# Valid Parentheses

## Problem Definition

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 Cases

**Example 1:**

Input: s = "()"
Output: true

**Example 2:**

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

**Example 3:**

Input: s = "(]"
Output: false

**Example 4:**

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

## Constraints

Constraints:

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

## Example test cases

In [2]:
```python
def test_cases():
    assert isValid("()") is True
    assert isValid("()[]{}") is True
    assert isValid("(]") is False
    assert isValid("([])") is True
    print("All test cases passed!")
```

## Solutions

$O(n)$

The first solution that seems reasonable is to keep a mapping with opening brackets as the keys and closing ones as the values.

- If the length of the string is odd, then it is False
- If the first character is a closing bracket, again False

Keep a list of the elements (as a stack)

For each letter, check if it is an opening bracket or closing.

- If the letter is opening, add it to the stack,
- If the letter is closing, see if the last element in the stack is an opening bracket or not. If not, return False, if yes, remove the last element from the stack.

At the end, if the sequence length is 0, return True, else False

In [56]:
```python
def isValid(s):
    mapping = {
        "{": "}",
        "[": "]",
        "(": ")"
    }

    if (len(s) % 2 != 0) or (s[0] in mapping.values()):
        return False

    stack = []
    for letter in s:
        if letter in mapping.keys():
            stack.append(letter)
        else:
            if mapping[stack[-1]] == letter:
                stack = stack[:-1]
            else:
                False
```

```
    return not stack
```

In [57]:
```
test_cases()
```

```
All test cases passed!
```

A small adjustment can be made to remove additional else block:

In [64]:
```
def isValid(s):
    mapping = {
        "{": "}",
        "[": "]",
        "(": ")"
    }

    if (len(s) % 2 != 0) or (s[0] in mapping.values()):
        return False

    stack = []
    for letter in s:
        if letter in mapping.keys():
            stack.append(letter)
        else:
            last_element = stack.pop() if stack else ""
            if (last_element == "") or (mapping[last_element] != letter):
                return False

    return not stack
```

In [65]:
```
test_cases()
```

```
All test cases passed!
```

Inverting the mapping would provide a clearer code ( `last_element == ""` is removed):

In [68]:
```
def isValid(s):
    mapping = {
        "}": "{",
        "]": "[",
        ")": "("
    }

    if (len(s) % 2 != 0) or (s[0] in mapping.keys()):
        return False

    stack = []
    for letter in s:
        if letter in mapping.values():
            stack.append(letter)
        else:
            last_element = stack.pop() if stack else ""
            if (mapping[letter] != last_element):
                return False
```

```
    return not stack
```

In [69]:
```
test_cases()
```

```
All test cases passed!
```