Homework 05

Re-submit Assignment

Due Feb 26 by 10pm **Points** 100 **Submitting** a file upload

Assignment Description

The assignment this week focuses on **string methods, slices, working with files, and automated testing**. Also, for the functions you are required to implement, you should assume that all functions will receive a correct type of input - a string.

Part 1: `reverse(s: str) -> str`

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' # This expression should be True
```

We saw in the lecture that `'abc'[::-1]` reverses the string. You **must** implement your own solution to reverse the string. You **MAY NOT** use any of Python's built in reverse functions.

Part 2: `substring(target: str, s: str) -> int`

Write a substring function that is similar to Python's string.find(target) method that returns the offset from the beginning of string s where target occurs in s. Return -1 if target is not a substring in s. For example,

```
>>> substring("he", "hello")
0
>>> substring("ell", "hello")
1
>>> substring("xxx", "hello")
-1
```

> **NOTE:** You **may not** use any Python string methods including string.find(), string.startswith(), string.index(), etc.
>
> **HINT**: slices might be very helpful

Part 3: `find_second(target: str, string: str) -> int`

The `str.find(substr)` (https://docs.python.org/3/library/stdtypes.html#str.find) method finds the first occurrence of substr in string and returns the offset of where s begins in string. Write a function `find_second(s1: str, s2: str)` -> int 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', 1)` finds the first occurrence of `'a'` in `'abaa'` beginning at offset 1 rather than offset 0.

Part 4: `get_lines(path: str) -> Iterator[str]`

Write a ***generator***, `get_lines(path: str) -> Iterator[str]`, that opens a file for reading and returns one line from the file at a time. `get_lines(path)` must:

1. 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
2. `get_lines(path)` should remove all comments from the file where comments begin with a '#' anywhere on the line and continue until the end of the line
3. Third, any line that begins with a comment should be ignored and not returned.

get_lines() should raise a FileNotFound exception if the file can't be opened for reading.

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>
# here's a comment line continued to the next line \
this line is part of the comment from the previous line
```

`get_lines(path)` 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.
> - Be careful to remove trailing newline (\n) but not to remove leading and trailing blanks and tabs. string.strip() removes whitespace from both the beginning and end of a string. string.rstrip('\n') removes only '\n' from the end of the string.
> - The backslash character (\) has special semantics. If you want to compare a string to the backslash character, then use either ***string* == '\\'** or ***string* == *r*'\'**
> - Files may have blank lines.   Be sure that your solution does not eliminate blank lines.
>
> Here's my main() routine to demonstrate using the generator

```
def main() -> None:
    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: List[str] = ['<line0>', '<line1>', '<line2>', '<line3.1 line3.2 line3.3>','<line4.1 line4.2>',
'<line5>', '<line6>']
        result: List[str] = list(get_lines(file_name))
        self.assertEqual(result, expect)
```

Part 5: Write human readable code

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

Insure that all of your functions include type hints for parameters and return types. Be sure to include type hints for all variables.

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

Deliverable

File Structure

Separate your code into two files - one for code logic and one for unit test. It’s always a good practice to separate the code and the test.

You are going to have **two .py files** for submission:

1. **`HW05_FirstName_LastName.py`**
2. **`HW05_Test_FirstName_LastName.py`**

`HW05_FirstName_LastName.py` should define all of your functions.

`HW05_Test_FirstName_LastName.py` should define all of your automated test cases.

Please let me know if you have any questions.