Python review: Values, variables, types, lists, and strings

These first few notebooks are a set of exercises with two goals:

1. Review the basics of Python
2. Familiarize you with Jupyter

Regarding the first goal, these initial notebooks cover material we think you should already know from Chris Simpkins's Python Bootcamp from Fall 2018. This bootcamp is what students on the on-campus Georgia Tech MS Analytics students took.

Regarding the second goal, you'll observe that the bootcamp has each student install and work directly with the Python interpreter, which runs locally on his or her machine (e.g., see the `Video: Getting Started` link and Slide 7 of his Intro to Python slides). But in this course, we are using Jupyter Notebooks as the development environment. You can think of a Jupyter notebook as a web-based "skin" for running a Python interpreter---possibly hosted on a remote server, which is the case in this course. Here is a good tutorial on Jupyter.

> **Note for MS Analytics students.** In this course we assume you are using Vocareum's deployment of Jupyter. You also have an option to use other Jupyter environments, including installing and running Jupyter on your own system. We can't provide technical support to you if you choose to go those routes, but if you'd like to do that anyway, we recommend Microsoft Azure Notebooks as a web-hosted option or the Continuum Analytics Anaconda distribution as a locally installed option.

Study hint: Read the test code! You'll notice that most of the exercises below have a place for you to code up your answer followed by a "test cell." That's a code cell that checks the output of your code to see whether it appears to produce correct results. You can often learn a lot by reading the test code. In fact, sometimes it gives you a hint about how to approach the problem. As such, we encourage you to try to read the test cells even if they seem cryptic, which is deliberate!

Exercise 0 (1 point). Run the code cell below. It should display the output string, `Hello, world!`.

In [1]:
```
print("Hello, world!")
```

```
Hello, world!
```

Exercise 1 (`x_float_test`: 1 point). Create a variable named `x_float` whose numerical value is one (1) and whose type is *floating-point*.

In [4]:
```
#
# YOUR CODE HERE
x=1
print (x)
```

```
1
```

In [ ]:
```
# `x_float_test`: Test cell
assert x_float == 1
assert type(x_float) is float
print("\n(Passed!)")
```

Exercise 2 (`strcat_ba_test`: 1 point). Complete the following function, `strcat_ba(a, b)`, so that given two strings, `a` and `b`, it returns the concatenation of bfollowed by a (pay attention to the order in these instructions!).

In [ ]:
```
def strcat_ba(a, b):
    assert type(a) is str
    assert type(b) is str
#
# YOUR CODE HERE
#
```

In [ ]:
```
# `strcat_ba_test`: Test cell

# Workaround:  # Python 3.5.2 does not have `random.choices()` (available in 3.6+)
def random_letter():
    from random import choice
    return choice('abcdefghijklmnopqrstuvwxyz')

def random_string(n, fun=random_letter):
    return ''.join([str(fun()) for _ in range(n)])

a = random_string(5)
b = random_string(3)
c = strcat_ba(a, b)
print('strcat_ba("{}", "{}") == "{}"'.format(a, b, c))
assert len(c) == len(a) + len(b)
assert c[:len(b)] == b
assert c[-len(a):] == a
print("\n(Passed!)")
```

Exercise 3 (`strcat_list_test`: 2 points). Complete the following function, `strcat_list(L)`, which generalizes the previous function: given a *list* of strings, `L[:]`, returns the concatenation of the strings in reverse order. For example:

```
strcat_list(['abc', 'def', 'ghi']) == 'ghidefabc'
```

In [ ]:
```
def strcat_list(L):
    assert type(L) is list
    #
    # YOUR CODE HERE
    #
```

In [ ]:
```
# `strcat_list_test`: Test cell
n = 3
nL = 6
L = [random_string(n) for _ in range(nL)]
Lc = strcat_list(L)

print('L == {}'.format(L))
print('strcat_list(L) == \'{}\''.format(Lc))
assert all([Lc[i*n:(i+1)*n] == L[nL-i-1] for i, x in zip(range(nL), L)])
print("\n(Passed!)")
```

Exercise 4 (`floor_fraction_test`: 1 point). Suppose you are given two variables, `a` and `b`, whose values are the real numbers, $a \geq 0$ a≥0 (non-negative) and $b > 0$ b>0 (positive). Complete the function, `floor_fraction(a, b)` so that it returns $\lfloor \frac{a}{b} \rfloor$ ⌊ab⌋, that is, the *floor* of $\frac{a}{b}$ab. The *type* of the returned value must be `int` (an integer).

In [ ]:
```
def is_number(x):
    """Returns `True` if `x` is a number-like type, e.g., `int`, `float`, `Decimal()`, ..."""
    from numbers import Number
    return isinstance(x, Number)

def floor_fraction(a, b):
    assert is_number(a) and a >= 0
    assert is_number(b) and b > 0
    #
    # YOUR CODE HERE
    #
```

In [ ]:
```
# `floor_fraction_test`: Test cell
from random import random
a = random()
b = random()
c = floor_fraction(a, b)

print('floor_fraction({}, {}) == floor({}) == {}'.format(a, b, a/b, c))
assert b*c <= a <= b*(c+1)
assert type(c) is int
print('\n(Passed!)')
```

Exercise 5 (`ceiling_fraction_test`: 1 point). Complete the function, `ceiling_fraction(a, b)`, which for any numeric inputs, `a` and `b`, corresponding to real numbers, $a \geq 0$ a≥0 and $b > 0$ b>0, returns $\lceil \frac{a}{b} \rceil$ ⌈ab⌉, that is, the *ceiling* of $\frac{a}{b}$ab. The type of the returned value must be `int`.

In [ ]:
```
def ceiling_fraction(a, b):
    assert is_number(a) and a >= 0
    assert is_number(b) and b > 0
    #
    # YOUR CODE HERE
    #
```

In [ ]:
```
# `ceiling_fraction_test`: Test cell
from random import random
a = random()
b = random()
c = ceiling_fraction(a, b)
print('ceiling_fraction({}, {}) == ceiling({}) == {}'.format(a, b, a/b, c))
assert b*(c-1) <= a <= b*c
assert type(c) is int
print("\n(Passed!)")
```

Exercise 6 (`report_exam_avg_test`: 1 point). Let `a`, `b`, and `c` represent three exam scores as numerical values. Complete the function, `report_exam_avg(a, b, c)` so that it computes the average score (equally weighted) and returns the string, `'Your average score: XX'`, where `XX` is the average rounded to one decimal place. For example:

```
report_exam_avg(100, 95, 80) == 'Your average score: 91.7'
```

In [ ]:
```
def report_exam_avg(a, b, c):
    assert is_number(a) and is_number(b) and is_number(c)
    #
    # YOUR CODE HERE
    #
```

In [ ]:
```
# `report_exam_avg_test`: Test cell
msg = report_exam_avg(100, 95, 80)
print(msg)
```

```
assert msg == 'Your average score: 91.7'

print("Checking some additional randomly generated cases:")
for _ in range(10):
    ex1 = random() * 100
    ex2 = random() * 100
    ex3 = random() * 100
    msg = report_exam_avg(ex1, ex2, ex3)
    ex_rounded_avg = float(msg.split()[-1])
    abs_err = abs(ex_rounded_avg*3 - (ex1 + ex2 + ex3)) / 3
    print("{}, {}, {} -> '{}' [{}]".format(ex1, ex2, ex3, msg, abs_err))
    assert abs_err <= 0.05

print("\n(Passed!)")
```

Exercise 7 (`count_word_lengths_test`: 2 points). Write a function `count_word_lengths(s)` that, given a string consisting of words separated by spaces, returns a list containing the length of each word. Words will consist of lowercase alphabetic characters, and they may be separated by multiple consecutive spaces. If a string is empty or has no spaces, the function should return an empty list.

For instance, in this code sample,

```
count_word_lengths('the quick  brown   fox jumped over     the lazy  dog') == [3, 5, 5, 3, 6, 4, 3, 4, 3]
```

the input string consists of nine (9) words whose respective lengths are shown in the list.

In [ ]:
```
def count_word_lengths(s):
    assert all([x.isalpha() or x == ' ' for x in s])
    assert type(s) is str
    #
    # YOUR CODE HERE
    #
```

In [ ]:
```
# `count_word_lengths_test`: Test cell

# Test 1: Example
qbf_str = 'the quick brown fox jumped over the lazy dog'
qbf_lens = count_word_lengths(qbf_str)
print("Test 1: count_word_lengths('{}') == {}".format(qbf_str, qbf_lens))
assert qbf_lens == [3, 5, 5, 3, 6, 4, 3, 4, 3]

# Test 2: Random strings
from random import choice # 3.5.2 does not have `choices()` (available in 3.6+)
#return ''.join([choice('abcdefghijklmnopqrstuvwxyz') for _ in range(n)])

def random_letter_or_space(pr_space=0.15):
    from random import choice, random
    is_space = (random() <= pr_space)
    if is_space:
        return ' '
    return random_letter()

S_LEN = 40
W_SPACE = 1 / 6
rand_str = random_string(S_LEN, fun=random_letter_or_space)
rand_lens = count_word_lengths(rand_str)
print("Test 2: count_word_lengths('{}') == '{}'".format(rand_str, rand_lens))
c = 0
while c < len(rand_str) and rand_str[c] == ' ':
    c += 1
for k in rand_lens:
    print("  => '{}'".format (rand_str[c:c+k]))
    assert (c+k) == len(rand_str) or rand_str[c+k] == ' '
    c += k
    while c < len(rand_str) and rand_str[c] == ' ':
        c += 1

# Test 3: Empty string
print("Test 3: Empty strings...")
assert count_word_lengths('') == []
assert count_word_lengths('    ') == []

print("\n(Passed!)")
```

Fin! You've reached the end of this part. Don't forget to restart and run all cells again to make sure it's all working when run in sequence; and make sure your work passes the submission process. Good luck!