Stack

CPE202

What is a stack?

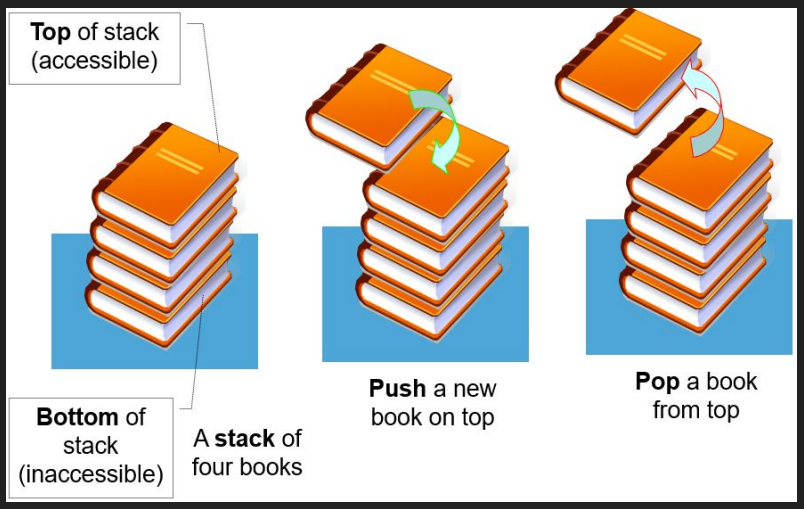
Imagine a stack of books.

LIFO

Stack allows you to process items in a LIFO fashion.

Last **I**n **F**irst **O**ut

How can you use a stack?



- Push
- Pop

Abstract Data Type (ADT)

- An Abstract Data Type is defined by its behavior (semantics) from the point of view of a user of the data.
- Stack is an abstract data type.

Stack Abstract Data Type

- push(item)
  - adds a new item to the top of the stack. It needs the item and returns nothing.

Stack Abstract Data Type

- pop()
  - removes the top item from the stack. It needs no parameters and returns the item. The stack is modified.

Stack Abstract Data Type

- peek()
  - returns the top item from the stack but does not remove it. It needs no parameters. The stack is not modified.

Stack Abstract Data Type

- is_empty()
  - tests to see whether the stack is empty. It needs no parameters and returns a boolean value.

Stack Abstract Data Type

- size()
  - returns the number of items on the stack. It needs no parameters and returns an integer.

Stack Abstract Data Type

- stack()
  - A means to create a new stack that is empty. Ex. Constructor.

Stack Implementation

```
class Stack:
    def __init__(self):
        #write your code here

    def is_empty(self):
        #write your code here

    def push(self, item):
        #write your code here
```

```
...
def pop(self):
    #write your code here

def peek(self):
    #write your code here

def size(self):
    #write your code here
```

Stack in action

Stack Operation	Stack Contents	Return Value
s.is_empty()	[ ]	True
s.push(4)	[4]	
s.push('dog')	[4,'dog']	
s.peek()	[4,'dog']	'dog'
s.size()	[4,'dog']	2
s.is_empty()	[4, ‘dog’]	False
s.pop()	[4]	‘dog’
s.pop()	[ ]	4

Applications

- Supporting recursion
  - compile time call stack

Applications

- Supporting recursion
  - compile time call stack
- Reversing the order of items

Applications

- Supporting recursion
  - compile time call stack
- Reversing the order of items
- Undoing Something
- Backtracking

Applications

- Supporting recursion
  - compile time call stack
- Reversing the order of items
- Undoing Something
- Backtracking
- Syntax checking
  - checking if parentheses are balanced

What are stacks good for?

- Keeping track of recursive calls.

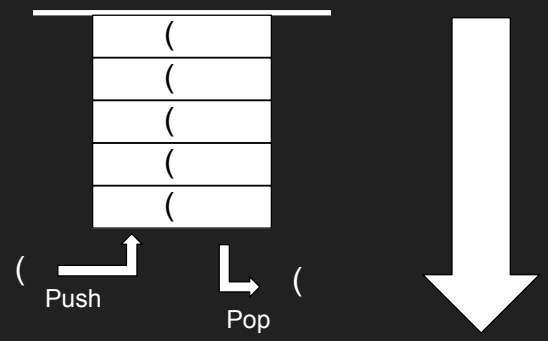
```
def sum(int_list):
    if len(int_list) == 0:
        return 0
    return int_list[0] + sum(int_list[1:])

s = sum([1,3,5,7,9])
```

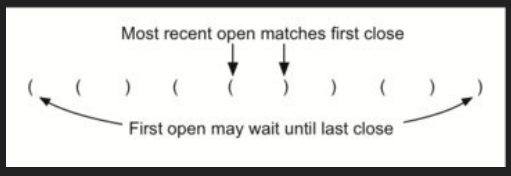
callee	calls	returns
sum([1,3,5,7,9])	1 + sum([3,5,7,9])	25
sum([3,5,7,9])	3 + sum([5,7,9])	24
sum([5,7,9])	5 + sum([7,9])	21
sum([7,9])	7 + sum([9])	16
sum([9])	9 + sum([])	9
sum([])		0

What are stacks good for?

- To check if opening parentheses match closing parentheses. ((((( )( )))))



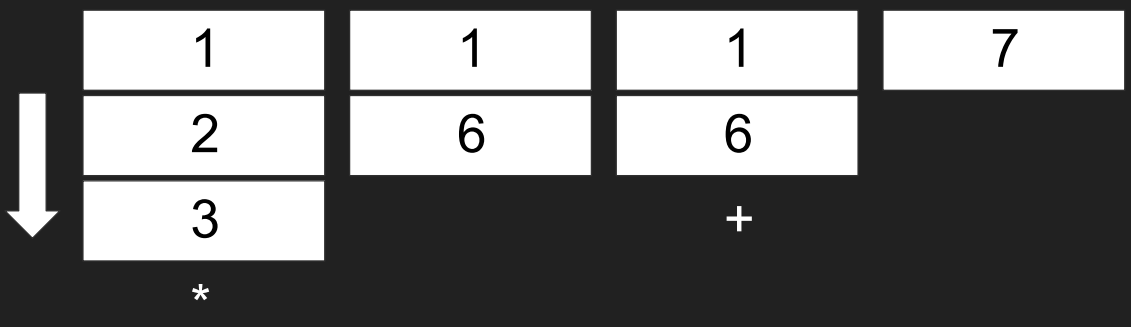
Check Balanced Parentheses




```
from my_stack import Stack
def par_checker(symbol_string):
    s = Stack()
    balanced = True
    index = 0
    size = len(symbol_string)
    while index < size and balanced:
        symbol = symbol_string[index]
        if symbol == "(":
            s.push(symbol)
        elif s.is_empty():
            balanced = False
        else:
            s.pop()
        index = index + 1
    if balanced and s.is_empty():
        return True
    return False
```

Evaluating Postfix Arithmetic Expression with Stack

1 + 2 * 3 ➡ 1 2 3 * +



Main Aspects of Object Oriented Programming

- Encapsulation
  - Structured data type: one or more data bundled
  - With methods to operate on the data
  - Encapsulated
    - Restrict access to the data only through public interfaces (methods).

- Polymorphism
  - We can override the behaviors of inherited methods.
    - __repr__ returns different string representations for different classes of objects.

Summary

- Stack provides LIFO data structure
- Useful for keeping track of the order of things so that the things can be accessed in the reverse order later.
- Useful for checking if parentheses, etc. are balanced.
- Useful for evaluating postfix arithmetic expressions.