'

Problem 20: Valid Parentheses

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Given a string

s

containing just the characters

'('

,

')'

,

'{'

,

'}'

,

'['

and

']'

, determine if the input string is valid.

An input string is valid if:

Open brackets must be closed by the same type of brackets.

Open brackets must be closed in the correct order.

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

'()[]{}'

.

Code Snippets

C++:

```
class Solution {
public:
bool isValid(string s) {

}
};
```

Java:

```
class Solution {
public boolean isValid(String s) {

}
}
```

Python3:

```
class Solution:
def isValid(self, s: str) -> bool:
```

Python:

```
class Solution(object):
def isValid(self, s):
"""
:type s: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} s
* @return {boolean}
*/
var isValid = function(s) {

};
```

TypeScript:

```
function isValid(s: string): boolean {

};
```

C#:

```
public class Solution {
public bool IsValid(string s) {

}
}
```

C:

```
bool isValid(char* s) {

}
```

Go:

```
func isValid(s string) bool {

}
```

Kotlin:

```
class Solution {
fun isValid(s: String): Boolean {

}
}
```

Swift:

```
class Solution {
func isValid(_ s: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn is_valid(s: String) -> bool {

}
}
```

Ruby:

```
# @param {String} s
# @return {Boolean}
def is_valid(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return Boolean
*/
function isValid($s) {

}
}
```

Dart:

```
class Solution {
bool isValid(String s) {

}
}
```

Scala:

```
object Solution {
def isValid(s: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec is_valid(s :: String.t) :: boolean
def is_valid(s) do

end
end
```

Erlang:

```
-spec is_valid(S :: unicode:unicode_binary()) -> boolean().
is_valid(S) ->
.
```

Racket:

```
(define/contract (is-valid s)
(-> string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Valid Parentheses
* Difficulty: Easy
* Tags: string, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
bool isValid(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Valid Parentheses
* Difficulty: Easy
* Tags: string, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public boolean isValid(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Valid Parentheses
Difficulty: Easy
Tags: string, stack

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def isValid(self, s: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def isValid(self, s):
"""
:type s: str
:rtype: bool
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Valid Parentheses
* Difficulty: Easy
* Tags: string, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string} s
* @return {boolean}
*/
var isValid = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Valid Parentheses
* Difficulty: Easy
* Tags: string, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function isValid(s: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Valid Parentheses
* Difficulty: Easy
* Tags: string, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public bool IsValid(string s) {

}
}
```

C Solution:

```
/*
* Problem: Valid Parentheses
* Difficulty: Easy
* Tags: string, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

bool isValid(char* s) {

}
```

Go Solution:

```
// Problem: Valid Parentheses
// Difficulty: Easy
// Tags: string, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach
```

```
func isValid(s string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun isValid(s: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func isValid(_ s: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Valid Parentheses
// Difficulty: Easy
// Tags: string, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn is_valid(s: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {Boolean}
def is_valid(s)
```

```



end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return Boolean
*/
function isValid($s) {

}
}
```

Dart Solution:

```
class Solution {
bool isValid(String s) {

}
}
```

Scala Solution:

```
object Solution {
def isValid(s: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec is_valid(s :: String.t) :: boolean
def is_valid(s) do

end
end
```

Erlang Solution:

```
-spec is_valid(S :: unicode:unicode_binary()) -> boolean().
is_valid(S) ->
.
```

Racket Solution:

```
(define/contract (is-valid s)
(-> string? boolean?)
)
```