String

The assignment this week focuses on string methods and automated testing. You may include an **optional** main() routine to allow the user to interact with your program but unittest automated tests cases are **mandatory**.

Part 1:

Write a function that takes a string as an argument and returns a new string which is the reverse of the argument. E.g.

```
reverse('abc') == 'cba'
```

We saw in the lecture that 'abc'[::-1] reverses the string. You must provide a different solution that ***does not use*** any of Python's built in reverse functions.

Part 2:

Write a generator rev_enumerate(seq) that is similar to Python's built in enumerate(seq) generator but rather than starting at 0 and the first element in the sequence, start at the end of the sequence and return the elements in the sequence from last to first along with the corresponding offset. For example,

```
for i, c in rev_enumerate("Python"):
    print(i, c)

5 n
4 o
3 h
2 t
1 y
0 P
```

Hint: You can return or yield multiple values from a function simply by including all of the values separated by commas. E.g.

```
def foo():
    yield 1, 2   # return two values 1, 2
```

Hint: Your solution to Part 1 is very relevant here.

Part 3:

The string.find(s) method finds the first occurrence of s in string and returns the offset of where s begins in string. Write a function find_second(s1, s2) that returns the offset of the ***second*** occurrence of s1 in s2. Return -1 if s1 does not occur twice in

s2. E.g. find_second('iss','Mississippi') == 4. Careful, the second occurrence might be in the middle of the first occurrence, e.g. find_second('abba', 'abbabba') == 3.

Hint: Python's string.find method accepts an optional second argument that specifies the offset of where to start looking for target in string. E.g. 'abaa'.find('a', begin=1) finds the first occurrence of 'a' in 'abaa' beginning at offset 1 rather than offset 0.

Part 4:

Write a **generator**, get_lines(path), that opens a file for reading and returns one line from the file at a time. However, get_lines() should first combine lines that end with a backslash (a continuation) with the subsequent line or lines until a line is found that does not end with a backslash. Also, get_lines() should remove all comments from the file where comments may begin with a '#' anywhere on the line and continue until the end of the line. Also any line that begins with a comment should be ignored and not returned. E.g.

Given a file:

```
# this entire line is a comment - don't include it in the output
<line0>
# this entire line is a comment - don't include it in the output
<line1># comment
<line2>
# this entire line is a comment - don't include it in the output
<line3.1 \
line3.2 \
line3.3>
<line4.1 \
line4.2>
<line5># comment \
# more comment1 \
more comment2>
<line6>

get_lines() should yield the following lines, one at a time:
<line0>
<line1>
<line2>
<line3.1 line3.2 line3.3>
<line4.1 line4.2>
<line5>
<line6>
```

Hints:

- Combine lines where line i ends with line i + 1 **before** removing comments

Here's my main() routine to demonstrate using the generator

```
def main():
    file_name = '/Users/jrr/Downloads/HW05/test1.txt'

    for line in get_lines(file_name):
        print(line)

Here's my test case for the file above

class GetLinesTest(unittest.TestCase):
    def test_get_lines(self):
        file_name = '/Users/jrr/Downloads/hw05.txt'

        expect = [<line0>', '<line1>', '<line2>', '<line3.1 line3.2 line3.3>
','<line4.1 line4.2>', '<line5>', '<line6>']
        result = list(get_lines(file_name))
        self.assertEqual(result, expect)
```

Part 5:

Be sure your code includes docstrings and follows the PEP-8 coding guidelines, e.g. CamelCase only for class names, appropriate spaces, etc.

Include adequate unittest test cases to demonstrate that your functions work properly.

Please feel free to contact me if you have any questions or run into any problems.