0-basics

September 4, 2023

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 [2]: ###
        ### YOUR CODE HERE
        x_float = 1.0
        type(x_float)
        ###
```

```
Out[2]: float
```

```
In [3]: # `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 [4]: def strcat_ba(a, b):
            assert type(a) is str, f"Input argument `a` has `type(a)` is {type(a)} rather than
            assert type(b) is str, f"Input argument `b` has `type(b)` is {type(b)} rather than
        ###
        ### YOUR CODE HERE; don't forget to indent items within the function!!!
            return(b + a)
        ###
```

```
In [5]: # `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 {len
        assert c[:len(b)] == b
        assert c[-len(a):] == a
        print("\n(Passed!)")
```

```
strcat_ba("ccdnw", "kxs") == "kxsccdnw"

(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 [6]: def strcat_list(L):
            assert type(L) is list
            ###
```

```
            ### YOUR CODE HERE
            for x in L[::-1]:
                print(x)
            return "".join(L[::-1])
            ###
```

```
In [7]: # `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!)")
```

```
eci
dwr
oxi
fck
elr
mba
L == ['mba', 'elr', 'fck', 'oxi', 'dwr', 'eci']
strcat_list(L) == 'ecidwroxifckelrmba'

(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 [8]: 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
            return int(a//b)
            ###
```

```
In [9]: # `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(0.21822808437275154, 0.9359869244437382) == floor(0.2331529198470861) == 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 [10]: def ceiling_fraction(a, b):
             assert is_number(a) and a >= 0
             assert is_number(b) and b > 0
             ###
             ### YOUR CODE HERE
             return int(a//b) + 1
             ###
```

```
In [11]: # `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(0.22764511896777773, 0.4696112849815176) == ceiling(0.4847522328530438) == 1

(Passed!)
```

```
In [12]: 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 [13]: def report_exam_avg(a, b, c):
             #assert is_number(a) and is_number(b) and is_number(c)
             ###
             ### YOUR CODE HERE
             d = round((a + b + c)/3,1)
             print('Avg Score is = ', d)
             return ('Your average score: {}'.format(d))
             ###
```

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

```
Avg Score is =  91.7
Your average score: 91.7
Checking some additional randomly generated cases:
Avg Score is =  69.8
33.56093598356962, 87.26685556368335, 88.43335829620675 -> 'Your average score: 69.8' [0.046283
Avg Score is =  49.2
6.447337487822158, 86.16476223199047, 54.8630467369436 -> 'Your average score: 49.2' [0.041617
Avg Score is =  67.3
34.45995646635937, 90.97389472600506, 76.57547440335483 -> 'Your average score: 67.3' [0.03644
Avg Score is =  57.9
55.42257933410346, 90.63284077321424, 27.760932992520594 -> 'Your average score: 57.9' [0.0387
```

```
Avg Score is =  33.2
1.3484202545615998, 59.5670740662527, 38.81772999083489 -> 'Your average score: 33.2' [0.044408
Avg Score is =  25.7
6.55300308614668, 62.29927011910827, 8.25305743671666 -> 'Your average score: 25.7' [0.0017768
Avg Score is =  80.8
96.4157841644683, 97.43471287205601, 48.66248337339606 -> 'Your average score: 80.8' [0.037660
Avg Score is =  83.7
99.40713441490156, 70.72172250866183, 81.0992854018537 -> 'Your average score: 83.7' [0.042714
Avg Score is =  37.3
30.925873508733403, 16.48193199650524, 64.45319692061084 -> 'Your average score: 37.3' [0.0129
Avg Score is =  52.2
48.9746370168478, 72.53868950570512, 35.10403181064905 -> 'Your average score: 52.2' [0.005786

(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 [15]: def count_word_lengths(s):
             assert all([x.isalpha() or x == ' ' for x in s])
             assert type(s) is str
             ###
             ### YOUR CODE HERE
             print("".join(s))
             new_strn = s.split(' ')
             print('New strn = ', new_strn)
             list_lengths = [len(x) for x in new_strn if x != '']
             if len(list_lengths) == 1:
                 return []
             else:
                 return list_lengths
             ###

In [16]: # `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!)")
```

```
the quick brown fox jumped over the lazy dog
New strn =  ['the', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy', 'dog']
Test 1: count_word_lengths('the quick brown fox jumped over the lazy dog') == [3, 5, 5, 3, 6, 4
qzqjkambptulskyhn fdiiwemkllyhimypiljlzh
New strn =  ['qzqjkambptulskyhn', 'fdiiwemkllyhimypiljlzh']
Test 2: count_word_lengths('qzqjkambptulskyhn fdiiwemkllyhimypiljlzh') == '[17, 22]'
  => 'qzqjkambptulskyhn'
  => 'fdiiwemkllyhimypiljlzh'
Test 3: Empty strings...

New strn =  ['']

New strn =  ['', '', '', '']
```

```
the
New strn =  ['the']
[]

(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!