JavaScript Valid Parentheses

Challenge

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.

1st Example

```
Input: s = '()'
Output: true
```

2nd Example

```
Input: s = '()[]{}'
Output: true
```

3rd Example

```
Input: s = '(]'
Output: false
```

Constraints

- `1 <= s.length <= 10⁴`
- `s` consists of parentheses only `'()[]{}'`.

Solution

```
const isValid = (s) => {
    const stack = [],
          map   = {
            '(': ')',
            '[': ']',
            '{': '}'
        };

        for (let i = 0; i < s.length; i++) {
        let c = s[i];

        if (map[c]) {
            stack.push(map[c]);
        } else if (c !== stack.pop()) {
            return false;
        }
    }

    return !stack.length;
};
```

Explanation

I've created a function called `isValid` that checks whether a string contains valid pairs of brackets.

The function initializes an empty array called `stack` to serve as a stack data structure. It also defines a map object called `map` that maps opening brackets to their corresponding closing brackets.

A `for` loop is used to iterate through each character in the input string. Inside the loop, the current character is assigned to the variable `c`.

If `c` exists as a key in the `map` object, it means it is an opening bracket. In this case, the corresponding closing bracket is pushed onto the stack.

If `c` is not an opening bracket, it is checked whether it matches the topmost element in the stack (the last opening bracket encountered). If they don't match, it means the brackets are not balanced, and the function returns `false`.

After the loop finishes, if there are any remaining elements in the stack, it means there were opening brackets without corresponding closing brackets. In this case, the function returns `false`.

If the stack is empty, it means all opening brackets had corresponding closing brackets, and the function returns `true`.

In summary, this function checks whether a given string contains valid pairs of brackets by using a stack data structure. It iterates through the string, pushing opening brackets onto the stack and checking if closing brackets match the topmost element in the

stack. If the brackets are balanced, the function returns `true`; otherwise, it returns `false`.

Author: Trevor Morin