20. Valid Parentheses

Easy | Topics | Companies | Hint

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.
3. Every close bracket has a corresponding open bracket of the same type.

Example 1:

> **Input:** `s = "()"`
>
> **Output:** `true`

Example 2:

> **Input:** `s = "()[]{}"`
>
> **Output:** `true`

Example 3:

> **Input:** s = "(]"
>
> **Output:** false

Example 4:

> **Input:** s = "([])"
>
> **Output:** true

Example 5:

> **Input:** s = "([)]"
>
> **Output:** false

Constraints:

- `1 <= s.length <= 10^4`
- `s` consists of parentheses only `'()[]{}'`.

Java

```
class Solution {
    public boolean isValid(String s) {
        Stack<Character> stack = new Stack<>();

        for (char c : s.toCharArray()) {
            // If it's an opening bracket, push it to stack
            if (c == '(' || c == '{' || c == '[') {
                stack.push(c);
            }
            // If it's a closing bracket, check with top of stack
            else {
```

```
                if (stack.isEmpty()) return false; // No opening bracket
                char top = stack.pop();
                if (c == ')' && top != '(') return false;
                if (c == '}' && top != '{') return false;
                if (c == ']' && top != '[') return false;
            }
        }
        // If stack is empty, everything matched correctly
        return stack.isEmpty();
    }
}
```

Javascript

```
/**
 * @param {string} s
 * @return {boolean}
 */
var isValid = function(s) {
    let stack = [];
    let map = {
        ')': '(',
        '}': '{',
        ']': '['
    };

    for (let char of s) {
        // If it's a closing bracket
        if (char in map) {
            // Pop the top element from stack, if empty use dummy value
            let top = stack.length > 0 ? stack.pop() : '#';

            // Check if it matches the corresponding opening bracket
            if (map[char] !== top) {
                return false;
            }
        } else {
            // If it's an opening bracket, push to stack
            stack.push(char);
        }
    }

    // If stack is empty, all brackets matched
    return stack.length === 0;
};
```

Python

```
class Solution:
    def isValid(self, s: str) -> bool:
        bracket_map = {')': '(', '}': '{', ']': '['}

        stack = []

        for char in s:
            if char in bracket_map.values():  # If it's an opening bracket
                stack.append(char)
            elif char in bracket_map:  # If it's a closing bracket
                if not stack or stack[-1] != bracket_map[char]:
                    return False
                stack.pop()
            else:
                return False  # Invalid character (not needed as per constraints)

        return not stack  # Stack should be empty if valid
```