CS 1110, LAB 3: STRINGS; TESTING

http://www.cs.cornell.edu/courses/cs1110/2017sp/labs/lab03.pdf

First Name: _____ **Last Name**: _____ **NetID**: _____

Getting Credit: Deadline: the first 10 minutes of (your) lab **two weeks from now (Tue Feb 28 or Wed Mar 1)**, due to February break. The checking-off procedure is *almost* the same as before,[1] *except* that you can also get this lab checked off at a one-on-one session if you sign up and attend one starting before 3:45pm on Wed Mar 1. An announcement explaining what one-on-ones are will be made later.

1. String Methods

A method is a special kind of function that is associated with items of a specific type. Because of this association, for *string* methods, the way you call one is to first use the name of a particular string literal or variable storing a string, followed by a period, followed by the name of the method, and then the usual parentheses and arguments. For example, the following are all valid Python method calls, assuming they are made in the order depicted:

```
s=' DRINK ME '.lower()  # s is ' drink me '
s.find('d')
'CS 1110'.find('1')
s.strip().find(' ')     # s.strip() returns a string, whose find() method can be called.
                        # The value returned is 5, but s is still ' drink me '.
```

As you can see from the last comment above, string methods don't change their "owner" string.[2]

Course authors: E. Andersen, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White. This lab has additional contributions by Stephen McDowell about handling zip files on Windows.

[1] In case you've forgotten, here's a reminder: Show this handout and/or your code to a staff member either (a) during your lab 03 session, (b) in consulting hours listed at http://www.cs.cornell.edu/courses/cs1110/2017sp/about/staff.php up to the day **before** your next scheduled lab section, or (c) in the first 10 minutes of (your) next scheduled lab (Tue Feb 28 or Wed Mar 1). Beyond that time, the staff have been instructed not to give you credit. Labs are graded on effort, not correctness. We just want to see that you tried all the exercises, and to clarify any misunderstandings or questions you have.

[2] Even experienced Python programmers sometimes forget this, from time to time, to our shame.

Start up Python interactive mode[3] and enter the *second* line below, the one with a mixed-case word[4]:

```
#    0123456789      These numbers show you the indices of the first 10 characters
s = 'HeLLo WorLd!'
```

Now fill in the tables below, as usual. We've provided a visual guide to the indices of the first characters in `s` in the first comment above.

Expression	Expected Value	Calculated Value
`s[1]`		
`s[15]`		
`s[1:5]`		
`s[:5]`		
`s[4:]`		
`'e' in s`		
`'x' in s`		

Expression	Expected Value	Calculated Value
`s.index('e')`		
`s.find('e')`		
`s.index('x')`		
`s.find('x')`		
`s.count('o')`		
`s.index('L',5)`		
`# this means, "look starting from index 5"`		

Next, fill in the last column with a literal that, if it were place in the box, would make the expression evaluate to the indicated desired value. We've done the first one for you.

Expression	Desired Value	Literal in the Box
☐`[1]`	`'i'`	`'Hi'` or `'Fie'` are valid responses, but not `'i'` (error) or `'iH'`
`s[`☐`]`	`'W'`	
☐`[2]`	`'C'`	
`s[4:`☐`]`	`'o Wo'`	
`s[:`☐`]`	`'He'`	
`s[`☐`:]`	`'rLd!'`	
`s.find(`☐`)`	4	
`s.count(`☐`)`	3	

[3]Enter `python` at the command shell prompt.

[4]We made the "ell"s uppercase so you wouldn't think they are "one"s. In fact, we advise you never to use the letter l ("ell") as a variable name, because of its confusability with the digit 1.

2. Unit Tests For Testing Function `replace_first()`

Suppose we wanted to write a function with the following specification:

> `replace_first(word,target,rep)` returns a copy of string `word` with the *first* instance of string `target` in `word` replaced by string `rep`. Precondition: `target` has length ≥ 1 , and occurs at least once in `word`.

2.1. **Write test cases.** Our first step in writing such a function is to develop a good set of *representative* test cases, developed just from reading the specification alone. Complete the missing entries below; note that we're asking you to invent at least one additional full test case yourself.

Test cases for `replace_first(word,target,rep)`				
`word`	`target`	`rep`	**Desired output**	**Case motivation**
`'crane'`	`'a'`	`'o'`	`'crone'`	single-letter `target` and `rep`
`'POLL'`	`'L'`	`'o'`		multiple occurrences of `target`
`'a bird'`	`'bird'`	`'plane'`		

Why **don't** you need to test the case where `target` is the empty string `''`?

2.2. **Enocde your test cases in test procedure `test_replace_first()`.**

Create a new directory on your hard drive for this lab's files. Then, download into that new directory the files you need for lab 03; get them from the Labs section of the course web page, http://www.cs.cornell.edu/courses/cs1110/2017sp/labs . You can do this either by individually downloading each of the `.py` files, *or* get all of the files bundled in a single ZIP file called `lab03.zip`; you can turn a ZIP file into a folder by double clicking on it. (Windows users: see footnote.[5])

In file `lab03.py`, there's a possibly incorrect implementation of `replace_first`. But don't look at it yet! Instead open separate file `lab03_test.py` in Komodo Edit. In it is an incomplete test procedure, `test_replace_first()`, for checking `lab03.replace_first`. Notice that in lines 17-19, we've encoded the first test case from the table above for you, using `assert_equals` from module `cornelltest`.

[5] Windows has an odd way of dealing with ZIP files, so Windows users will need to either (a) drag the folder contents to *another* folder before using them, or (b) right-click and choose "Extract Here.". (The issue is that the Windows default is that when you double click on a .zip file, it does not actually extract it. The file browser is simply uncompressing on the fly. The same thing happens these days in Google drive.)

Open a command shell and navigate[6] to your new directory with the lab 03 files in it. Then, run Python on `lab03_test.py`.[7] You should get the output `Module lab03: all tests passed`. But, this is because there aren't enough test cases in `test_replace_first()`!

Finish `test_replace_first()` by encoding the remaining test cases from the table above into the test procedure. Use lines 17-19 as a guide.

Then, fix the header comments (lines 2-3) to have your name, netID, and the date.

Now, save `lab03_test.py`, and, in the command shell, run Python on the file again. You should get an error: what are the last three lines of the error message?

2.3. **Hunt Down Errors.** Unit tests are great at detecting errors. But they do not necessarily tell you what line(s) of code are the root cause of these errors.

In particular, open `lab03.py` in Komodo Edit and look at procedure `replace_first()`. The error could have occurred at any of its several lines of code. But the error message you recorded above doesn't even mentioned the right file!

We often use `print` statements to help us isolate an error. These statements allow us to *inspect* a variable immediately after it is assigned a value.

Look at the comments explaining what the variables `pos`, `before`, `after`, and `result` are supposed to mean. *According to those comments*, *not* the code itself, for the test case `word: 'POLL', target: 'L', rep: 'o'`, what *should* the values of these four variables be?

pos: before: after: result:

Let's add a `print` statement to inspect the variable `pos`. Inside of `replace_first`, *right after* the assignment to `pos`, add, properly indented, the informative statement

```
print "DEBUG: pos is: " + str(pos)
```

Do the analogous thing for the other three variables, `before`, `after`, and `result`.

Interlude: get a copy of your work off the lab machines! If you are working on a lab machine, know that **your files will be automatically deleted at some point soon after you log out or are auto-logged out**. It is therefore vitally important that, as you get near the end of the lab, **GET A COPY OF YOUR FILES TO YOURSELF — MAIL THEM TO YOURSELF, SAVE THEM TO A USB FlASH DRIVE**, or whatever works for you.

[6] Use the `cd` commands you practiced in Lab 2; see http://www.cs.cornell.edu/courses/cs1110/2017sp/materials/command.php for our documentation.

[7] That is, in the command shell, enter `python lab03_test.py`

Save `lab03.py`, then on the command shell, then run the *test* program again.[8]. Before you see the error message, you should see four DEBUG lines printed to the screen. These are the result of your print statements. The output helps you "visualize" what is going on in `replace_first()`. What does the output tell you, for the test case `word: 'POLL', target: 'L', rep: 'o'`, that the four variables `pos`, `before`, `after`, and `result` are *actually* set to by the code?

pos:	before:	after:	result:

By comparing your "should" answers two boxes above with your "actually" answers in the box above, you should see that there is a problem with the variable `pos`. Look where there's an assignment to `pos` in `replace_first`. What is the error in that assignment statement?[9]

Fix the error, save your files, and test the procedure again by running the test script. You should get the second test case, for `'POLL'`, passing without an error message.

But, you should get *another error*, for the "bird"/"plane" example. Using the methodology above — identifying what the variable values "should" be, and then comparing them to what your `print` statements tell you they "actually" are — how should another line in `replace_first` be corrected?

Fix the error, test your program, and repeat until all your test cases pass! What other errors did you find, if any?

2.4. **Clean up `replace_first()`.** While your print statements proved very useful for debugging, you do not want those print statements showing information on the screen every time you run the procedure.[10]

So once you are sure the program is running correctly, you should remove all of your debugging print statements. You can either comment them out (fine in small doses, as long as it does not make your code unreadable), or you can delete them entirely.

However, once you remove these, it is important that you test the procedure one last time. You want to be sure that you did not delete the wrong line of code by accident. Run the test script one last time to make sure no errors were introduced by your deletions.

[8] Enter `python lab03_test.py`.

[9] Hint: what does `rfind` do? You might use Google or another search engine to find out.

[10] In fact, their presence technically violates the function specification, since no mention of printing is made there.

3. Optional: Complete and Test a Function on Assignment 1

You do *not* have to complete this section to get checked off for this lab, but, since this section involves some work you need to do for assignment 1, we highly recommend that you work on this with a staff member, either in lab, consulting hours, or in the one-on-ones that will be announced in lecture.

Write test cases for function `first_in_double_quotes`, using the specification in `lab03.py`. Put these test cases in the skeleton test procedure in `lab03_test.py`.

Uncomment the call to the test procedure in the second to last line of `lab03_test.py`, so that your test procedure is actually called by the test script!

Run your test script, to make sure it actually reports that it is testing `first_in_double_quotes`, and that at least one test case for `first_in_double_quotes` fails.

Now replace the code "dummy" return statement in `first_in_double_quotes`, in file `lab03.py`. with code that makes it work, i.e., satisfy its specification.

Test and debug your implementation using print-statement and unit-test methodology you engaged in for `replace_first()` above.