Carleton University
Department of Systems and Computer Engineering
ECOR 1041 - Computation and Programming

Lab 6 - 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.

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 in Wing 101 or Python Tutor, writing your answers in a simple text file. The template for this file is provided on Brightspace as **`lab6.txt`.**

Part B - Designing Functions that Process Text

In this second part, we will move back to creating-and-saving your work in .py files. All the code for all the following exercises should be placed in the same file 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.

Begin by creating a new file within Wing 101. Save it as lab6.py

Exercise 4 (…/20)

Step 1: Type this code in the editor window:

```
def concatenates_total_length(s1: str, s2: str) -> str:
     """ The function returns the concatenated string of the two arguments
          (s1 and s2) and the length of the concatenated string.

    >>> concatenates_total_length('Hello, ', 'world')
    "Hello, world" has a length of 12 characters

    >>> concatenates_total_length('Welcome to ', 'Canada')

    >>> concatenates_total_length('ECOR', '1041')

    >>> concatenates_total_length('', '')

    """
```

Step 2: Notice that the docstring is incomplete! The first test command (`concatenates_total_length('Hello, ', 'world')` is followed by its ***expected result*** (`"Hello, world" has a length of 12 characters`), but the following three test commands are missing their ***expected results***.
Complete the docstring with the expected results for the rest of the examples (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 test command listed in the docstring.

   Example: `>>> concatenates_total_length('Hello, ', 'world')`

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

Example:
```
result = concatenates_total_length('Hello, ', 'world')
print (result)
```

Hint: Review Phyton's built-in functions that start by "L" (https://docs.python.org/3/library/functions.html#). You may find that useful to solve this exercise!

Exercise 5 (…/20)

Continue adding code to your existing file. Follow the steps from the previous exercise to complete and test the following function.

```
def replicate(s1: str, s2: str, rep: int) -> str:
    """
    The function returns a string that is the result of concatenating the
    two string arguments and replicating the concatenated string "rep"
    times.

    >>> replicate("a", "b" , 2)
    'abab'
    >>> replicate("ab" ,"c" , 2)
    'abcabc'
    >>> replicate("abc", "d", 3)
    'abcdabcdabcd'
    """
```

Exercise 6 (…/30)

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

Design, code, and test the definition of a function named `to_string`. This function has one input parameter, which is an integer, and returns the integer converted into a string. For example, when called this way: `to_string(10)`, the function returns `'10'`. When called this way, `to_string(987)`, the function returns `'987'`.

Ensure to follow FDR when writing your function.

Wrap Up

You need to submit lab6.**py**.

Copy your entire lab6.**txt** file to the top of lab6.**py**. lab6.**txt** starts and ends with three double-quotes. Thus, lab6.**txt** will be a comment at the top of **your lab6.py file**. We recommend that you ensure that your lab6.py still runs after you have added the text file.

- Make sure that you included your name and student number
- Check proper use constants (UPPER_CASE) and variables (lower_case) (There is a 10/100 deduction for misuse of UPPPER & lower case)
- Check the indents of the function bodies. (There is a 10/100 deduction for misuse of indentation)
- Check file organization: (1) imports, (2) **all** function definitions; (2) Main Script (There is a 10/100 deduction for not organizing the file according to the instructions)
- Confirm that your filename **lab6.py** matches exactly.
- Confirm that your .py script runs properly, otherwise the TA will also assign a zero.
- Submit the file on Brightspace.

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

Last edited: January 24, 2022