CS50's Introduction to Programming with Python

OpenCourseWare

Donate (https://cs50.harvard.edu/donate)

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan)
(https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/)
(https://www.reddit.com/user/davidjmalan)
(https://www.threads.net/@davidjmalan) (https://twitter.com/davidjmalan)

Regular, um, Expressions

It's not uncommon, in English, at least, to say "um" when trying to, um, think of a word. The more you do it, though, the more noticeable it tends to be!

In a file called `um.py`, implement a function called `count` that expects a line of text as input as a `str` and returns, as an `int`, the number of times that "um" appears in that text, case-insensitively, as a word unto itself, not as a substring of some other word. For instance, given text like `hello, um, world`, the function should return `1`. Given text like `yummy`, though, the function should return `0`.

Structure `um.py` as follows, wherein you're welcome to modify `main` and/or implement other functions as you see fit, but you may not import any other libraries. You're welcome, but not required, to use `re` and/or `sys`.

```
import re
import sys


def main():
    print(count(input("Text: ")))


def count(s):
    ...


...
```

```
if __name__ == "__main__":
    main()
```

Either before or after you implement `count` in `um.py`, additionally implement, in a file called `test_um.py`, **three or more** functions that collectively test your implementation of `count` thoroughly, each of whose names should begin with `test_` so that you can execute your tests with:

```
pytest test_um.py
```

▼ **Hints**

- Recall that the `re` module comes with quite a few functions, per docs.python.org/3/library/re.html (https://docs.python.org/3/library/re.html), including `findall`.
- Recall that regular expressions support quite a few special characters, per docs.python.org/3/library/re.html#regular-expression-syntax (https://docs.python.org/3/library/re.html#regular-expression-syntax).
- Because backslashes in regular expressions could be mistaken for escape sequences (like `\n`), best to use Python's raw string notation for regular expression patterns (https://docs.python.org/3/library/re.html#module-re). Just as format strings are prefixed with `f`, so are raw strings prefixed with `r`. For instance, instead of `"harvard\.edu"`, use `r"harvard\.edu"`.
- Note that `\b` is "defined as the boundary between a `\w` and a `\W` character (or vice versa), or between `\w` at the beginning/end of the string," per docs.python.org/3/library/re.html#regular-expression-syntax (https://docs.python.org/3/library/re.html#regular-expression-syntax).
- You might find regex101.com (https://regex101.com/) or regexr.com (https://regexr.com/) helpful for testing regular expressions (and visualizing matches).
- See thefreedictionary.com/words-containing-um (https://www.thefreedictionary.com/words-containing-um) for some words that contain "um".

Demo

```
Input: hello, um, world
1
$ python um.py
Input: um, hello, um, world
2
$ python um.py
Input: um...
1
$ python um.py
Input: yum
0
$
```

Recorded with asciinema

Before You Begin

Log into cs50.dev (https://cs50.dev/), click on your terminal window, and execute `cd` by itself. You should find that your terminal window's prompt resembles the below:

```
$
```

Next execute

```
mkdir um
```

to make a folder called `um` in your codespace.

Then execute

```
cd um
```

to change directories into that folder. You should now see your terminal prompt as `um/ $`. You can now execute

```
code um.py
```

to make a file called `um.py` where you'll write your program. Be sure to also execute

```
code test_um.py
```

to create a file called `test_um.py` where you'll, um, write tests for your program.

How to Test

How to Test `um.py`

Here's how to test `um.py` manually:

- Run your program with `python um.py`. Ensure your program prompts you for an input. Type `um`, followed by Enter. Your `count` function should return `1`.
- Run your program with `python um.py`. Type `um?`, followed by Enter. Your `count` function should return `1`.
- Run your program with `python um.py`. Type `Um, thanks for the album.`, followed by Enter. Your `count` function should return `1`.
- Run your program with `python um.py`. Type `Um, thanks, um...`, followed by Enter. Your `count` function should return `2`.

How to Test `test_um.py`

To test your tests, run `pytest test_um.py`. Try to use correct and incorrect versions of `um.py` to determine how well your tests spot errors:

- Ensure you have a correct version of `um.py`. Run your tests by executing `pytest test_um.py`. `pytest` should show that all of your tests have passed.
- Modify the `count` function in the correct version of `um.py`. `count` might, for example, mistakenly also count any "um" that is part of a word. Run your tests by executing `pytest test_um.py`. `pytest` should show that at least one of your tests has failed.
- Again modify the `count` function in the correct version of `um.py`. `count` might, for example, mistakenly only match an "um" that is surrounded on either side by a space. Run your tests by executing `pytest test_um.py`. `pytest` should show that at least one of your tests has failed.

You can execute the below to check your code using `check50`, a program that CS50 will use to test your code when you submit. But be sure to test it yourself as well!

```
check50 cs50/problems/2022/python/um
```

Green smilies mean your program has passed a test! Red frownies will indicate your program output something unexpected. Visit the URL that `check50` outputs to see the input `check50` handed to your program, what output it expected, and what output your program actually gave.

How to Submit

In your terminal, execute the below to submit your work.

```
submit50 cs50/problems/2022/python/um
```