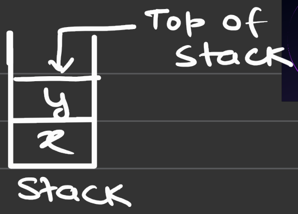
Your Ultimate Guide To Landing Top AI roles

DECODE
AiML

2.6

Stack and Queue



① Introduction to stack

→ data structure → Linear

→ It follows LIFO (Last In, First out) principle.

→ This means the last element added to the stack is the first one to be removed.

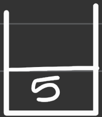
→ In Python, we implement stack data structures using List

→ key operations of stack.

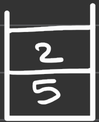
① Push → Add an element to top of stack
② Pop → Remove the element from the top of stack
③ Top/Peek → returns the top element of the stack
④ isEmpty → check if stack is Empty
⑤ Size → Check the number of element in the stack.

Example: Let's check an example

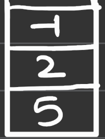
① Push(5)



② Push(2)



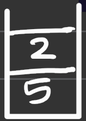
③ Push(-1)



④ top()

return -1

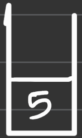
⑤ Pop()



⑥ isEmpty()

return False

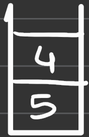
⑦ Pop()



⑧ size()

return 1

⑨ push(4)



Stack using List

→ **Initialization** ← O(1)

```
Stack = []
```

← Initialize stack using empty list

→ **Push** ← O(1) amortized.

```
Stack.append(x)
```

→ **Pop** ← O(1)

```
if len(stack) > 0:
        Stack.pop()
```

Stack using List

→ **Top** ← O(1)

```
if len(stack)>0:
      return stack[-1]
```

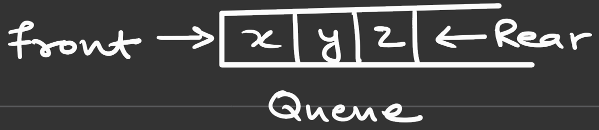
→ **IsEmpty** ← O(1)

```
return len(stack)==0
```

→ **Size** ← O(1)

```
return len(stack)
```

① Introduction to Queue



→ data structure → Linear

→ It follows FIFO (First In, First out) principle.

→ This means the first element added to the Queue is the first one to be removed.

→ In Python, we implement Queue data structures using Collections. deque

→ key operations of Queue

① Enqueue → Add an element to end of Queue
② Dequeue → Remove the element from the front of queue
③ Peek / front → returns the front element of the Queue.
④ isEmpty → check if Queue is Empty
⑤ Size → Check the number of element in the Queue

Example: Let's check an example

① enqueue(5)

5	

② enqueue(2)

5	2	

③ enqueue(-1)

5	2	-1	

④ front()

return ⑤

⑤ dequeue()

2	-1	

⑥ isEmpty()

return False

⑦ dequeue()

-1	

⑧ size()

return ①

⑨ enqueue (4)

-1	4	

Queue using Collections.deque

→ Initialization ← O(1)

```
from collections import deque
Self.queue = deque()
```

→ Enqueue ← O(1) amortized.

```
queue.append(x)
```

→ Dequeue ← O(1)

```
if len(queue) > 0:
      queue.popleft()
```

Queue using Collections.deque

→ **Front** ← O(1)

```
if len(queue)>0:
        return queue[0]
```

→ **IsEmpty** ← O(1)

```
return len(queue)==0
```

→ **Size** ← O(1)

```
return len(queue)
```

Like ▶ Subscribe