Carleton University
Department of Systems and Computer Engineering
ECOR 1051 - Fundamentals of Engineering I

Lab 5 - Python's `str` Type

Objectives

- To learn about Python's built-in `str` type, which represents textual data.
- To gain more experience using the Python shell to build software experiments
- To apply *Practical Programming*'s function design recipe (FDR) to develop simple functions that process text.

Learning outcomes: 1, 4, 5; Graduate attributes: 1.3, 5.3 (see the course outline)

Getting Started

Begin by creating a new file within Wing 101. **Save it as lab5.py**

Part A - Software Experiments

In this first part, we will return to working in the Python Shell just as you did in the first couple of labs. You will perform a series of exercises either in Wing 101 or in Python Tutor, writing your answers in simple text file. The template for this file is provided on CULearn as **`lab5.txt`**.

In your file lab5.py, put three double quotes on the first line and three double quotes on the second line. Now copy and paste all of file lab5.txt **between** those two lines. This effectively creates a big comment block at the top of lab5.py where you will insert your answers to Part A. (This is so that you only need to submit one file for this lab.)

Part B - Designing Functions that Process Text

All of the code for all the following exercises should be placed in lab5.py **below** part A using the same layout described previously in Lab 4 (import, functions, main script).

For this lab (and all subsequent work in this course), function headers must contain a docstring and type annotations, as described in the lectures on the Function Design Recipe.

Exercise 4 (Adapted from *Practical Programming*, Chapter 4, Exercise 8)

Step 1: Type this code in the editor window:

```
def repeat(s: str, n: int) -> str:
    """ Return s repeated n times; if n is negative, return the
    empty string.
```

```
    >>> repeat('yes', 4)
    'yesyesyesyes'
    >>> repeat('no', 0)
    >>> repeat('no', -2)
    >>> repeat('yesnomaybe', 3)
    """
```

Step 2: Notice that the docstring is actually incomplete! The first test command (`repeat('yes,4)` is followed by its ***expected result*** (`'yesyesyesyes'`) but the next three test commands are missing their ***expected results***.

Please enter in the expected results for the final three test commands (Insert a line between each one).

Step 3: Write the body of the function. To convince yourself that your function's body is correct, you may [optionally] begin by testing your function using the Python shell

1. Hit the green arrow button to execute the function's definition (to see if the code has any syntax errors to fix up)
2. In the Python shell, type each of the test commands listed in the docstring.

   Example: `>>> repeat('yes',4)`

To finish, though, you must write the tests of the function as call expressions below in the main script.

Example:
```
result = repeat('yes', 4)
print (result)
```

Exercise 5 (Adapted from *Practical Programming*, Chapter 4, Exercise 9)

Continue adding code to your existing file. Follow the same steps from the previous exericse to complete and test the next function.

```
def total_length(s1: str, s2: str) -> int:
    """ Return the sum of the lengths of s1 and s2.

    >>> total_length('yes', 'no')
    5
    >>> total_length('yes', '')
    >>> total_length('YES!!!!', 'Noooooo')
    """
```

Exercise 6

Continue adding code to your existing file. You will write a third function but this time, we won't give you the starting code.

Design, code and test the definition of a function named `replicate`. This function has one argument, which is a string. It returns a new string that contains one or more copies of the argument. The number of copies is equal to the number of characters in the original string. For example, when called this way: `replicate('a')`, the function returns `'a'`. When called this way, `replicate('abc')`, the function returns `'abcabcabc'`. (This string contains three copies of `'abc'` because the argument has three characters.)

Wrap Up

Ensure that your code meets the posted marking rubrics for the labs.

Submit (only) file lab5.py. Ensure that your work from Part A is at the top of your file, as per the description given on the first page.

You are required to keep a backup copy of (all) your work for the duration of the term.

Last edited: April 23, 2020