COMP 1900 - Fall 2021

Lab 7: Strings

Total Points: 34

Due: Fri., Nov. 12, by 2359 CST
Please carefully read the submission instructions at the end of the assignment. As with all other assignments, this should be done *individually*.

Grader: The lab TA who supervises your registered lab section will grade your submission. Grades will generally be posted within 1-2 weeks of the assignment due date. Questions about grading? Please contact them first.

NOTE: If you happen to be familiar with regular expressions (regexes), *don't* use them for this lab. They can trivialize some things :)

1. Within a file named `word_analysis.py`, do the following:

   (a) (4 points) Write a function `avg_word_length(s)` that computes and returns the average word length in the string `s`. Assume that every "word" in the string is separated by spaces, and that the length of each "word" *does* include all non-letter characters.

   Here are some example arguments and their expected return values:

Argument	Return Value
`'sloths are nice'`	4.333333333333333
`'sloths are nice!'`	4.666666666666667
`'sloths are nice!!'`	5.0

   (b) (8 points) The function from above can be improved, since adding ending punctuation to a word can change the results even though the punctuation isn't really part of the word. Write a function `avg_word_length_better(s)` that computes and returns the average word length in the string `s`, ignoring the last character(s) of each word if they're not letters. Note that *any* ending characters that are not letters should be ignored, even if they're not valid punctuation marks. However, non-letter characters that appear at the beginning or middle of the word *should* count towards its length.

   Here are some example arguments and their expected return values:

Argument	Return Value
`'sloths are nice'`	4.333333333333333
`'sloths are nice!'`	4.333333333333333
`'5L0Ths... ARe, n1cE!!11!!'`	4.333333333333333

(c) (2 points) Below your function definitions in `word_analysis.py`, write a program that repeatedly gets user input for a string until they enter X (case-insensitive) to exit. For each string that's entered, call and show the return values from both `avg_word_length` and `avg_word_length_better`.

Example program run (underlined parts indicate what the user enters)

```
Enter a string (X to exit): It was a dark and stormy night.  Sloths were
everywhere!
Average word length: 4.6
Average word length (better): 4.4

Enter a string (X to exit): x
Thanks, come back soon!
```

2. Suppose you're starting a new sloth fan club website. You want to collect some information about your users, and for totally wholesome reasons you decide you need to get their Social Security numbers (SSNs). Since your users aren't always careful when entering data, you want to make sure that the entered SSNs are valid. We'll assume that "valid" SSNs are anything in the following formats:

```
DDDDDDDDD
DDD DD DDDD
DDD-DD-DDDD
DDD.DD.DDDD
```

where `D` is a digit from 0-9. Note that the separators between groups of digits cannot be mixed; for example, 123-45.6789 is an *invalid* SSN.

Within a file named `ssn_validator.py`, do the following:

(a) (2 points) Write a function `is_valid_sep(s)` that returns whether the string `s` is a valid separator, according to the description above. (This should be extremely short!)

Here are some example arguments and their expected return values:

Argument	Return Value
`' '`	`True`
`'-'`	`True`
`'*'`	`False`

(b) (5 points) Write a function `is_valid_ssn(ssn)` that returns whether the string `ssn` is a valid SSN, according to the description above. Call the `is_valid_sep` function as needed.

Hint: Given a string `s`, `s.isdigit()` returns either `True` or `False` depending on whether *all* characters of `s` are digits.

Here are some example arguments and their expected return values:

Argument	Return Value
`'123456789'`	`True`
`'123-45-6789'`	`True`
`'123-45,6789'`	`False`
`'12E 45 6789'`	`False`

(c) (10 points) Real-world data is rarely perfect, but we can sometimes try to improve it! Write a function `suggest_correction(ssn)` that returns a suggested "corrected" SSN (as a string) based on its string argument `ssn`. The suggested corrections work like this:

- Case 1: The argument is in the form `DDD*DD*DDDD`, where `D` is a digit from 0-9 and `*` is a separator. If either separator is valid, the function suggests using the same separator in both positions. If neither separator is valid, the function suggests omitting the separators completely.
- Case 2: The argument is in the form `DDD*DDDDDD` or `DDDDD*DDDD`. If the separator is valid, the function suggests using the same separator in both positions. If the separator is not valid, the function suggests omitting the separator completely.

If the argument doesn't fit either form above, the function returns `None`, which is a special Python value representing "no value." Call the `is_valid_sep` function as needed.

Here are some example arguments and their expected return values:

Argument	Return Value
'123456789'	None
'123-45-6789'	'123-45-6789'
'123-45,6789'	'123-45-6789'
'123*45,6789'	'123456789'
'12E 45 6789'	None
'12345 6789'	'123 45 6789'
'123.456789'	'123.45.6789'
'123*456789'	'123456789'

(d) (3 points) Below your function definitions in `ssn_validator.py`, write a program that repeatedly gets user input for an SSN (as a string) until they enter X (case-insensitive) to exit. For each string that's entered, show whether the SSN is valid. If it's invalid, also show the suggested correction if applicable. If there's no suggested correction, just show that it's invalid. Call the `is_valid_ssn` and `suggest_correction` functions as needed.

Example program run (underlined parts indicate what the user enters)

```
Enter an SSN (X to exit): 123.45.6789
Valid
Enter an SSN (X to exit): 123x45*6789
Invalid, but maybe you meant 123456789?
Enter an SSN (X to exit): 123,45-6789
Invalid, but maybe you meant 123-45-6789?
Enter an SSN (X to exit): abc-de-fghi
Invalid
Enter an SSN (X to exit): 123.456789
Invalid, but maybe you meant 123.45.6789?
Enter an SSN (X to exit): 12345,6789
Invalid, but maybe you meant 123456789?
Enter an SSN (X to exit): 12345,67890
Invalid
Enter an SSN (X to exit): x
Thanks, come back soon!
```

Code Guidelines

Points can be deducted for not following these guidelines!

- Most importantly, your code *must* run. Code that does not run due to syntax errors may receive zero credit, at the TA's discretion.
- Follow Python capitalization conventions for `variable_and_function_names`.
- Use consistent indentation throughout your code. (Python kind of forces you to do this!)
- Include a reasonable amount of comments in your code. "Reasonable" is somewhat subjective, but at the very least include:
  1. A comment at the top of each source code file and before each function summarizing what it does.
  2. Comments that indicate the major steps taken by the program. There are generally at least two or three of these, such as collecting user input or making calculations.

Need Help?

- Attend your weekly lab meeting. Your lab TA can give you live help during this session.
- Contact your lecture instructor, and/or attend office hours.
- Use the CS Discord server – however, this is *not* for other people to write code for you.
- The UofM offers free online tutoring through the Educational Support Program (ESP): `https://www.memphis.edu/esp/onlinetutoring.php`

Submission Instructions

- Create a zip file containing your Python source files. You can use any Python development environment you like, as long as you submit the source files.
- Upload your zip file to the appropriate dropbox folder on eCourseware. The dropbox *will* cut off submissions at precisely the stated deadline, so please submit with some time to spare. Late submissions are not accepted. You may submit as many times as you want up to the deadline. Each submission overwrites the previous one.
- Contact your lab TA to schedule a one-on-one code review via Zoom. During this short meeting (15-30 minutes), your TA will ask you to run your code. They may also ask you to explain your code and/or run different test cases. The code review must be completed for you to get a grade for the lab.

  Note that since the code review happens after your submission, its purpose is *not* for you to get help with writing your code! Getting help should be done during the scheduled lab sessions, or by reaching out to your lecture instructor/online tutors *before* the due date.