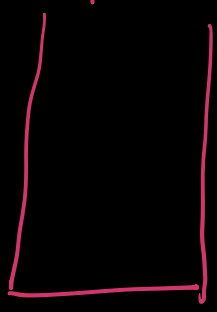
STACK → Stack of plates chairs books.

Open Container

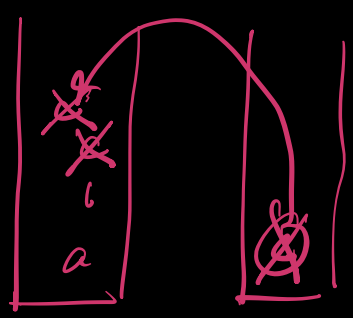


Recursion :

Application of stacks:

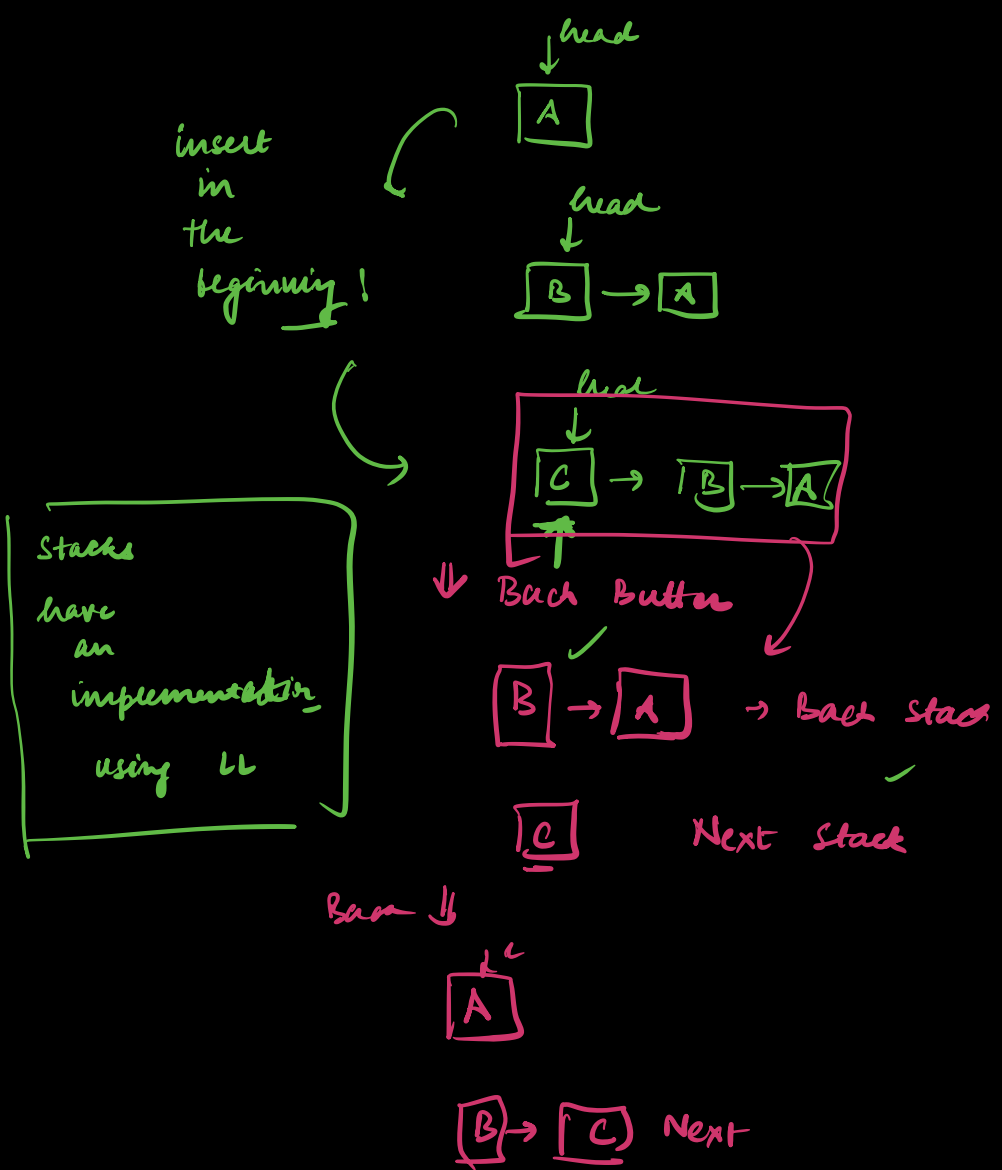
1. ~~1~~ Forward and Back button of browser
2. ~~2~~ Undo Redo
3. → 3 Parantheses checking
4. 4 Expression Evaluation.

a b c d



Exactly same as Forward and Back operation:

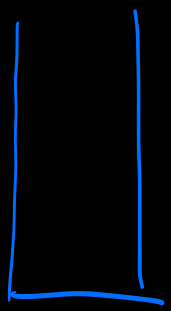
Can we implement Forward and Back using a LL



What is a difference b/w stacks and Arrays.

- Don't have Random Access
- Contiguous memory block
- Fixed size

However to use stacks in python we use lists!



→ stores Data

DS ⇒ Push ↓ means insert at top

↓ methods

Pop

Remove the top most element.

push(1) →

push(2)

push(4)

push(-1)

pop()

pop()

push(2)

top() → ? 2

pop()

top() → 2

Peek

↓

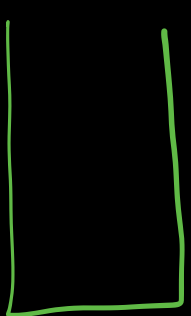
Get the topmost element

push(value)

Inst = insert_at_beginning(value)

pop()

delete_head()



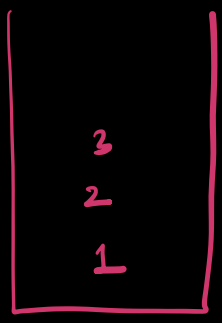
top/ peek()

return head.data

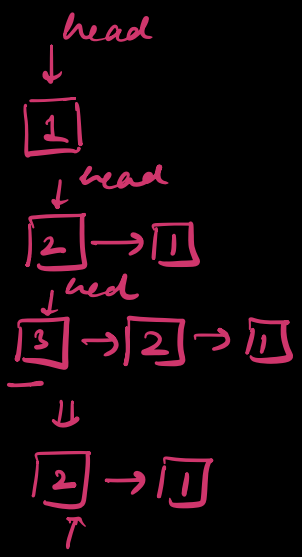
In LL

Top

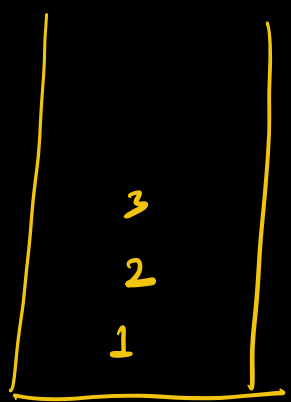
Bottom



push(1)
push(2)
push(3)
pop()



O(1)



push(1)
push(2)
push(3)
pop.

[1]
[1, 2]
[1, 2, 3]
[1, 2]

: pop()

↓

Removes the last added element

Top is represented by the last element in the list

Pop → O(1) → list.pop()

push → O(1) → list.append

top / peek → list[-1]

LIFO → Last IN First OUT

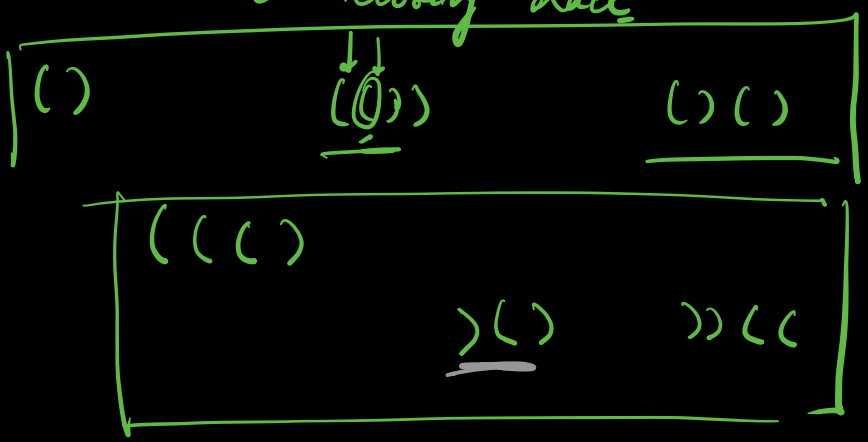
✓ List is Dynamic Array

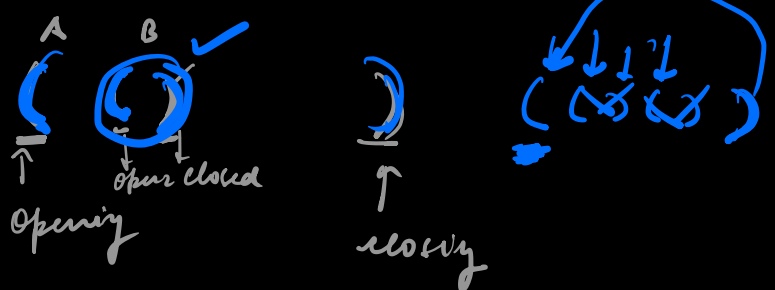
→ Applications
→ LL can be used to implement stack
→ use a list to replicate a stack

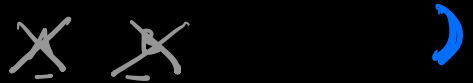
Parantheses checking :

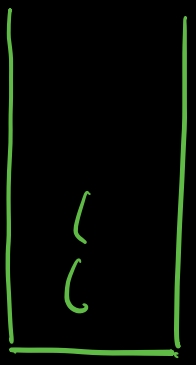
You are given a sequence of brackets
You have to tell if the sequence is balanced or not.

For every opening brace, we have a closing brace









String is balanced

If the stack is empty and we find a closing bracket → means the sequence is not balanced

```
def is_balanced( str) :
    l = [ ]   # stack.
    for e in str :
        if e == '(' :
            l. append ('(')
        else :
            if len(l) > 0 :
                l. pop ( )
            else :
                return False.
    return len(l) == 0
```

@ 2 variable
@ Diff paren

[{ (⇒ stack

At every point keep a track of open paranthesis that need to balanced.

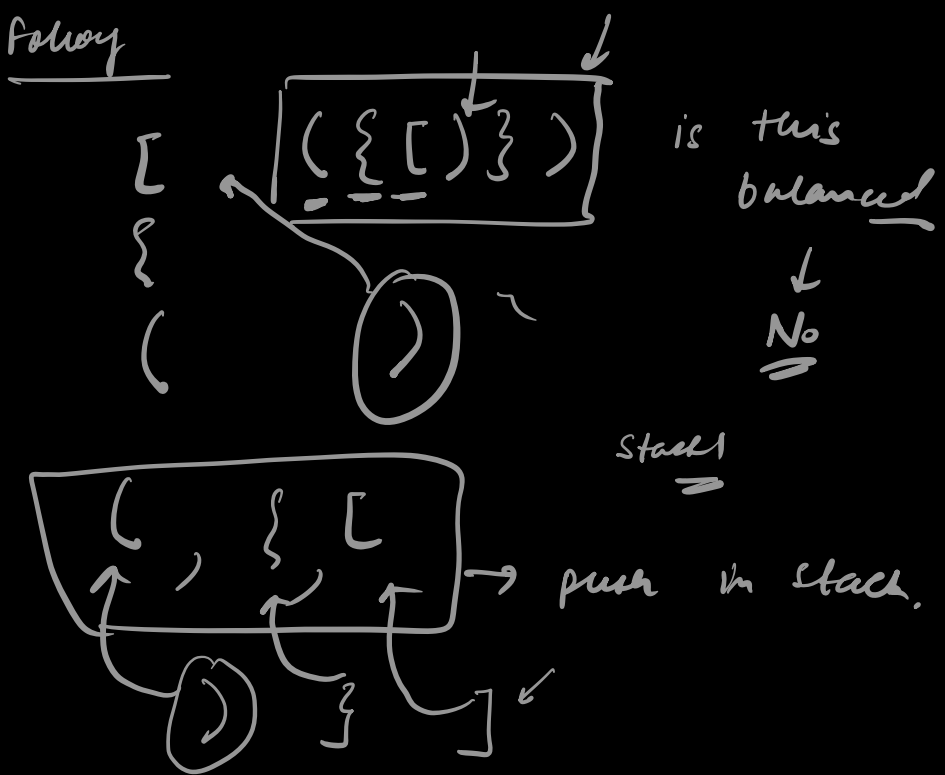
↓

for single type of paranthesis:

```
cnt = 0
if (   → cnt += 1
   )   → cnt -= 1
   if cnt < 0 → false

cnt != 0  false
```

Eg:

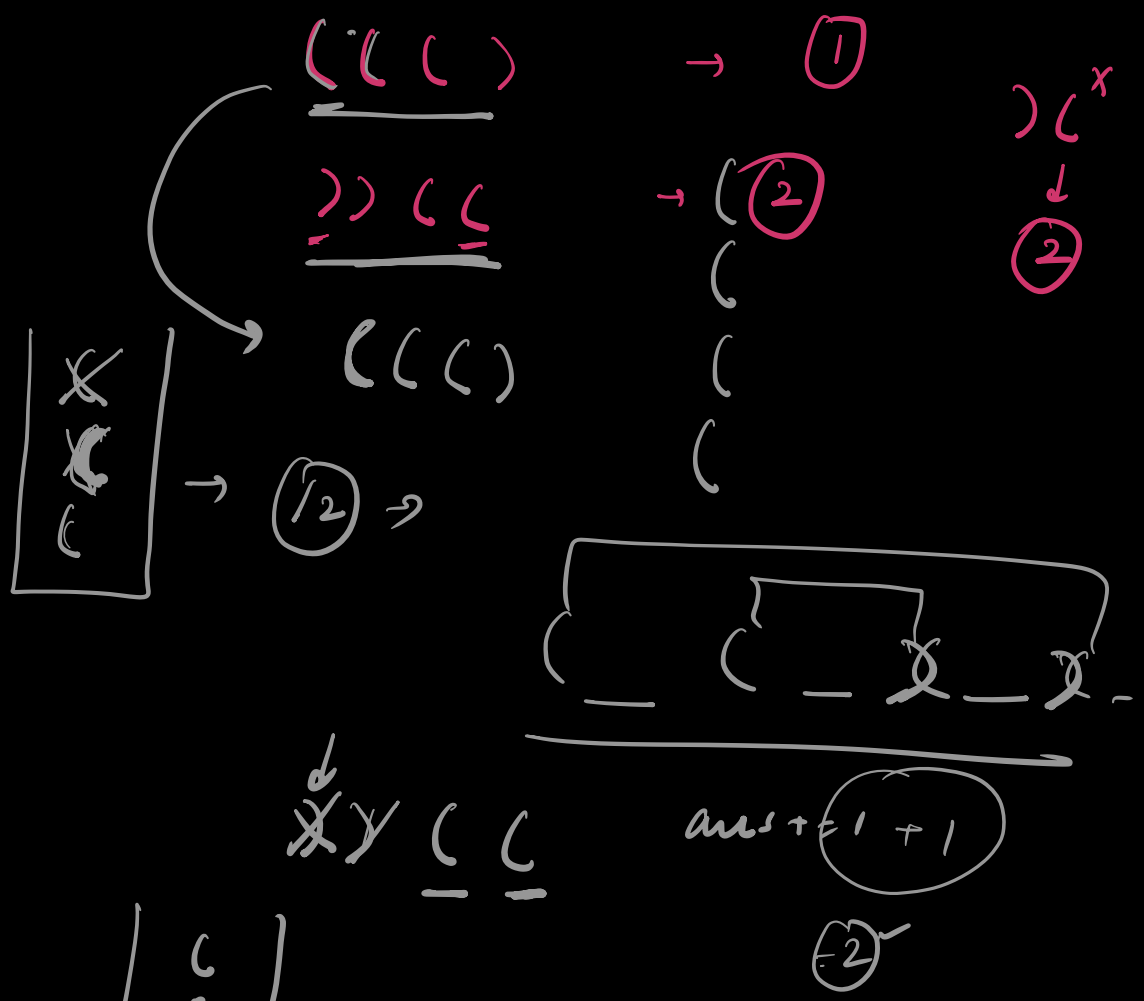


Amazon Q

How many min parantheses would you have to reverse to make a string balance

3 mins

EVEN LENGTH STRING!



Algorithm:

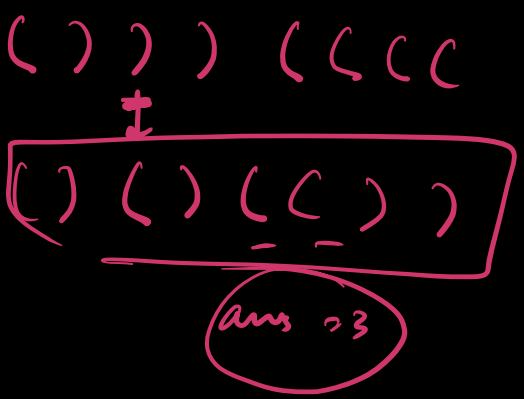
→ Iterate over the string

→ If we see a closing bracket and stack is empty →

Reverse that and add it to stack. Increment the ans

→ In the end, when x open parantheses. ↳ ans → ans + x/2

)) ((



ans = 3

) (( )) ( ) (

↓

((( )) ( ))

ans = 2

5 mins

Queue

i → (0 → n-1)

if a[i] == '('

st.push('(')

else

① st.empty

↳ st.push(')')

② →)

cnt++ = 1

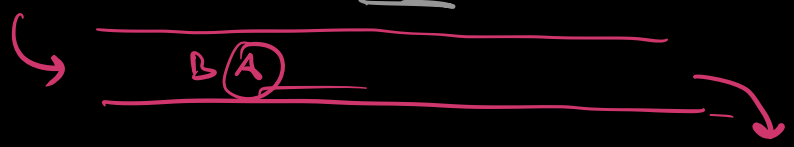
↓

st.pop

Balance

Queues

and ((( )) → Opening Brac



FIFO

↳ First In and First Out

push() → Add an element in the last

pop() → Remove an element from the first

Applications:

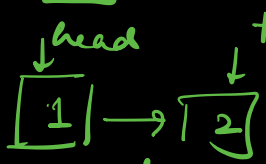
- → Ticketing Queue (WL)
- → Scheduling Algorithm
- → Messaging Queues.

↳ FIFO!

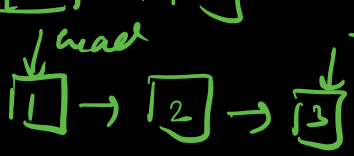
head, tail

push(1)

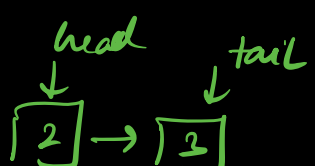
push(2)



push(3)



pop()



push()

tail.next = new_node

tail = new_node

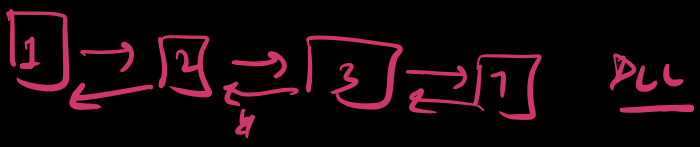
pop()

head = head.next

return head

Doubly Linked List

prev
next



DLL

```
import collection
collections.deque ( [1, 2, 3] )
```

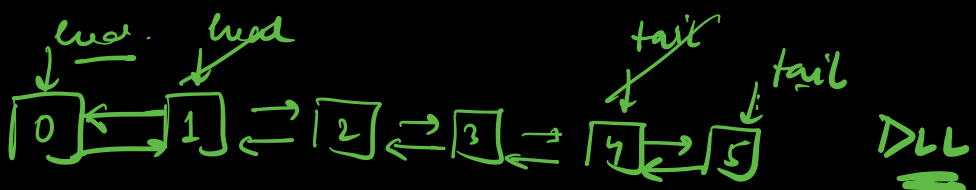
1, 2, 3

↳ append
append left
pop
popleft

O(1) ←

push() → append Left

pop() → pop()



- pop
- popleft
- append
- appendleft

→ O(1)

Q. You have only 3 digits → 1, 2, 3

Find the K^{th} number in ascending order that is formed using these digits. → 1, 2

K = 5 →

1, 2, 3, 11, 12, 13, 21, 22, 23

31 32 33

111 112

1 digit → 1 2 3

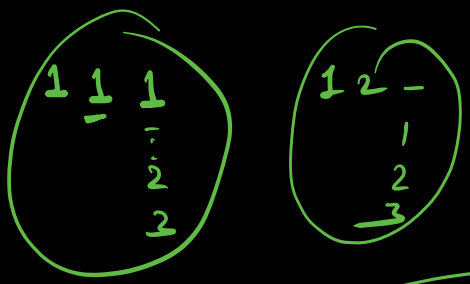
2 dig

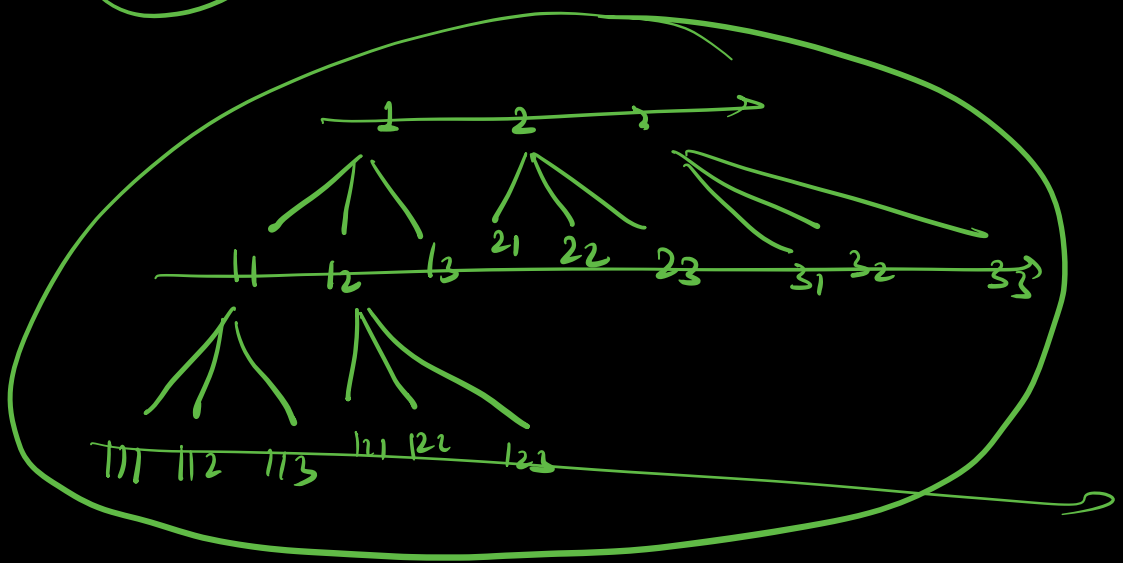
1 _ → 1 2 3

2 _ → 1 2 3

3 _ → 1 2 3

3 digits





1 2 3 11 12 13 21 22 23.

k = 5

~~1, 2, 3~~ 11 12 13 21 22 23

31 32 33 111 112 113

BFS