جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية التقنيات الصحية والطبية

قسم الانظمة الطبية الذكية

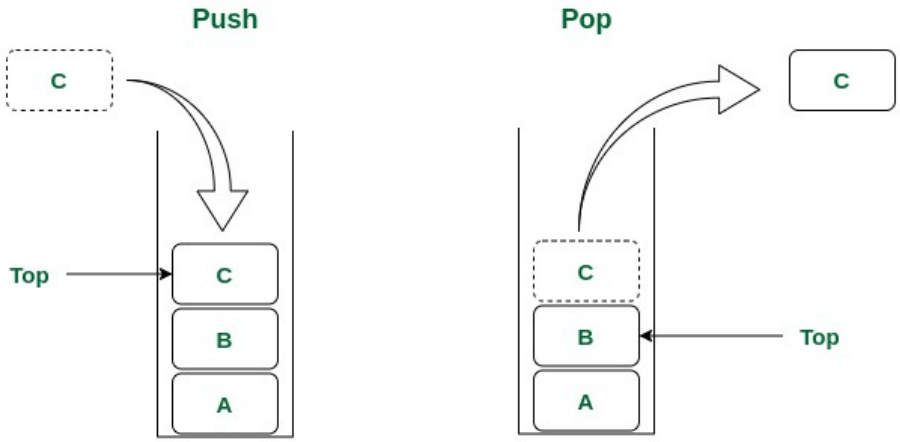
Intelligent Medical Systems Department

Lecture: (2)

Stack in Python

Subject: Data Structure Lab.
Class: Second
Lecturer: Asst. Prof. Mehdi Ebady Manaa
Asst. Lec. Sajjad Ibrahim Ismael

A **stack** is a linear data structure that stores items in a **Last-In/First-Out (LIFO)** or First-In/Last-Out (FILO) manner. In stack, a new element is added at one end and an element is removed from that end only. The insert and delete operations are often called push and pop.



The functions associated with stack are:

- **empty()** – Returns whether the stack is empty – Time Complexity: O(1)
- **size()** – Returns the size of the stack – Time Complexity: O(1)
- **top() / peek()** – Returns a reference to the topmost element of the stack – Time Complexity: O(1)
- **push(a)** – Inserts the element 'a' at the top of the stack – Time Complexity: O(1)
- **pop()** – Deletes the topmost element of the stack – Time Complexity: O(1)

Implementation:

There are various ways from which a stack can be implemented in Python.

Stack in Python can be implemented using the following ways:

- list
- Collections.deque
- queue.LifoQueue

Implementation using list:

Python's built-in data structure list can be used as a stack. Instead of push(), append() is used to add elements to the top of the stack while pop() removes the element in LIFO order.

Unfortunately, the list has a few shortcomings. The biggest issue is that it can run into speed issues as it grows. The items in the list are stored next to each other in memory,

if the stack grows bigger than the block of memory that currently holds it, then Python needs to do some memory allocations. This can lead to some append() calls taking much longer than other ones.

```
# Python program to demonstrate stack implementation using list

stack = []

# append() function to push element in the stack
stack.append('a')
stack.append('b')
stack.append('c')

print('Initial stack')
print(stack)

# pop() function to pop element from stack in LIFO order
print('\nElements popped from stack:')
print(stack.pop())
print(stack.pop())
print(stack.pop())

print('\nStack after elements are popped:')
print(stack)

# uncommenting print(stack.pop()) will cause an IndexError as the stack is now
empty
```

Output

```
Initial stack
['a', 'b', 'c']


Elements popped from stack:
c
b
a


Stack after elements are popped:
[]
```

Implementation using collections.deque:

Python stack can be implemented using the deque class from the collections module. Deque is preferred over the list in the cases where we need quicker append and pop operations from both the ends of the container, as deque provides an O(1) time complexity for append and pop operations as compared to list which provides O(n) time complexity.

The same methods on deque as seen in the list are used, append() and pop().

```
# Python program to demonstrate stack implementation using collections.deque

from collections import deque

stack = deque()

# append() function to push element in the stack
stack.append('a')
stack.append('b')
stack.append('c')

print('Initial stack:')
print(stack)

# pop() function to pop element from stack in LIFO order
print('\nElements popped from stack:')
print(stack.pop())
print(stack.pop())
print(stack.pop())

print('\nStack after elements are popped:')
print(stack)

# uncommenting print(stack.pop()) will cause an IndexError as the stack is now
empty
```

Output

```
Initial stack:
deque(['a', 'b', 'c'])


Elements popped from stack:
c
```

```
b
a

Stack after elements are popped:
deque([])
```

Implementation using queue module

Queue module also has a LIFO Queue, which is basically a Stack. Data is inserted into Queue using the put() function and get() takes data out from the Queue.

There are various functions available in this module:

- **maxsize** – Number of items allowed in the queue.
- **empty()** – Return True if the queue is empty, False otherwise.
- **full()** – Return True if there are *maxsize* items in the queue. If the queue was initialized with maxsize=0 (the default), then full() never returns True.
- **get()** – Remove and return an item from the queue. If the queue is empty, wait until an item is available.
- **get_nowait()** – Return an item if one is immediately available, else raise QueueEmpty.
- **put(item)** – Put an item into the queue. If the queue is full, wait until a free slot is available before adding the item.
- **put_nowait(item)** – Put an item into the queue without blocking. If no free slot is immediately available, raise QueueFull.
- **qsize()** – Return the number of items in the queue.

```python
# Python program to demonstrate stack implementation using queue module

from queue import LifoQueue

# Initializing a stack
stack = LifoQueue(maxsize=3)

# qsize() show the number of elements in the stack
print(stack.qsize())

# put() function to push element in the stack
stack.put('a')
stack.put('b')
stack.put('c')
```

```
print("Full: ", stack.full())
print("Size: ", stack.qsize())

# get() function to pop element from stack in LIFO order
print('\nElements popped from the stack')
print(stack.get())
print(stack.get())
print(stack.get())

print("\nEmpty: ", stack.empty())
```

Output

```
0

Full:  True

Size:  3


Elements popped from the stack

c

b

a


Empty:  True
```