CSC148 - Balancing Parentheses

We are writing client code and need a function (outside the class) to determine whether the parentheses in an expression are balanced: opening and closing parentheses match and are properly nested inside each other.

1. For four examples, we'll give you a string one character at a time. Your job is to determine whether the string has balanced parentheses or not. *Don't just write down every character without thinking!* Instead, use a stack to keep track of the minimum amount of information you need to solve the problem.

Expression 1:

Stack

Were the parentheses balanced?

Yes No

Expression 2:

Stack

Were the parentheses balanced?

Yes No

Expression 3:

Stack

Were the parentheses balanced?

Yes No

Expression 4:

Stack

Were the parentheses balanced?

Yes No

2. We need a general strategy that will work in all cases. To find it, answer these questions:

   (a) What will you do with each character as you receive it?

   Anytime we see a (we push it to the stack
   Anytime we see a) we try to pop from the stack

   (b) At the end, how will you know whether the parentheses were balanced?

   (1) if the stack is empty at the end of expression, and
   (2) it was never already empty when we tried to pop from it after seeing a closing bracket

3. Now implement the function.

```python
def is_balanced(line: str) -> bool:
    """Return whether <line> contains balanced parentheses.

    Ignore square and curly brackets.

    >>> is_balanced('(a * (3 + b))')
    True
    >>> is_balanced('(a * (3 + b]]')  # Note that the two ']'s don't matter.
    False
    >>> is_balanced('1 + 2(x - y)}')  # Note that the '}' doesn't matter.
    True
    >>> is_balanced('3 - (x')
    False
    """
```

see balancer_solution.py

4. How would you generalize this code to balance round, square, and curly brackets?

   see balancer_solution.py