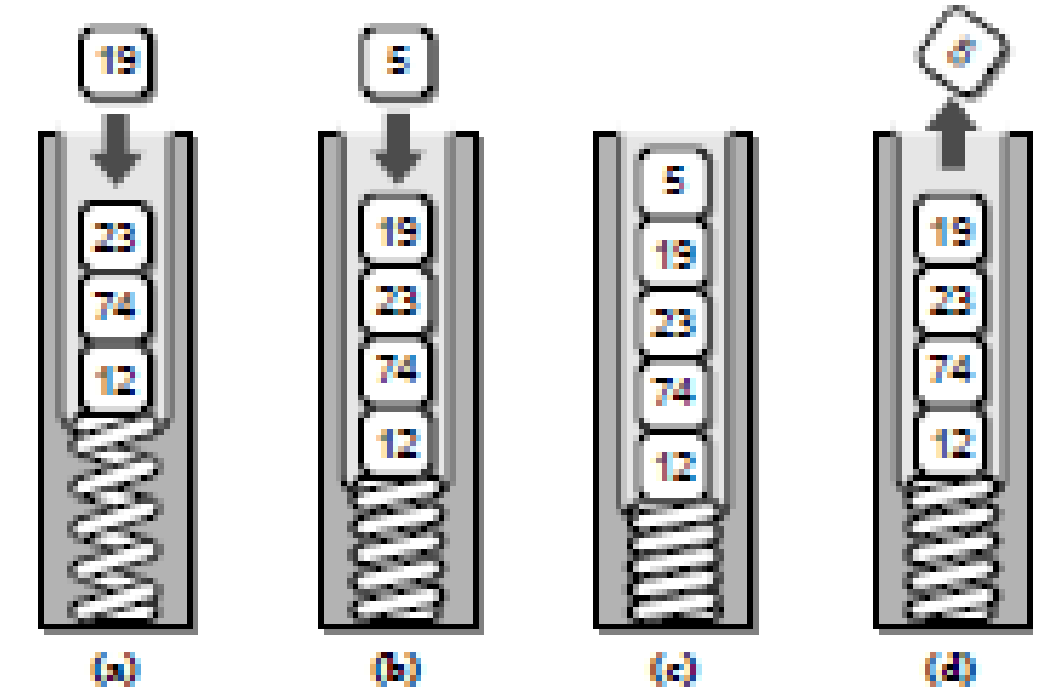


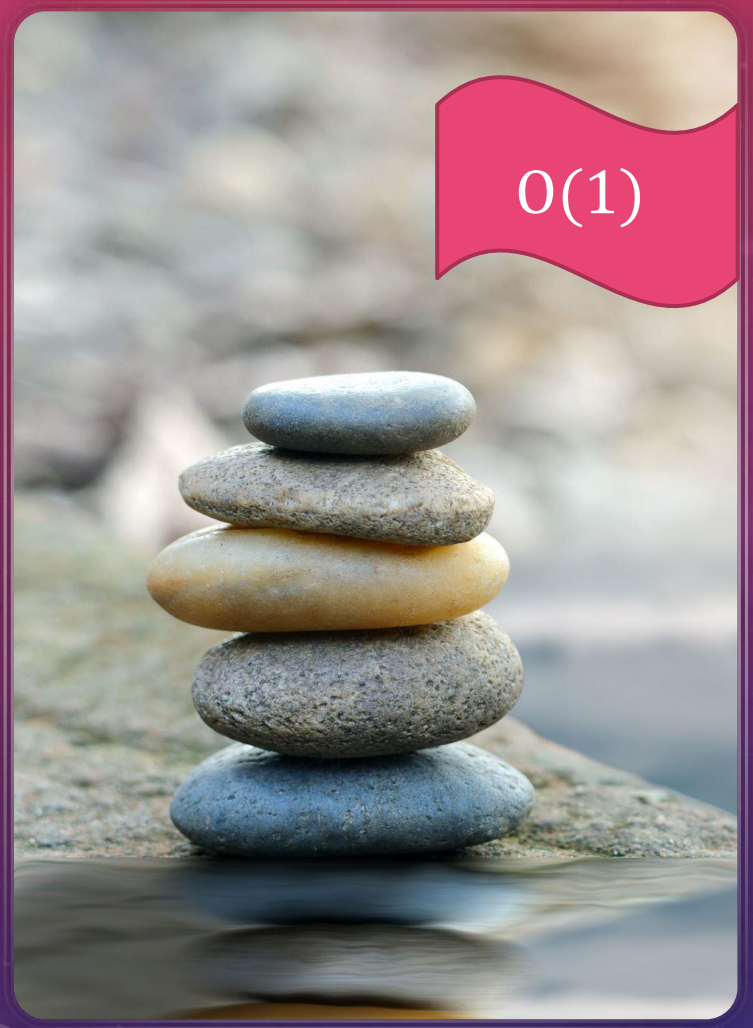
Figure 7.1: Abstract view of a stack: (a) pushing value 19; (b) pushing value 5; (c) resulting stack after 19 and 5 are added; and (d) popping top value.

THINGS WE NEVER DO WITH STACKS

- We never push data on to a full stack. This leads to **overflow**.
- We never pop data from an empty stack. This leads to **underflow**.
- We never traverse, search or sort in a stack.
- We never use the other end of the stack for insertion or deletion. Only the **top** is used at all times.

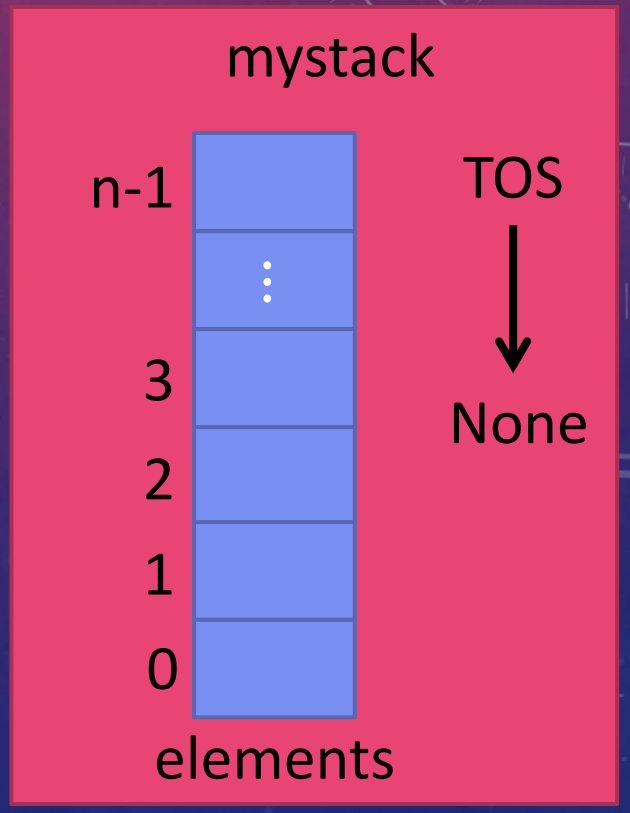
IMPLEMENTING A STACK

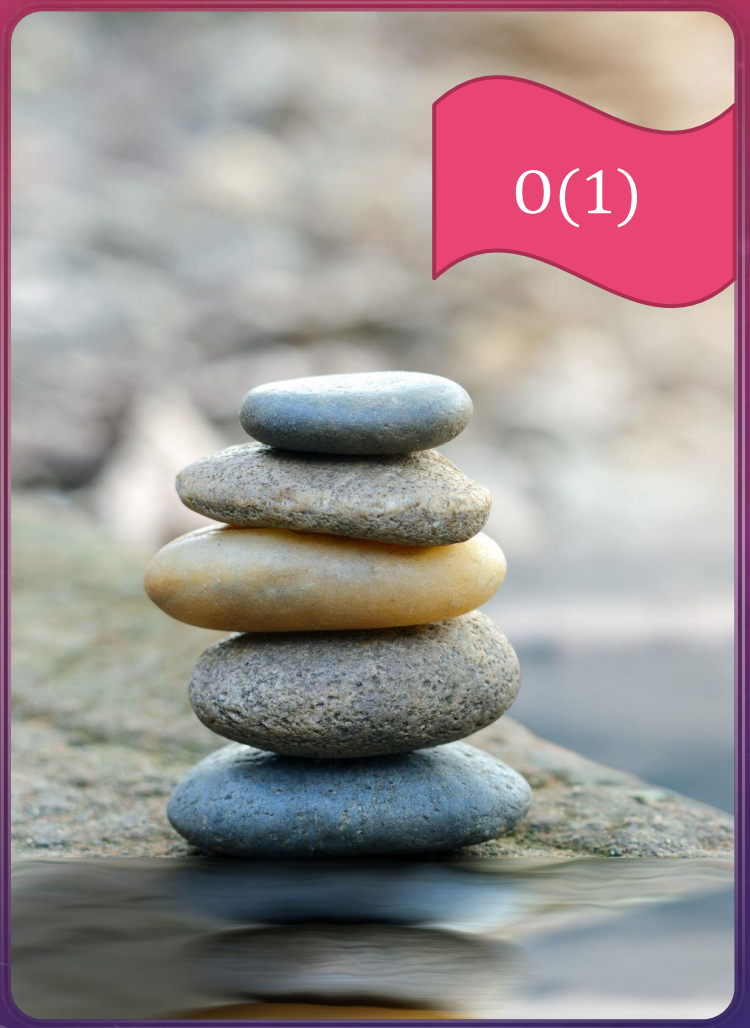
- The stack ADT can be implemented using:
  - A Python list or,
  - A linked list.
- We implement the class stack with both underlying data structures, with the same class function definitions, in different files.
- The user may change the implementation of stack by importing the relevant file into his code.
- No other changes will be required.



THE STACKWITHLIST.PY FILE – THE CLASS MYSTACK

```
class mystack:
    def __init__(self):
        self.elements = list()
```



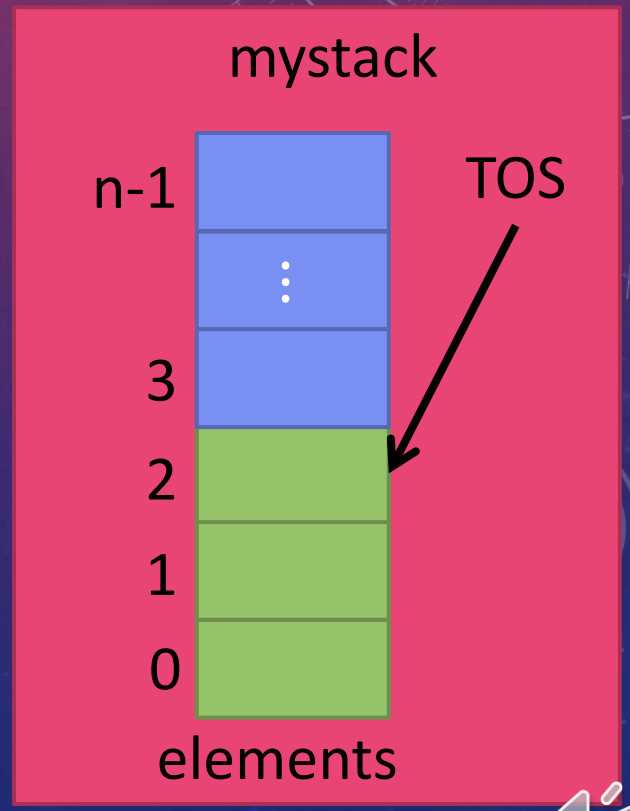


THE STACKWITHLIST.PY FILE – ISEMPTY

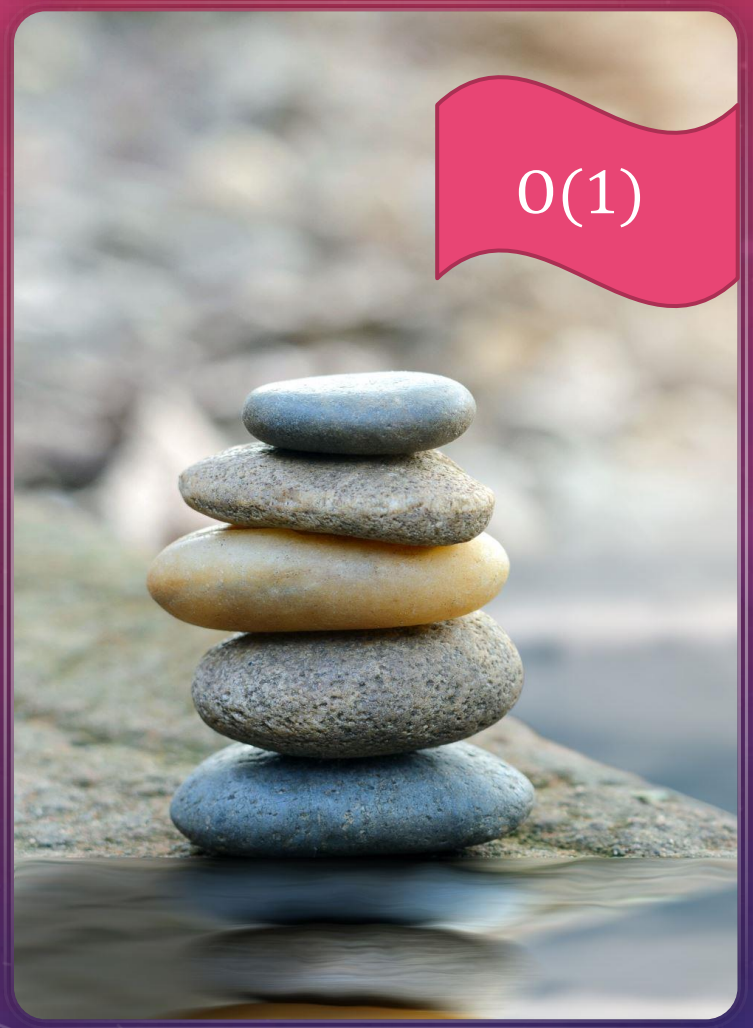
```
def isEmpty(self):
        return len(self.elements) == 0
```

THE STACKWITHLIST.PY FILE – PUSH

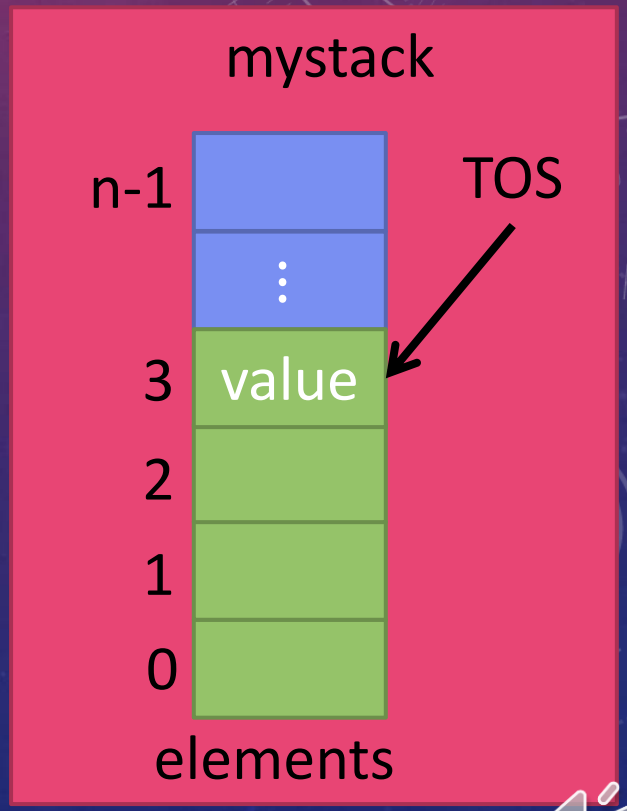
```
def push(self,value):
        self.elements.append(value)
```



THE STACKWITHLIST.PY FILE – PUSH

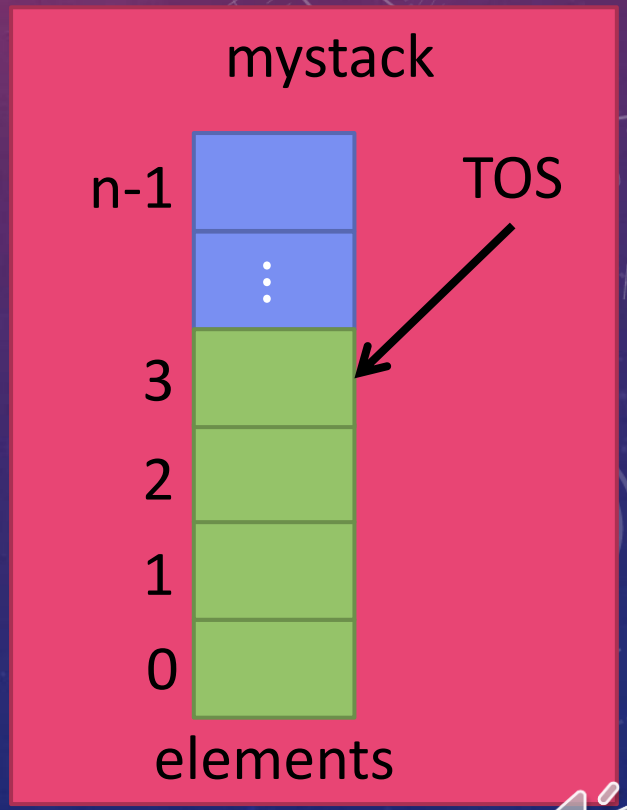



```
def push(self,value):
        self.elements.append(value)
```

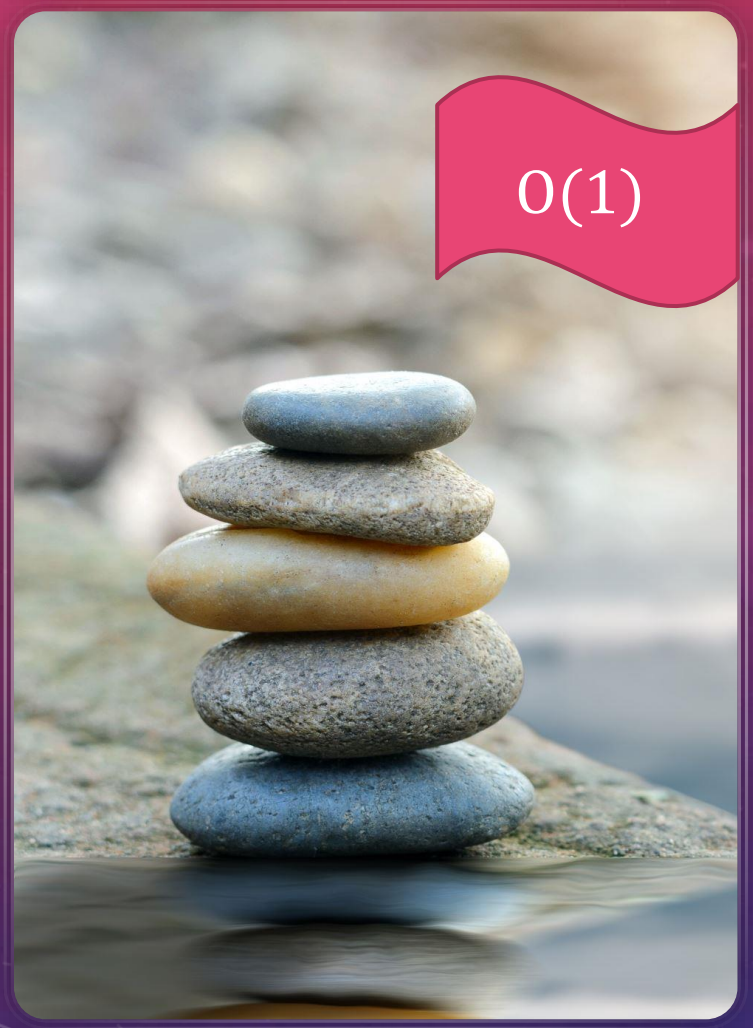


THE STACKWITHLIST.PY FILE – POP

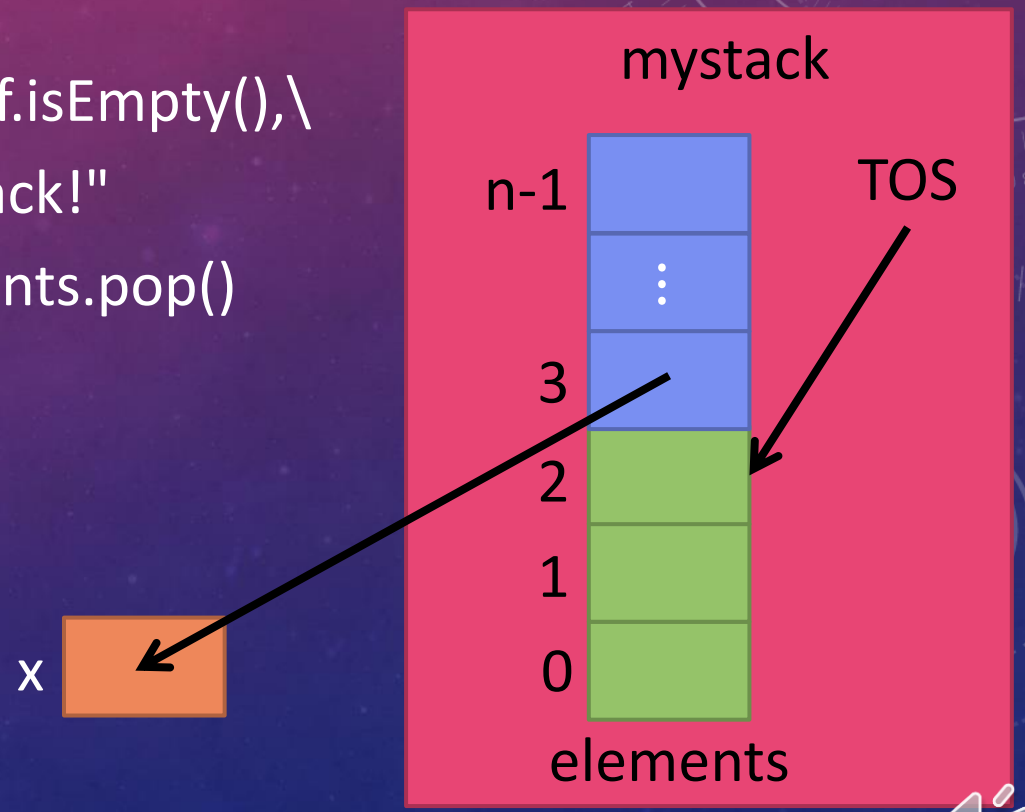
```
def pop(self):
        assert not self.isEmpty(),\
            "Empty stack!"
        x = self.elements.pop()
        return x
```



THE STACKWITHLIST.PY FILE – THE CLASS MYSTACK



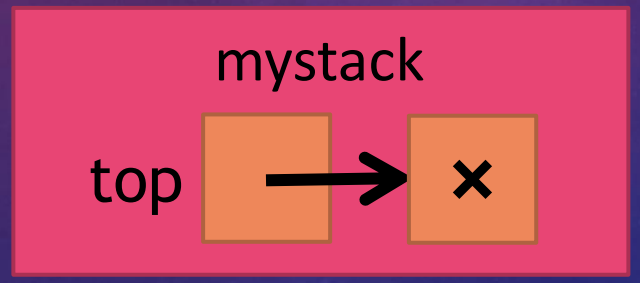

```
def pop(self):
        assert not self.isEmpty(),\
            "Empty stack!"
        x = self.elements.pop()
        return x
```



THE STACKWITHLINKEDLIST.PY FILE – THE CLASS MYSTACK

```
from singlylinkedlist import ListNode

class mystack:
    def __init__(self):
        self.top = None
```

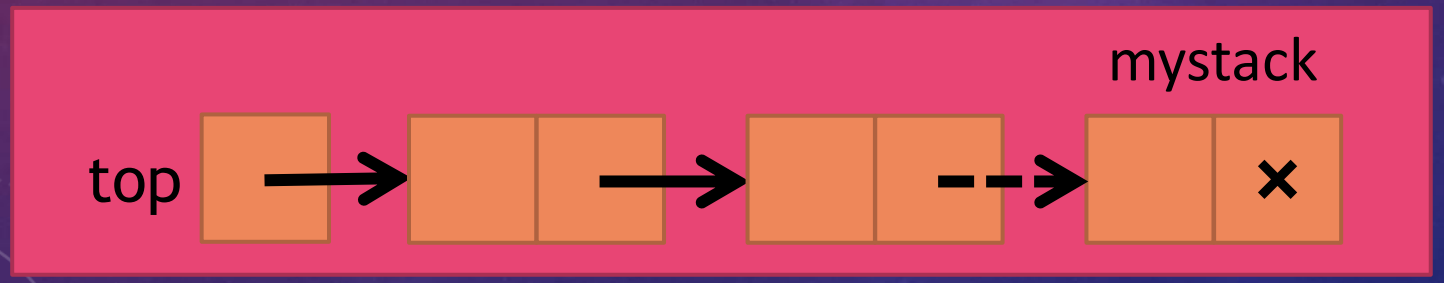


THE STACKWITHLINKEDLIST.PY FILE – ISEMPTY

```
def isEmpty(self):
        return self.top is None
```

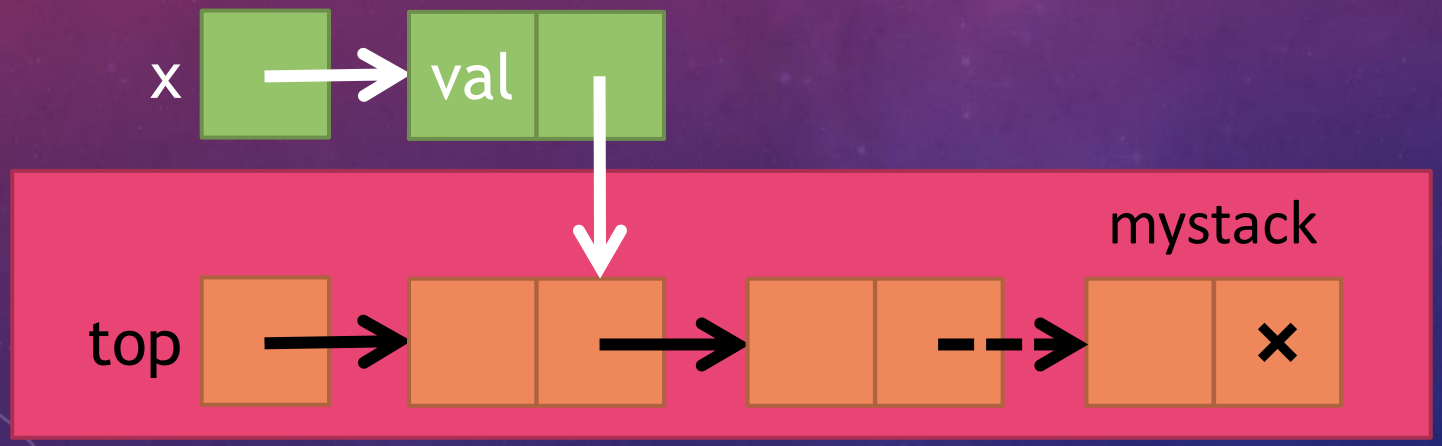
THE STACKWITHLINKEDLIST.PY FILE – PUSH

```
def push(self,val):
    x = ListNode(val)
    x.next = self.top
    self.top = x
```



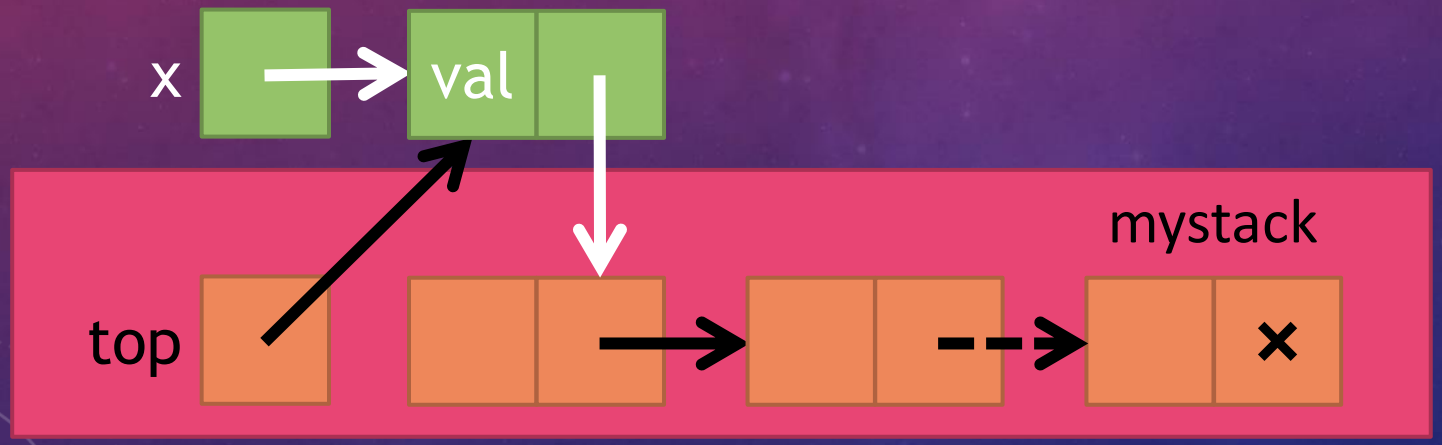
THE STACKWITHLINKEDLIST.PY FILE – PUSH

```
def push(self,val):
    x = ListNode(val)
    x.next = self.top
    self.top = x
```



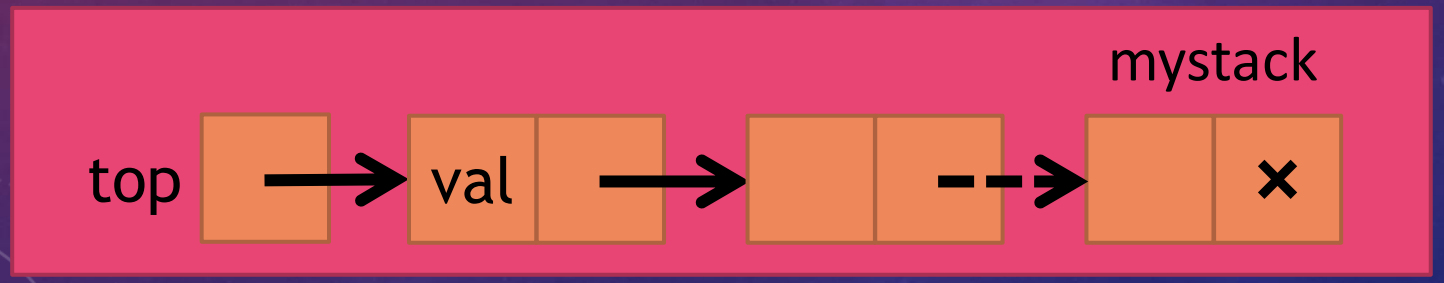
THE STACKWITHLINKEDLIST.PY FILE – PUSH

```
def push(self,val):
    x = ListNode(val)
    x.next = self.top
    self.top = x
```



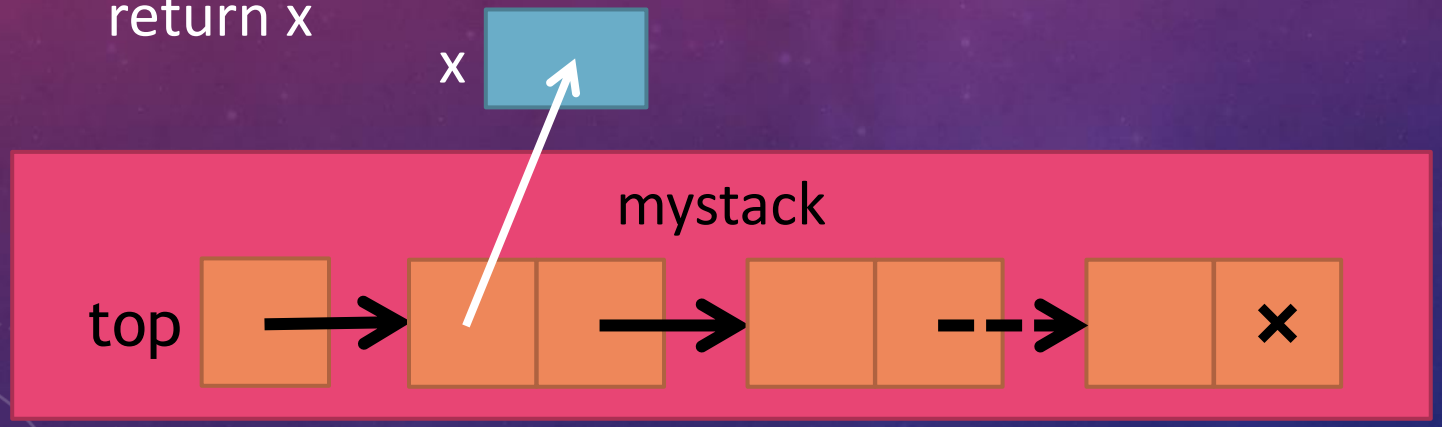
THE STACKWITHLINKEDLIST.PY FILE – PUSH

```
def push(self,val):
    x = ListNode(val)
    x.next = self.top
    self.top = x
```



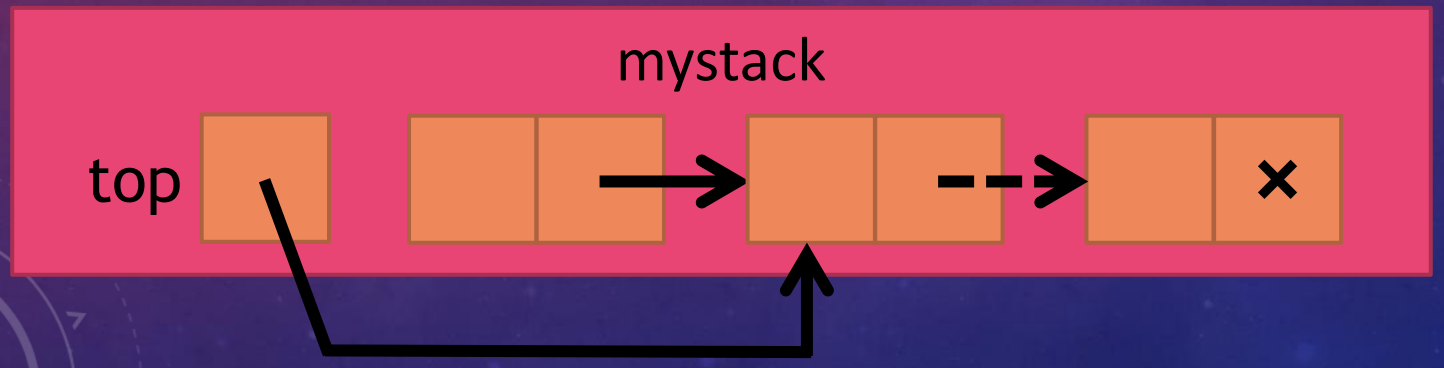
THE STACKWITHLINKEDLIST.PY FILE – POP

```
def pop(self):
    assert not self.isEmpty(),"Empty stack"
    x = self.top.data
    self.top = self.top.next
    return x
```



THE STACKWITHLINKEDLIST.PY FILE – POP

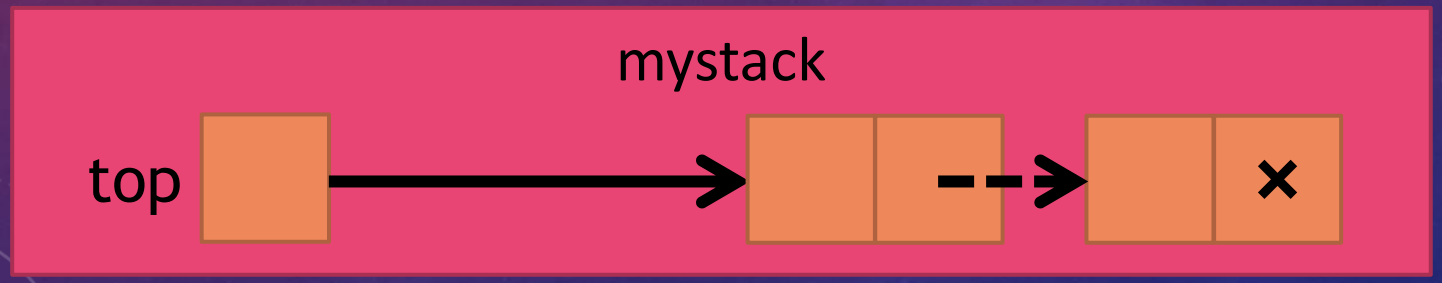
```
def pop(self):
    assert not self.isEmpty(),"Empty stack"
    x = self.top.data
    self.top = self.top.next
    return x
```



THE STACKWITHLINKEDLIST.PY FILE – POP

```
def pop(self):
    assert not self.isEmpty(),"Empty stack"
    x = self.top.data
    self.top = self.top.next
    return x
```

x



USING ONE OF THE TWO STACK IMPLEMENTATIONS

```
#from stackwithlist import mystack
from stackwithlinkedlist import mystack

sl=mystack()
print(sl.isEmpty())
mynum=[5,8,4,3,7]
for i in range(len(mynum)):
    sl.push(mynum[i])
print(sl.isEmpty())
while not sl.isEmpty():
    print(sl.pop())
print(sl.isEmpty())
```

What do you observe about the output of this code?

USES OF STACKS

- Conversion of infix expressions to postfix expressions
- Evaluation of postfix expressions
- Execution of the quicksort algorithm
- Transfer of control from one function to another, and back
- Implementation of recursion

MATHEMATICAL EXPRESSIONS

CONVERSION AND EVALUATION WITH STACKS

FORMATS FOR MATHEMATICAL EXPRESSIONS

Infix notation	The operator is mentioned between the operands.	$A + B$
Prefix notation	The operator is mentioned before the operands.	$+ A B$
Postfix notation	The operator is mentioned after the operands.	$A B +$

CONVERSION BETWEEN FORMATS

Convert the given infix expression to postfix.

a+b/(c+d)

a+b/cd+

a+bcd+/

abcd+/+

Convert the given infix expression to prefix.

a+b/(c+d)

a+b/+cd

a+/b+cd

+a/b+cd

CONVERSION BETWEEN FORMATS

Convert the given prefix expression to infix.

+/a×bcd

+/a(b×c)d

+(a/(b×c))d

(a/(b×c))+d

Convert the given postfix expression to infix.

abc×/d+

a(b×c)/d+

(a/(b×c))d+

(a/(b×c))+d

EVALUATION OF MATHEMATICAL EXPRESSION

Evaluate the given postfix expression.

$1,5,+,8,4,1,-,-,\times$

$6,8,4,1,-,-,\times$

$6,8,3,-,\times$

$6,5,\times$

30

Convert the given postfix expression to infix.

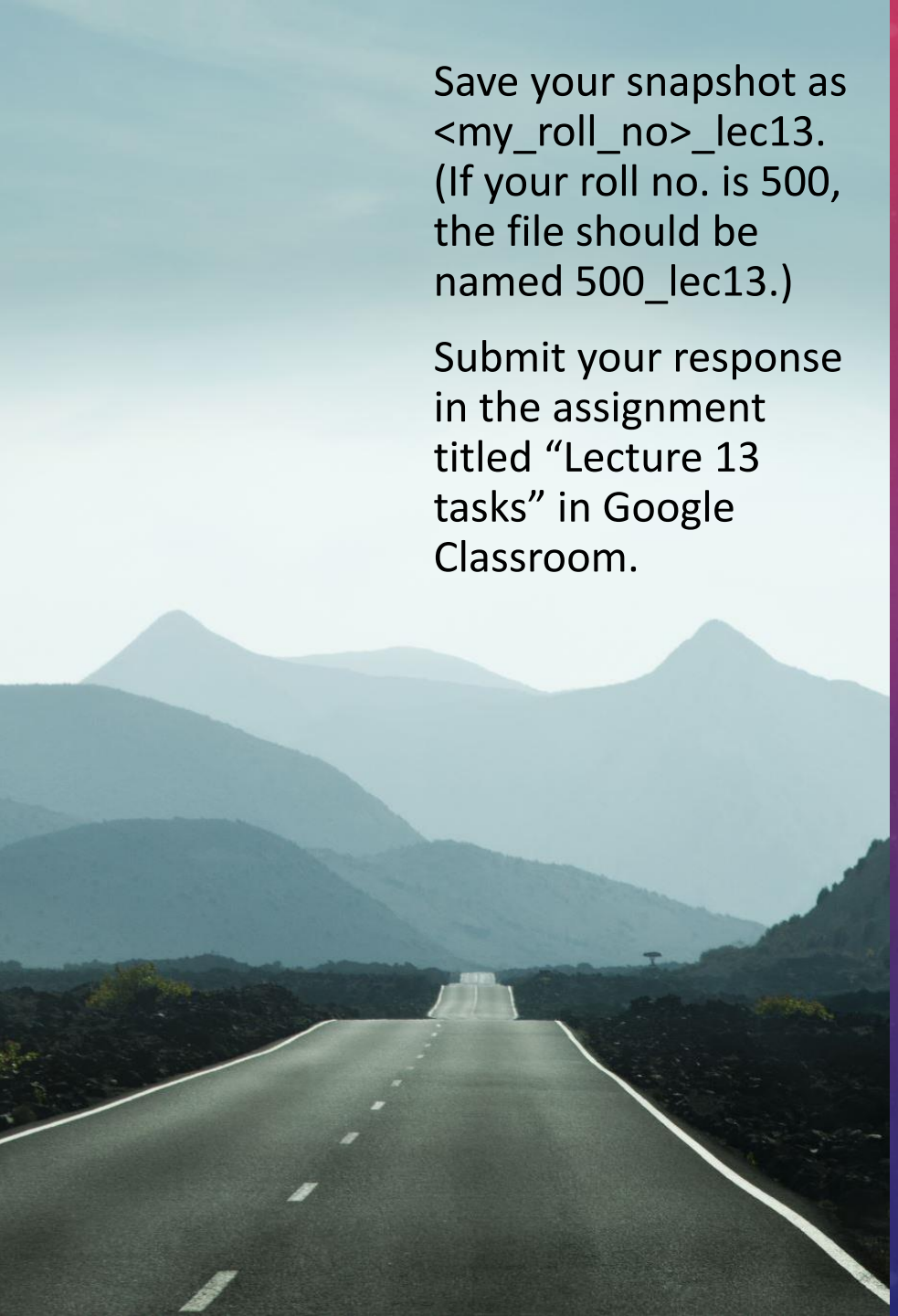
$1,5,+,8,4,1,-,-,\times$

$(1+5),8,4,1,-,-,\times$

$(1+5),8,(4-1),-,\times$

$(1+5),(8-(4-1)),\times$

$(1+5)\times(8-(4-1))$



Save your snapshot as <my_roll_no>_lec13. (If your roll no. is 500, the file should be named 500_lec13.)

Submit your response in the assignment titled “Lecture 13 tasks” in Google Classroom.

TASK

1. Convert the following infix expression to postfix and prefix:

$$8 + 9 - (5 - 2) \times ((1 - 7) + 4)/2$$

2. Evaluate the given postfix expression:

$$4,12,11,9,\times, 9, -, 6, +, 12,/, +, -$$

SO WHAT DID WE LEARN TODAY?

We were introduced to stacks.

We implemented stacks using the Python list and the singly linked list.

We listed several problems that use stacks for solution.

We learned about different formats for mathematical expressions.

We learned how to convert between formats.

We learned how to evaluate postfix expressions.

THINGS TO DO

- Read the book!
- Submit your answers to the tasks in this lecture.
- Note your questions and put them up in the relevant online session.
- Email suggestions on content or quality of this lecture at uroojain@neduet.edu.pk