Notebook 1 Part 0

Computing for Data Analysis (Georgia Institute of Technology)

0-basics

September 6, 2024

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

These first few notebooks are a set of exercises designed to reinforce various aspects of Python programming.

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!

Debugging tip: Read assertions. The test cells often run an `assert` statement to see whether some condition that it thinks should be true is true. If an assertion fails, look at the condition being checked and use that as a guide to help you debug. For example, if an assertion reads, `assert a + b == 3`, and that fails, inspect the values and types of `a` and `b` to help determine why their sum does not equal 3.

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* (i.e., `float`).

```
In [6]: ###
        ### YOUR CODE HERE
        ###
        x_float = 1.0
```

```
In [7]: # `x_float_test`: Test cell
        assert x_float == 1, f"`x_float` has the wrong value ({x_float} rather than 1.0)"
        assert type(x_float) is float, f"`type(x_float)` == {type(x_float)} rather than `float
        print("\n(Passed!)")
```

```
(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 `b` followed by `a` (pay attention to the order in these instructions!).

```
In [10]: def strcat_ba(a, b):
             assert type(a) is str, f"Input argument `a` has `type(a)` is {type(a)} rather tha
             assert type(b) is str, f"Input argument `b` has `type(b)` is {type(b)} rather tha
             return b + a
         ###
         ### YOUR CODE HERE
         ###
```

```
In [11]: # `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), "`c` has the wrong length: {len(c)} rather than {le
         assert c[:len(b)] == b
         assert c[-len(a):] == a
         print("\n(Passed!)")
```

```
strcat_ba("nltst", "sfd") == "sfdnltst"

(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 [13]: def strcat_list(L):
             assert type(L) is list
             for item in L:
                 assert type(item) is str
             return ''.join(L[::-1])
             ###
             ### YOUR CODE HERE
             ###
```

```
In [14]: # `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!)")
```

```
L == ['nrx', 'hhx', 'wkh', 'vll', 'wmf', 'nso']
strcat_list(L) == 'nsowmfvllwkhhhxnrx'

(Passed!)
```

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

```
In [21]: 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
             return int(a//b)
             ###
             ### YOUR CODE HERE
             ###
```

```
In [22]: # `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, f"type(c) == {type(c)} rather than `int`"
         print('\n(Passed!)')
```

```
floor_fraction(0.03584745083823737, 0.07978044386535865) == floor(0.44932628977015054) == 0

(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, $a \geq 0$ and $b > 0$, returns $\lceil \frac{a}{b} \rceil$, that is, the *ceiling* of $\frac{a}{b}$. The type of the returned value must be `int`.

```
In [23]: def ceiling_fraction(a, b):
             assert is_number(a) and a >= 0
             assert is_number(b) and b > 0
             ###
             ### YOUR CODE HERE
             ###
             result = a/b
             if result > int(result):
                 return int(result) +1
             else:
                 return int(result)
```

```
In [24]: # `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!)")
```

```
ceiling_fraction(0.30719367099950756, 0.4003670185701954) == ceiling(0.7672801623284762) == 1

(Passed!)
```

```
In [25]: a = 0.3
         b = 0.1
         c = ceiling_fraction(a, b)
         print(f"{a/b}")
         print('ceiling_fraction({}, {}) == ceiling({}) == {}'.format(a, b, a/b, c))
         assert b*(c-1) <= a <= b*c
         assert type(c) is int
```

```
2.9999999999999996
ceiling_fraction(0.3, 0.1) == ceiling(2.9999999999999996) == 3
```

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 [26]: def report_exam_avg(a, b, c):
             #assert is_number(a) and is_number(b) and is_number(c)
             ###
             ### YOUR CODE HERE
             ###
             exam_avg = (a + b + c) / 3
             rounded_avg = round(exam_avg, 1)
             return f'Your average score: {rounded_avg}'
```

```
In [27]: # `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!)")
```

```
Your average score: 91.7
Checking some additional randomly generated cases:
51.25205642329396, 76.38482164221789, 92.07887061629378 -> 'Your average score: 73.2' [0.038582
67.98513361626014, 91.01837871327524, 95.0684909722901 -> 'Your average score: 84.7' [0.0093322
13.154727506368246, 65.20425668116854, 29.12402462099337 -> 'Your average score: 35.8' [0.0276
64.49522207729626, 53.17208480701435, 10.101147952677858 -> 'Your average score: 42.6' [0.0105
85.7047282720495, 26.901005707707103, 29.437303607422027 -> 'Your average score: 47.3' [0.0476
86.78766011369554, 76.42926742237243, 35.691523031832 -> 'Your average score: 66.3' [0.0028168
17.81131217275921, 50.58313873108129, 92.82552738775752 -> 'Your average score: 53.7' [0.03999
32.750734092840425, 24.663438749083767, 56.49895180292843 -> 'Your average score: 38.0' [0.028
78.88485760484218, 0.5667632362435504, 83.26106954444839 -> 'Your average score: 54.2' [0.0375
58.56922505912513, 89.84528824992393, 24.067562640468086 -> 'Your average score: 57.5' [0.0059

(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
```

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

```
In [28]: def count_word_lengths(s):
             assert all([x.isalpha() or x == ' ' for x in s])
             assert type(s) is str

             words = s.split()

             word_length = [len(word) for word in words]
             return word_length
             ### YOUR CODE HERE
             ###
```

```
In [29]: # `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(count_word_lengths('the'))

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

```
Test 1: count_word_lengths('the quick brown fox jumped over the lazy dog') == [3, 5, 5, 3, 6, 4
Test 2: count_word_lengths('gufy ebc xw wqtzjtevsrxvj  obcbkqh ghu o') == '[4, 3, 2, 13, 7, 3,
  => 'gufy'
  => 'ebc'
  => 'xw'
  => 'wqtzjtevsrxvj'
  => 'obcbkqh'
  => 'ghu'
  => 'o'
Test 3: Empty strings...
[3]

(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!