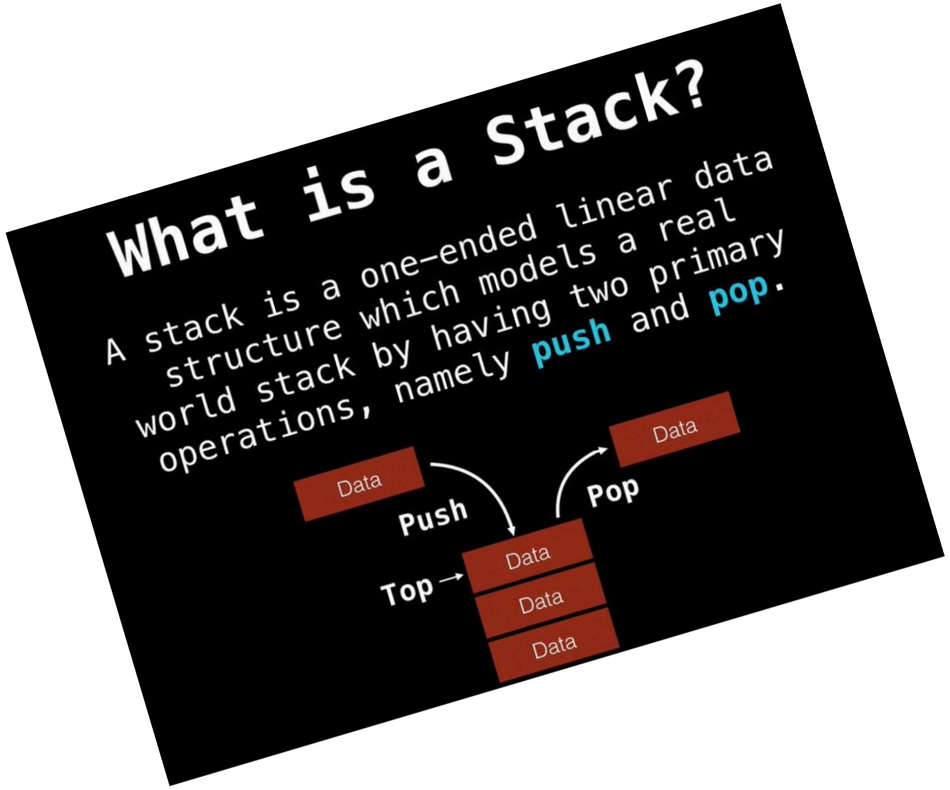
TODO BEFORE

Useful Resource #BeReady

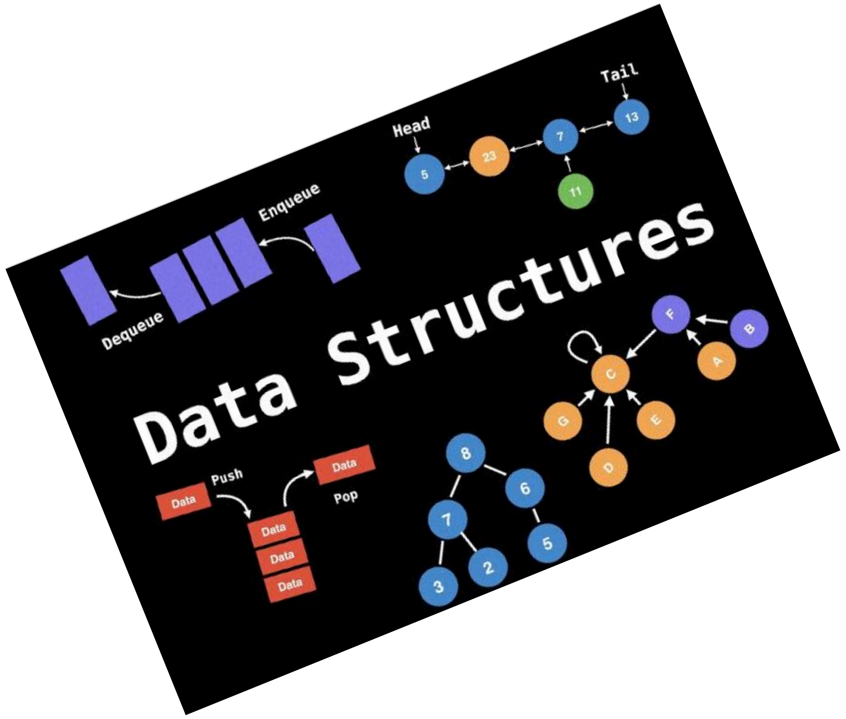
BEFORE THIS COURSE !

✓ What is a stack?


What is a Stack?
A stack is a one-ended linear data structure which models a real world stack by having two primary operations, namely push and pop.
Data
Push
Pop
Data
Top
Data
Data
Data

ALONG THE COURSE !

✓ Follow this playlist to understand the most important data structures



ADVANCED ALGORITHM

W5-S1 – Stack

CADT IDT

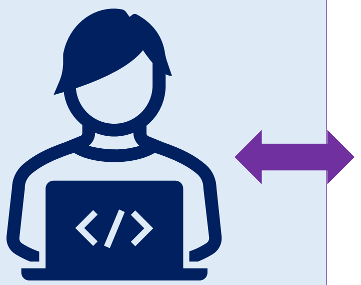
🥇 Objectives for today 🥇

- ✓ **Know** the LIFO (Last-In, First-Out) principle.
- ✓ **Understand** The Stack Structure
- ✓ **Differentiate** the implementation approach (using array vs linkedlist).

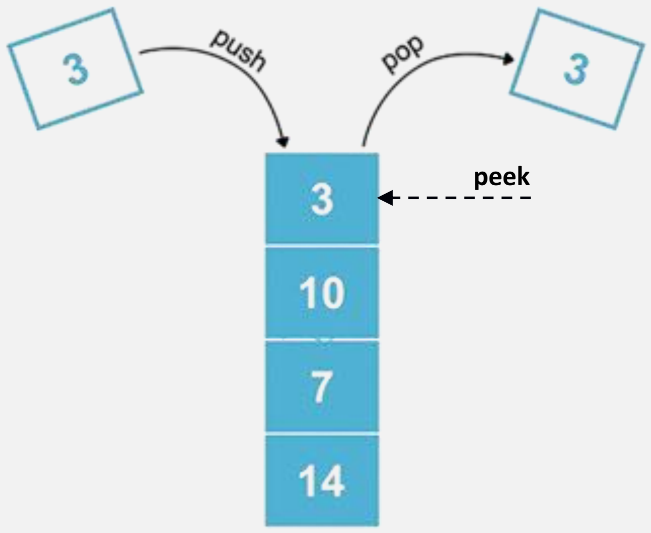
Last In, First Out

A stack uses a **LIFO** ordering: As a stack of dinner plates, the **most recent item** added to the stack is the **first item removed**

Stack ADT



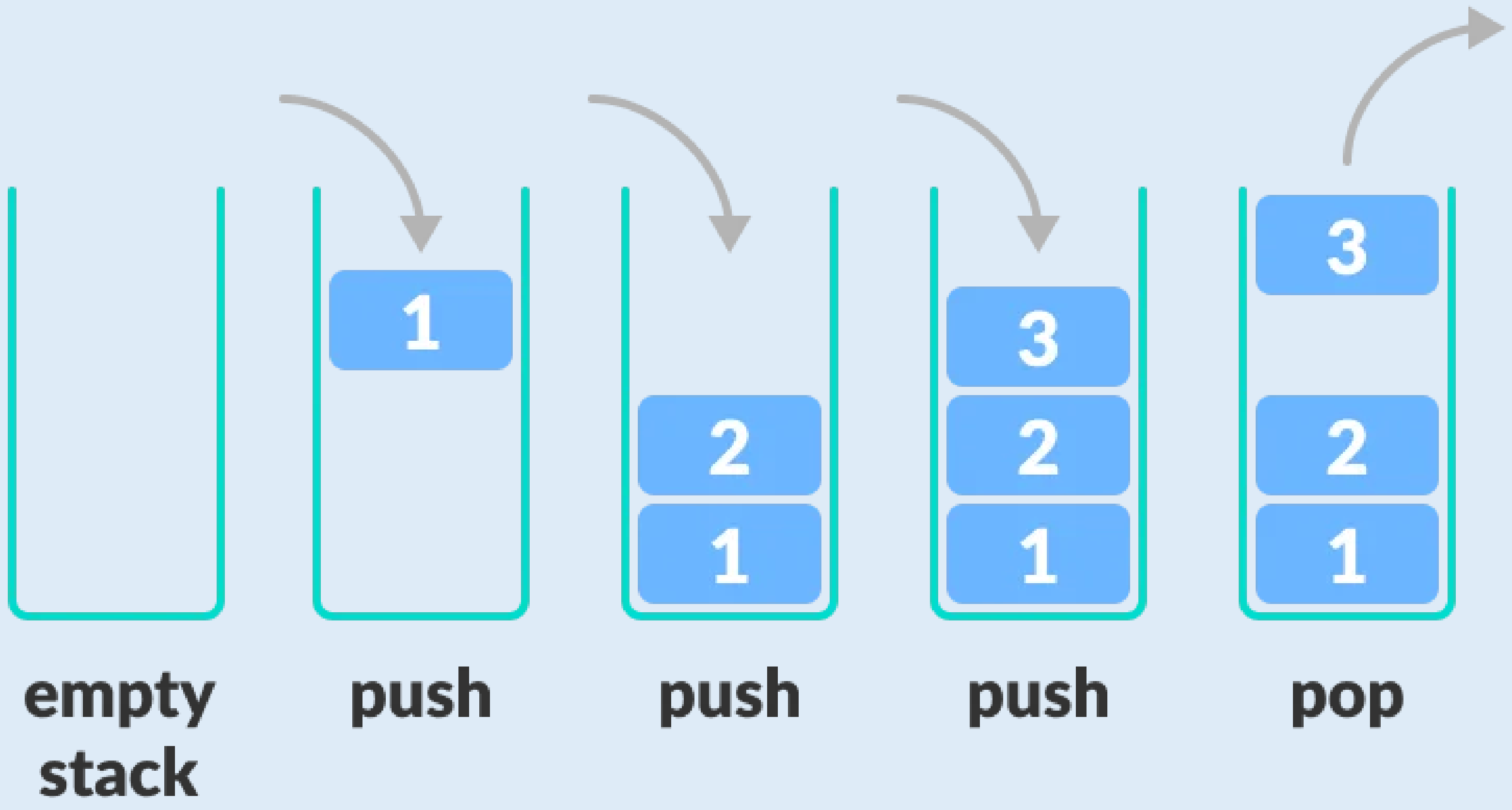
pop()
push()
peek()
isEmpty()



A stack ADT has only 4 basic operations

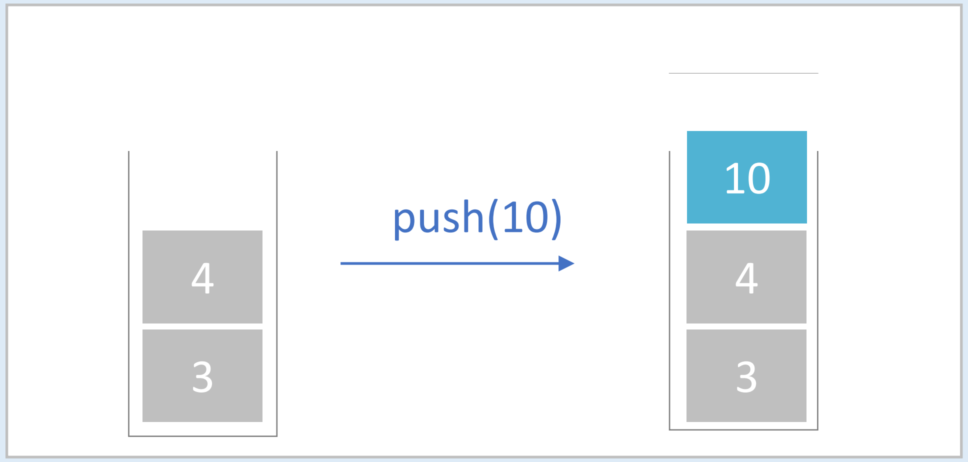
EXPLAIN

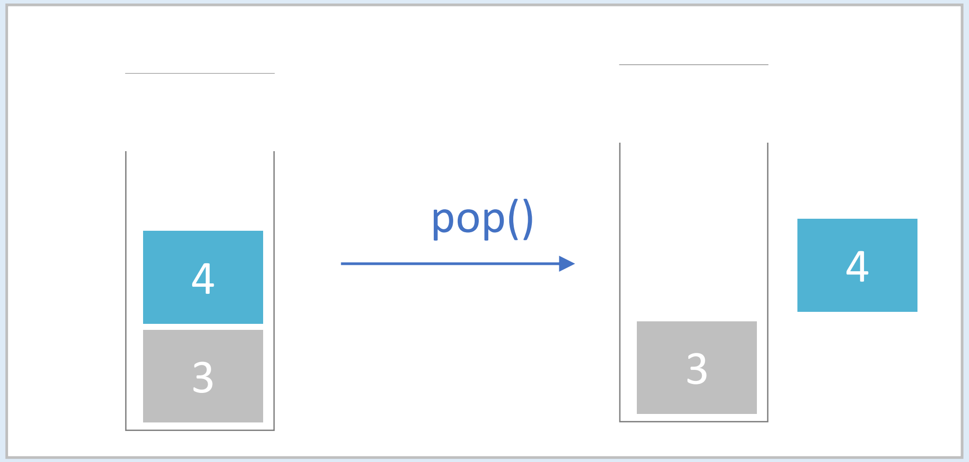
Stack **operations**

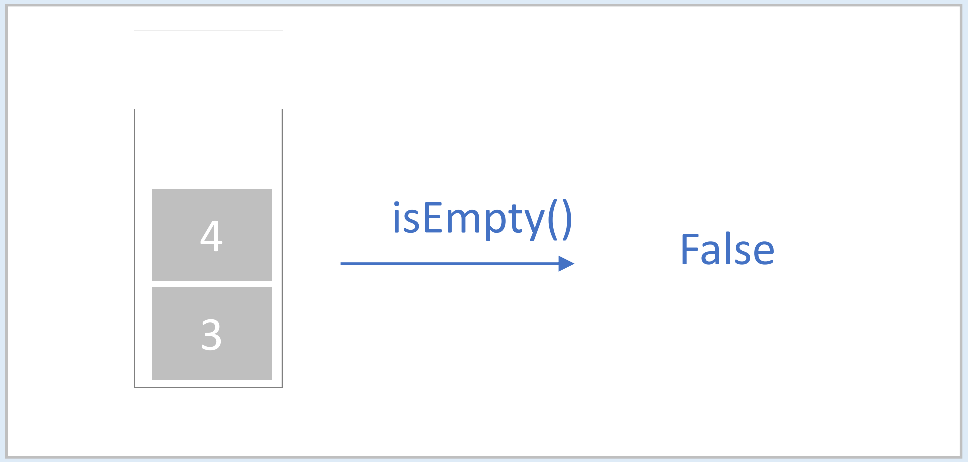


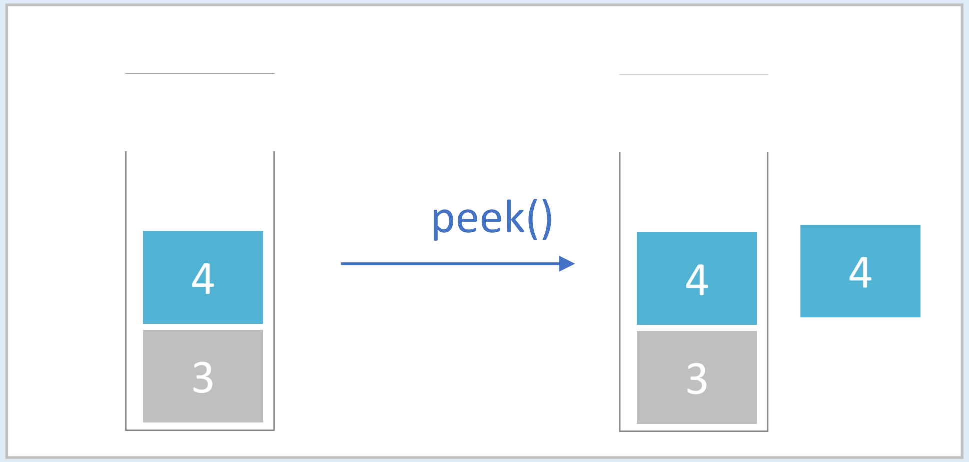
EXPLAIN

Stack **operations**









EXPLAIN

When a stack is **used**?

Just some examples !

- ✓ Back and forward buttons in browser
- ✓ Used by **undo/redo editors**
- ✓ Used in **compiler syntax checking / delimiter checking**
- ✓ **To manage recursion** – by keeping track of the previous function calls
- ✓ To do a **Depth First Search (DSF) on a graph** (to see later)
- ✓ Matching HTML tag in web development
- ✓ Etc.

https://www.enjoyalgorithms.com/blog/application-of-stack-data-structure-in-programming

ACTIVITY

10 MIN

Push

*The operation **push()** insert an element to the top of the Stack.*

Q1 – Let's define the **specifications** of this operation

Syntax	
Description	
Precondition	
Example	
Complexity	

ACTIVITY

10 MIN

Pop

*The operation **pop()** remove an element to the top of the Stack.*

Q1 - Define the **specifications** of this operation

Syntax	
Description	
Precondition	
Example	
Complexity	

ACTIVITY

10 MIN

Peek

The operation ***peek()*** *return a value of the top element of the Stack.*

Q1 – Let's define the **specifications** of this operation

Syntax	
Description	
Precondition	
Example	
Complexity	

ACTIVITY

10 MIN

isEmpty

*The operation **isEmpty()** return true when stack is empty. False, otherwise.*

Q1 - Define the **specifications** of this operation

Syntax	
Description	
Precondition	
Example	
Complexity	

ACTIVITY

The **Bracket validation** problem

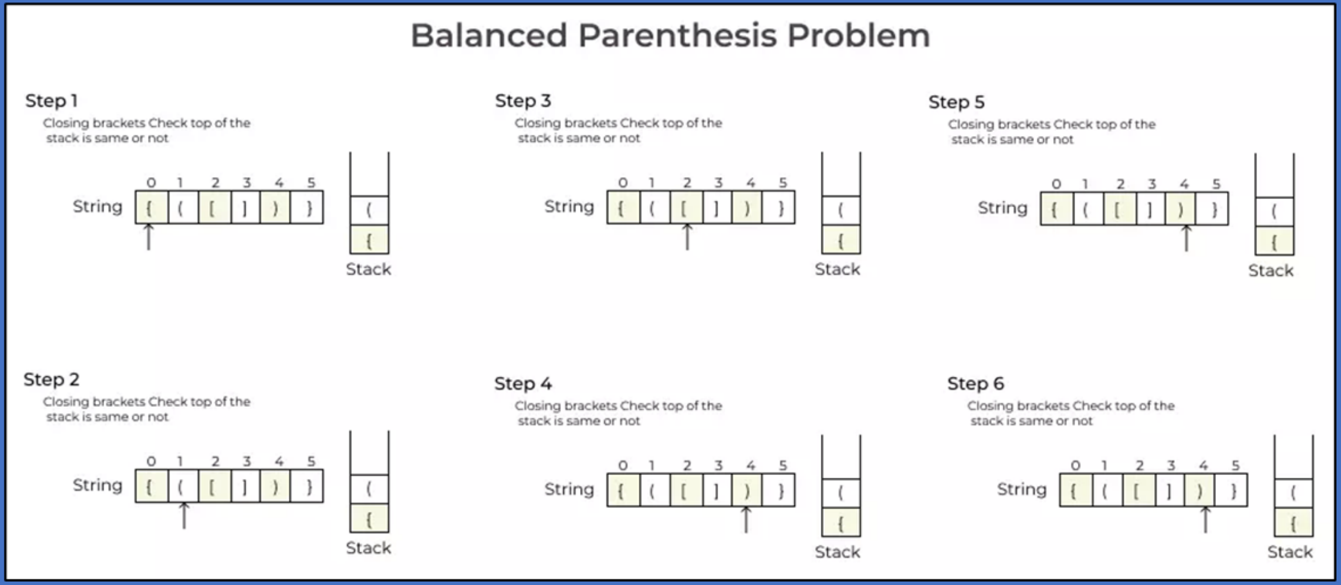
Can you come up with a solution to this problem using a Stack?

[{}]	→	Valid
(()())	→	Valid
{]	→	Invalid
[()]))()	→	Invalid
[]{}({})	→	Valid

Process each character of the input:

- When a character is left delimiter, push it to stack.
- When a character is right delimiter, pop data from stack and check whether popped element matches right delimiter.

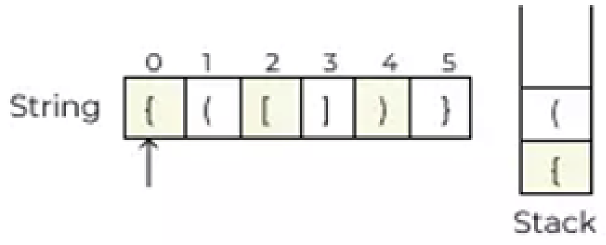
<u>REMARK:</u> It is balance when all matching are true and stack is empty when all characters have been processed.


Balanced Parenthesis Problem
Step 1
Closing brackets Check top of the stack is same or not
String
0 1 2 3 4 5
{ ([ ]) }
Stack
Step 3
Closing brackets Check top of the stack is same or not
Step 5
Closing brackets Check top of the stack is same or not
Step 2
Closing brackets Check top of the stack is same or not
Step 4
Closing brackets Check top of the stack is same or not
Step 6
Closing brackets Check top of the stack is same or not

Balanced Parenthesis Problem

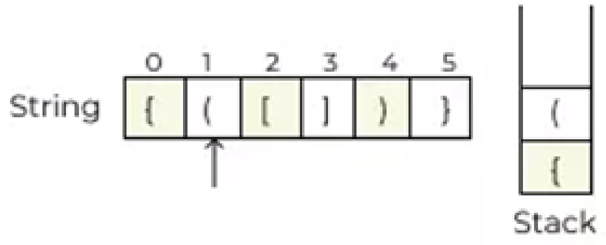
Step 1

Closing brackets Check top of the stack is same or not



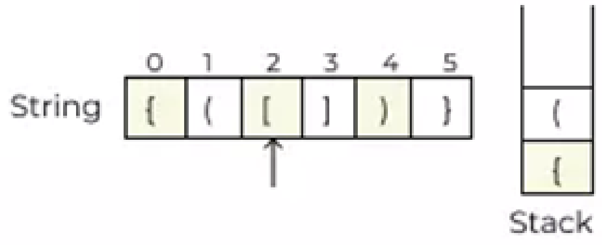
Step 2

Closing brackets Check top of the stack is same or not



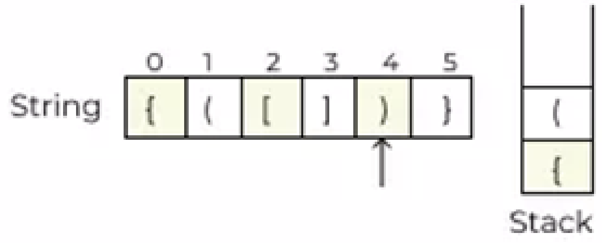
Step 3

Closing brackets Check top of the stack is same or not



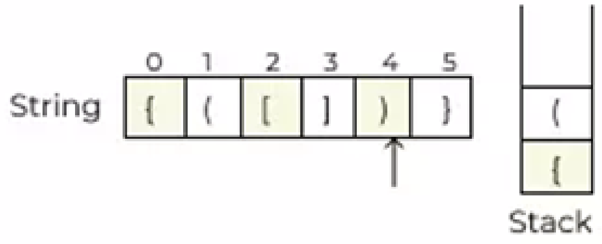
Step 4

Closing brackets Check top of the stack is same or not



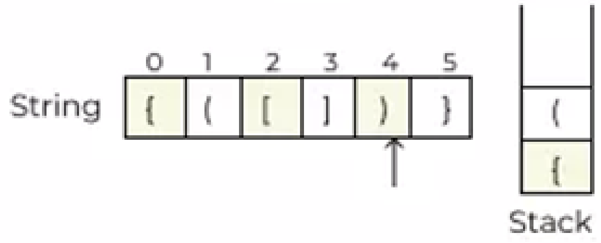
Step 5

Closing brackets Check top of the stack is same or not

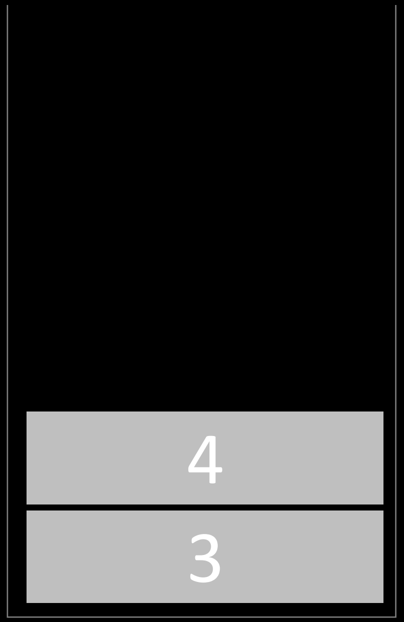


Step 6

Closing brackets Check top of the stack is same or not

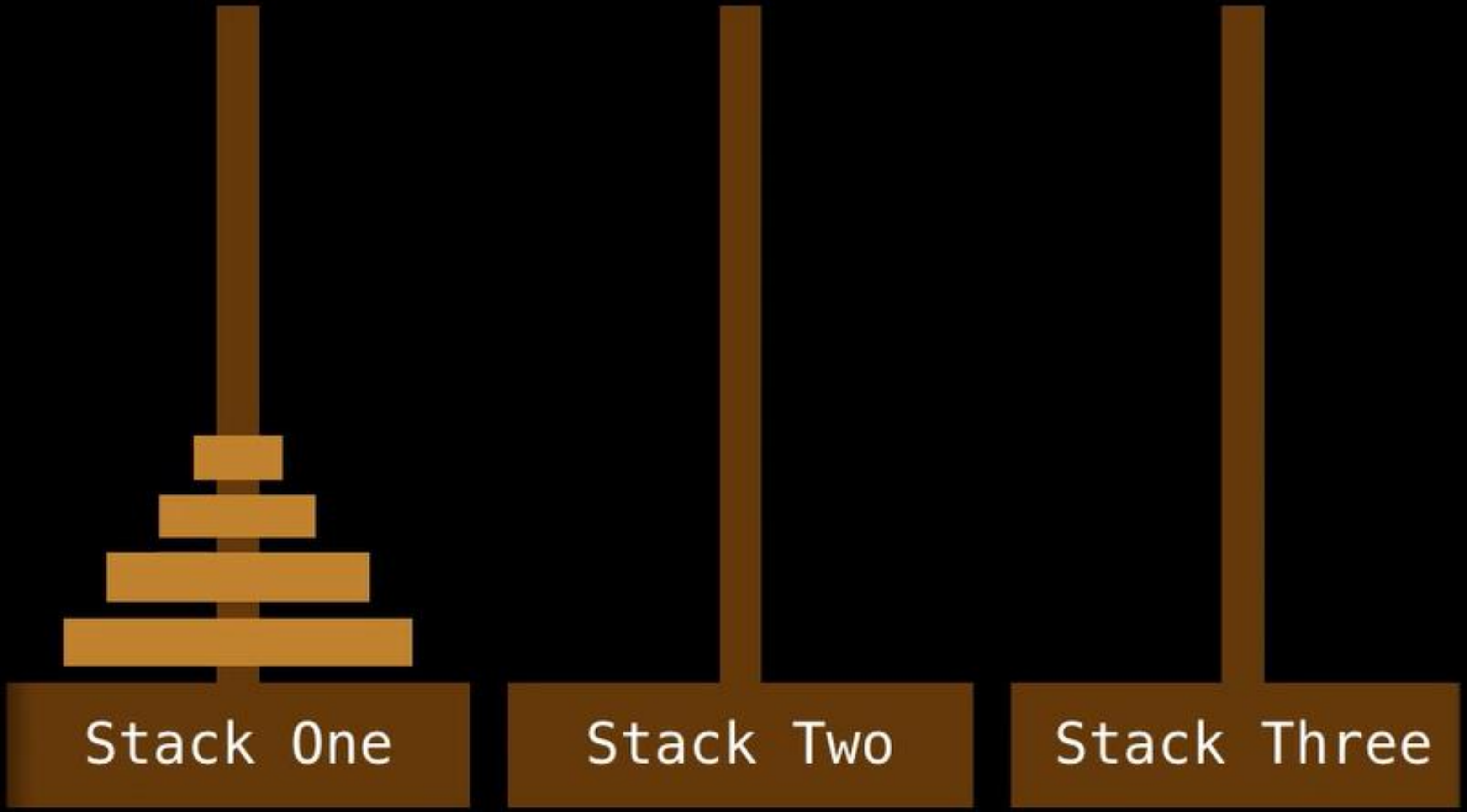


```
Let S be a stack
For bracket in bracket_string:

    rev = getReversedBracket(bracket)

    If isLeftBracket(bracket):
        S.push(bracket)

    Else If S.isEmpty() or S.pop() != rev:
        return false // Invalid

return S.isEmpty() // Valid if S is empty
```



The **Tower Of Hanoi**

Use push() and pop() operation to move the rings to the last tower



A larger item can't go on top of a smaller item

https://www.mathsisfun.com/games/towerofhanoi.html

EXPLAIN

Stack **Implementation**

A stack ADT can either be implemented using:

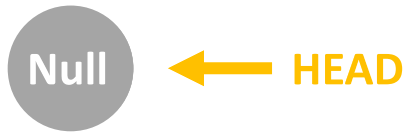
- A dynamic array
- **A single linked list**
- Or even a doubled linked list

Let s start with
This one!

EXPLAIN

Pushing

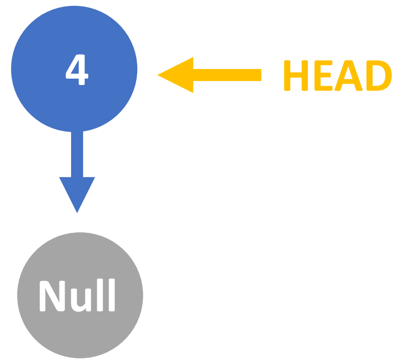
```
push(4)
push(2)
push(6)
push(13)
```



EXPLAIN

Pushing

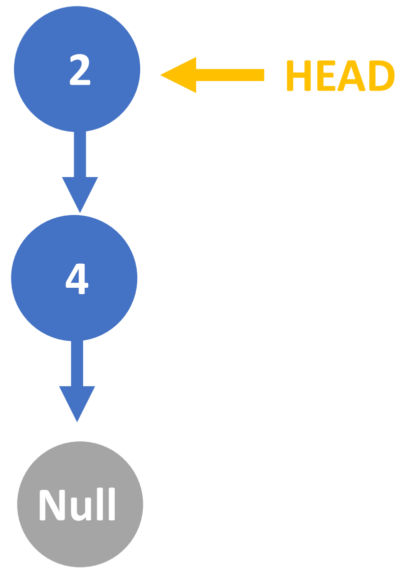
```
push(4)
push(2)
push(6)
push(13)
```



EXPLAIN

Pushing

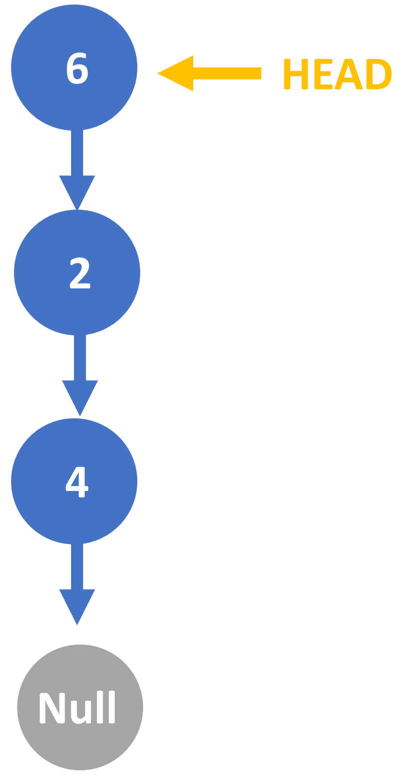
```
push(4)
push(2)
push(6)
push(13)
```



EXPLAIN

Pushing

```
push(4)
push(2)
push(6)
push(13)
```



EXPLAIN

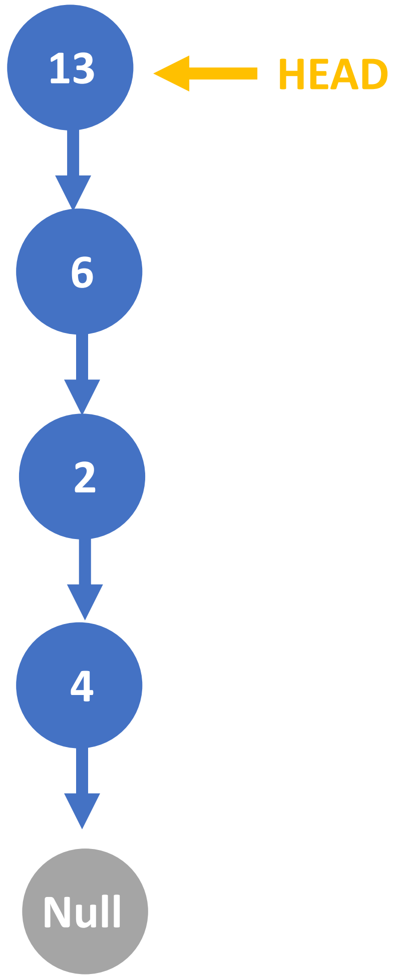
Pushing

push(4)

push(2)

push(6)

➡ push(13)



EXPLAIN

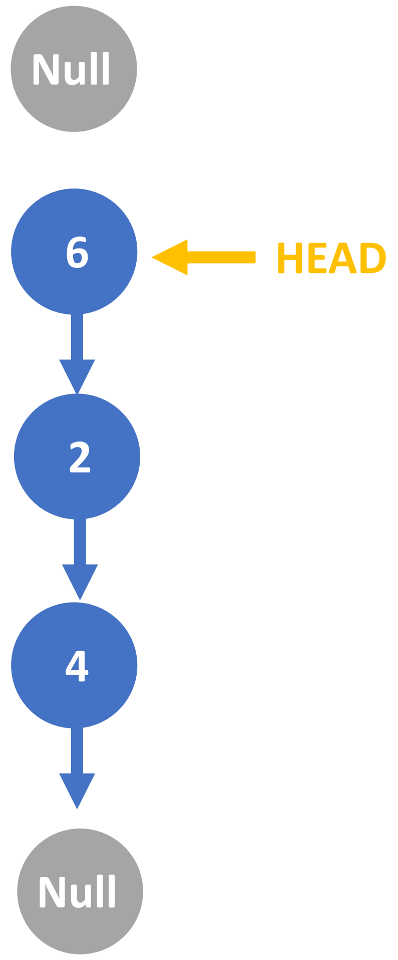
Popping

➡ pop()

pop()

pop()

pop()



EXPLAIN

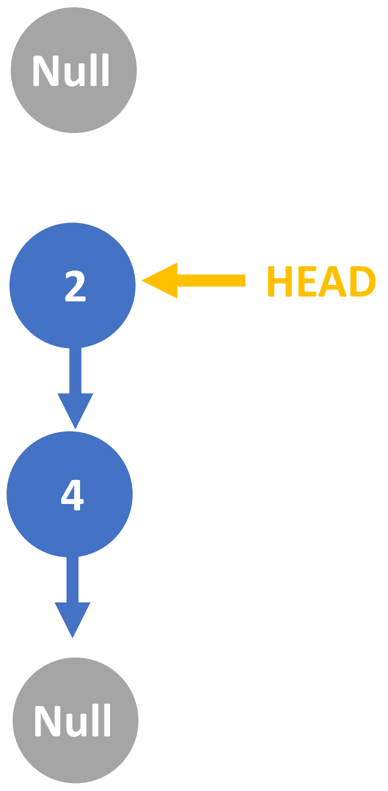
Popping

pop( )

➡ pop( )

pop( )

pop( )

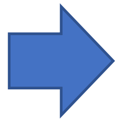


EXPLAIN

Popping

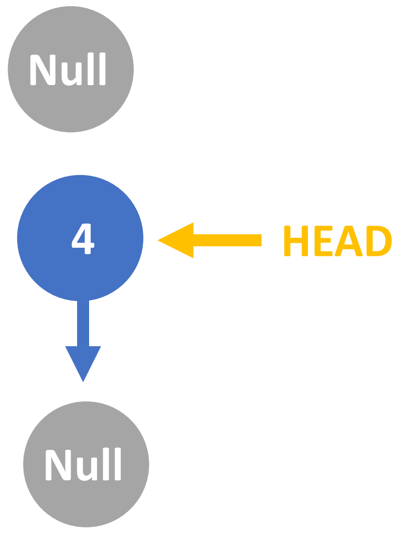
pop( )

pop( )



pop( )

pop( )



EXPLAIN

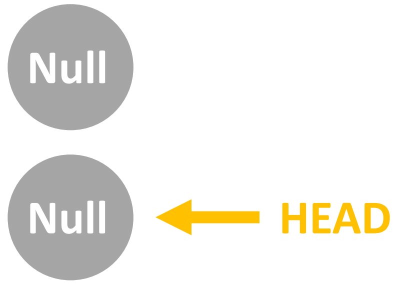
Popping

pop( )

pop( )

pop( )

pop( )



ACTIVITY

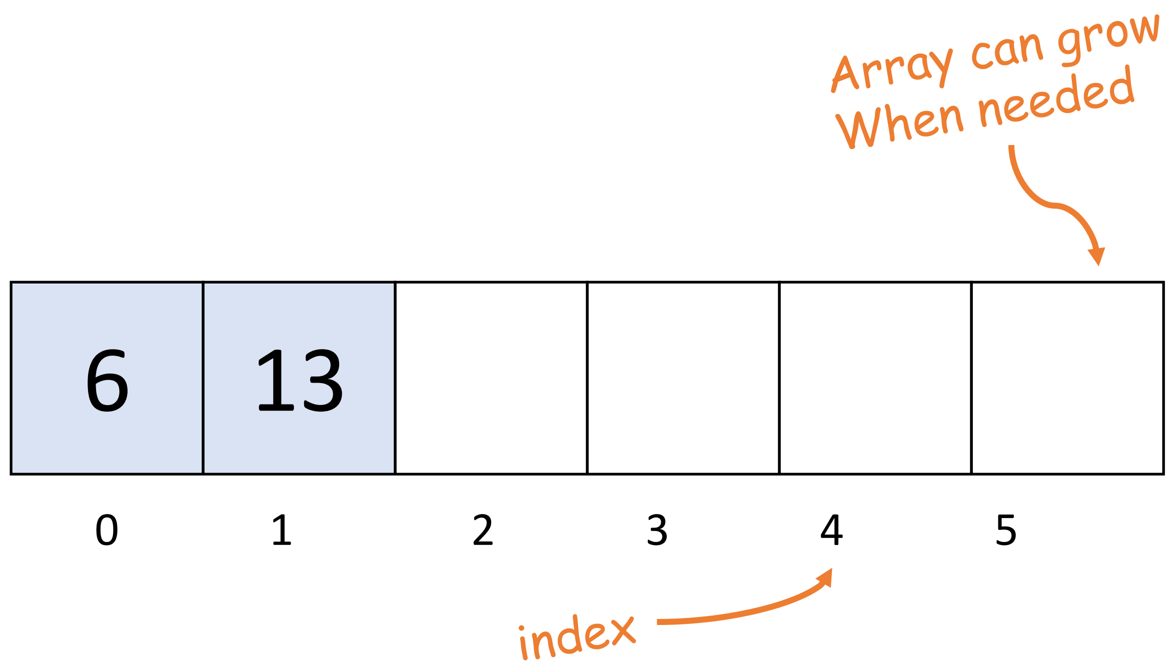
10 MIN

Stack **with an array**

Propose an implementation of the Stack ADT using a dynamic array

✓ Draw a use case with the stack operation

✓ Identify the specific cases
   - *Ex: the array need to grow…*



3-2-1 Challenge

- ✓ List three things you **learned** today.
- ✓ List two **questions** you still have.
- ✓ List one aspect of the lesson or topic you **enjoyed**.