Strings and File I/O

ISTA 130 Hw6, Due Thursday 4/9/2020 at 11:59 pm

Introduction. This homework is intended to give you practice with string indexing, slicing, and methods, and file I/O and traversing files by line.

Instructions. Create a module named `strings.py`. Below is the spec for six functions. Implement them and upload your module to the Hw6 D2L Assignments folder.

Testing. Download `test_strings.py` and put it in the same folder as your `strings.py` module and the auxiliary testing files. Run it from the command line to see your current correctness score. Each of the six functions is worth 16.7% of your correctness score. You can examine the test module in a text editor to understand better what your code should do. The test module is part of the spec. The test file we will use to grade your program may be different but if your code passes the one provided, it will pass the one we use.

Documentation. Your module must contain a header docstring containing your name, your section leader's name, the date, `ISTA 130 Hw6`, and a brief summary of the module. Each function must contain a docstring. Each function docstring should include a description of the function's purpose, the name, type, and purpose of each parameter, and the type and meaning of the function's return value.

Grading. Your module will be graded on correctness, documentation, and coding style. Code should be clear and concise. You will only lose style points if your code is a real mess. Include inline comments to explain tricky lines and summarize sections of code (not necessary on this assignment).

Collaboration. Collaboration is allowed. You are responsible for your learning. Depending too much on others will hurt you on the tests. "Helping" others too much harms them in reality. Cite any sources/collaborators in your header docstring. Leaving this out is dishonest.

Resources.
https://docs.python.org/3.7/tutorial/index.html
String methods: https://docs.python.org/3.7/library/stdtypes.html#string-methods
The standard library: https://docs.python.org/3.7/library/index.html
Built-in functions: https://docs.python.org/3.7/library/functions.html

If you don't have this document with you and you want to find, say, all string methods, you can start the interpreter and type `dir(str)` or pass a string variable to the built-in function `dir`. You will get a list of all of the string methods you might be interested in and then some. Here are the ones that count:

```
capitalize      isidentifier      rindex
casefold        islower           rjust
center          isnumeric         rpartition
count           isprintable       rsplit
encode          isspace           rstrip
endswith        istitle           split
expandtabs      isupper           splitlines
```

```
find                join                startswith
format              ljust               strip
format_map          lower               swapcase
index               lstrip              title
isalnum             maketrans           translate
isalpha             partition           upper
isdecimal           replace             zfill
isdigit             rfind
```

If you want more information on, say, `islower`, type `help(str.islower)` into the interpreter. This functionality is available for any type.

Function specifications.

Prior note: If your code opens an input file in write mode, it will destroy the file and you will need to re-download it. The three files that might potentially be destroyed are `redact_in.txt`, `highlight_in.txt`, and `plagiarism_in.txt`, so those functions can fail even if the code is correct if previous versions of the code have destroyed the test files.

`US_to_EU`: This function takes a string representing a US style date and returns a European style date string (if it takes an argument, it must have a corresponding parameter). US strings have the month first, European strings have the day first. US uses slashes as separators, Europe uses dots. Example calls:

```
US_to_EU('3/13/18')  -->  '13.3.18'
US_to_EU('3/13/2018')  -->  '13.3.2018'
US_to_EU('03/13/2018')  -->  '13.03.2018'
```

To prepare for this function, open a shell and enter the following code:

```
date = '3/13/18'
num_lst = date.split('/')
num_lst
num_lst[0]
```

This is not code that is meant to be cut-and-paste into your file. It should teach you what you need to know to solve the problem as described – it is not itself a solution.

`is_phone_num`: This Boolean function takes a string and returns `True` if it is in the format `ddd-ddd-dddd`, where `d` is a digit, `False` otherwise. Example calls:

```
is_phone_num('123-456-7890')  -->  True
is_phone_num('123-4556-7890')  -->  False
is_phone_num('(123)456-7890')  -->  False
```

To prepare for this function, open a shell and enter the following code:

```
possible_num = '123-456-7890'
num_lst = possible_num.split('-')
```

```
num_lst
len(num_lst)
len(num_lst[0])
num_lst[0].isdigit()
```

`redact_line`: This function takes a string and returns a new string that has all of the phone numbers (as defined in the previous function description) in the original string replaced by `'XXX-XXX-XXXX'`. Assume that the argument ends in a newline. Phone numbers are bracketed by spaces, newlines, and/or the beginning of the string. You will need to use slicing with or without the `replace` method because `replace` always starts searching from the start of the string. Therefore, `replace` alone will not be enough (and is not actually necessary). `split` and `join` will also not be helpful.

```
redact_line('123-456-7890\n')  -->  'XXX-XXX-XXXX\n'
redact_line('123-456-7890 123-456-78901 123-456-7890\n') -->
            'XXX-XXX-XXXX 123-456-78901 XXX-XXX-XXXX\n'
```

This one is hard, so here's a restatement of the problem, with more implementation suggestions: you need a loop to traverse the string. Make the limit of your `for` loop `len(s) - 12`. This will make it easier. There are three requirements for having found a phone number. On the left, it must be the beginning of the line or the character prior to the potential number must be a space. The potential number (a substring of length 12) must return `True` when passed to the `is_phone_number` function. Finally, the character after the potential number must be a space or a newline. Once you check for that, you need to redact the number if all of those conditions were `True`. As stated above, you can't use `replace`. You will need:

```
line = slice up to number + 'XXX-XXX-XXXX' + slice after number
```

If you like a challenge, try this yourself. But here's the entire solution (which won't work until `is_phone_num` works):

```
def redact_line(line):
    # can't use replace only because it always starts at 0
    for i in range(len(line) - 12):
        if ((i == 0 or line[i - 1] == ' ') and
             line[i + 12] in ' \n' and is_phone_num(line[i:i + 12])):
            line = line[:i] + line[i:].replace(line[i:i + 12], "XXX-
XXX-XXXX", 1)
    return line
```

`redact_file`: This function takes a string filename. It writes a new file that has the same contents as the argument, except that all of the phone numbers are redacted. Assume that the filename has only one period in it. The new filename is the same as the original with `'_redacted'` added before the period. For instance, if the input filename were `'myfile.txt'`, the output filename would be `'myfile_redacted.txt'`. Make sure you close your output file.

The first hard task in this function is to make the output filename from the input filename that was passed in. You can break it into two pieces by splitting on the dot or by using the `index` method and slicing. Then put it back together again using string concatenation.

We are also working with files in this one. To open a file with a filename stored in the variable `fname`, in read mode use the `open` function as such: `fp = open(fname)`. You will need to open the output file in write mode: `open(fname, 'w')`. Then you can traverse the input file line-by-line with this syntax: `for line in fp:`. To write to an output file, use the write method: `fp_out.write(line)`.

`plagiarism`: This Boolean function takes two filenames. If any line occurs in both files, return `True`. If not, return `False`. I suggest using nested loops. With nested loops, we call the loop that is in the body of the other loop, the "inner loop". The loop that contains the inner loop is the "outer loop". Open the second file inside the outer loop:

```
for line1 in file1:
    file2 = open(fname2)
    for line2 in file2:
```

Python only lets you read the file once per open, so you need this order of instructions. Make sure your `return False` is outside of both loops. Otherwise, you will only compare the first two lines of the files. Make sure you close the file that gets read repeatedly after the inner loop but still inside the outer loop.

`count_word`: This function takes a filename and a keyword. Return the number of times the keyword occurs in the file. Initialize a variable to keep track of the number of times the keyword appears. Traverse the file by line. Use the `count` method on each line to find the number of times the keyword appears in it and add that amount to the variable.