TIPS template

# Interviewer:

## Behavioral:

Ask the interviewee a star based question (https://www.themuse.com/advice/star-interview-method)  
Describe a long-term project that you managed. How did you keep everything moving along in a timely manner?

## Question:

<https://leetcode.com/problems/valid-parentheses/>

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

## Examples:

Example 1:

Input: s = "()"

Output: true

Example 2:

Input: s = "()[]{}"

Output: true

Example 3:

Input: s = "(]"

Output: false

Example 4:

Input: s = "([)]"

Output: false

Example 5:

Input: s = "{[]}

## Follow up Q&A:

* What if you get an empty string?
  + Return True still a valid expression
* What if there are no non parenthesis characters?
  + Assume you are not getting that in the input
* Are there any time or space complexity restrictions?
  + No

## Hint(s):

Hint 1:

An interesting property about a valid parenthesis expression is that a sub-expression of a valid expression should also be a valid expression. (Not every sub-expression) e.g.

{ { } [ ] [ [ [ ] ] ] } is VALID expression

[ [ [ ] ] ] is VALID sub-expression

{ } [ ] is VALID sub-expression

Can we exploit this recursive structure somehow?

Hint 2:

What if whenever we encounter a matching pair of parenthesis in the expression, we simply remove it from the expression? This would keep on shortening the expression. e.g.

{ { ( { } ) } }

|\_|

{ { ( ) } }

|\_\_\_\_\_\_|

{ { } }

|\_\_\_\_\_\_\_\_\_\_|

{ }

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

VALID EXPRESSION!

Hint 3:

The stack data structure can come in handy here in representing this recursive structure of the problem. We can't really process this from the inside out because we don't have an idea about the overall structure. But, the stack can help us process this recursively i.e. from outside to inwards.

## Solution(s): (General concept and time/space complexity)

### Stack Solution

Description:

1. Initialize a stack S.
2. Process each bracket of the expression one at a time.
3. If we encounter an opening bracket, we simply push it onto the stack. This means we will process it later, let us simply move onto the sub-expression ahead.
4. If we encounter a closing bracket, then we check the element on top of the stack. If the element at the top of the stack is an opening bracket of the same type, then we pop it off the stack and continue processing. Else, this implies an invalid expression.
5. In the end, if we are left with a stack still having elements, then this implies an invalid expression.

Time complexity: O(n)

Space complexity: O(n)

Python Solution

class Solution(object):

def isValid(self, s):

"""

:type s: str

:rtype: bool

"""

# The stack to keep track of opening brackets.

stack = []

# Hash map for keeping track of mappings. This keeps the code very clean.

# Also makes adding more types of parenthesis easier

mapping = {")": "(", "}": "{", "]": "["}

# For every bracket in the expression.

for char in s:

# If the character is an closing bracket

if char in mapping:

# Pop the topmost element from the stack, if it is non empty

# Otherwise assign a dummy value of '#' to the top\_element variable

top\_element = stack.pop() if stack else '#'

# The mapping for the opening bracket in our hash and the top

# element of the stack don't match, return False

if mapping[char] != top\_element:

return False

else:

# We have an opening bracket, simply push it onto the stack.

stack.append(char)

# In the end, if the stack is empty, then we have a valid expression.

# The stack won't be empty for cases like ((()

return not stack

Python solution

### Other questions follow up

*Ask if there is more than 5 minutes remaining when they finish their code and testing.*

* (If iterative) How would you do this recursively
* (if recursive) How would you do this iteratively
* How would you handle non bracket/parenthesis characters?

# Interviewee:

## Question:

## Example(s):

## Code below