CSC110 Fall 2022 Assignment 1: Written Questions

Yehyun Lee

September 26, 2022

Part 1: Data and Comprehensions

1. **Imagine this scenario...**

   (a) (The total amount of money you're planning to spend on your trip.)

   "**Float**" will be best to represent the total amount of money. When it comes to financing that deals with money, it requires a high degree of precision of decimals because it is very sensitive. So you need a float that supports decimals rather than an int.

   (b) (The restaurant names in your sister's "top ten restaurants" message, in the order of her preferences.)

   "**List**" is the best one to use in this case. Its elements' data type will consist of "**string**", and elements in order of my sister's preferences.

   Some might wonder why not use other data types like set or dict, but a problem with the set is, that it has no order, thus, we cannot use it. "Dict" works, you can rank the top ten restaurants as int for key and use the name of the restaurant as string for associated value; however, it is more efficient to use the list as we do not need to pair ranks and names of a restaurant, the list does itself, we just need the name of the restaurant as a string to be in order. There is a problem with the list allowing duplicates, but we can simply avoid it by not writing the same name of the restaurant twice or more.

   Now, the question is why use string for elements of a list? "**String**" must be used to represent the restaurant name. Since the message, "top ten restaurants" is a message form of data, we cannot represent this with int nor float, but with string, we can represent a message.

   (c) (The number of places you are staying that have laundry service.)

   "**Int**" is most suited to the situation. Well, some people might say float can be another option, however, a number of places are not decimals, but it is a whole number, we do not need a high precision data type like float. Thus, "int" is most suited.

   (d) (The names of the cities you will be visiting and the corresponding number of days you are staying in each city.)

   For this collection, we need to use "**dict**". The key type will be "**string**" and the associated values' type will be "**int**". In this way, we can pair up the names of cities and the number of days staying in each city, and avoid possible duplicates of names of cities, as dict does not allow duplicates of keys.

As names of cities are names, not numbers, we need string to represent the names. And since the number of days staying in each city is a whole number, we should use int to represent whole number rather than float that has decimals.

(e) (Whether or not you have a valid passport.)

"**Bool**" is most suited. Bool's result is either True or False, and the question can only be answered either if you do or do not have a valid passport. So, to represent only two possible outcomes, bool is the best way to represent this.

2. **Exploring comprehensions.**

(a) i. The expression output: ['B', 'l', 'u', 'e', 'b', 'e', 'r', 'r', 'y']

ii. Value's data type is a "**list**", consists elements of "**strings**".

(b) i. The expression output: {'u', 'e', 'r', 'y', 'B', 'l', 'b'}

ii. Value's data type is a "**set**", consists elements of "**strings**".

iii. Expression in (b) is using the data type of set, thus, the order does not matter whereas a list in (a)'s order matters. Due to their different characteristics, they both have different orders. In addition, the list allows having duplicate elements, however, in the set, there are no duplicates. Therefore, in terms of the size calculated by len() in Python, (b) has 7 elements, whereas (a) has 9 elements. And one last thing to note is, (a) has all the characters that (b) has, and vice versa. In conclusion, expression in (a) and (b) are different in terms of order, number of elements, and whether they have duplicates or not.

(c) **Expression 1**

[name + '!' for name in names] output: ['David!', 'Tom!', 'Mario!']

Expression 2

[name for name in names] + ['!' for name in names]" output: ['David', 'Tom', 'Mario', '!', '!', '!']

Why are their values different?

Note: [<expression> for <variable> in <collection>]

Although expression 1 and 2 seems to be similar, they both operate differently.

What the first expression does is that it simply adds the symbol, "!" after each element of: "names = ['David', 'Tom', 'Mario']". Thus, it output: ['David!', 'Tom!', 'Mario!'].

In expression 2, "[name for name in names]" only contains the variable "**name**" in side <expression>, thus, it creates a list of "['David', 'Tom', 'Mario']". However, "['!' for name in names]" does not contain the variable variable "**name**" inside <expression>, but only "!", thus, it creates a list of, "['!', '!', '!']". When you add both together, you get one list of: ['David', 'Tom', 'Mario', '!', '!', '!'].

To explain this more thoroughly, expression 1 is just one comprehension that contains the variable, "**name**" that is added with string "**!**". Whereas, expression 2 is two list comprehensions added into one list, where it's coded differently to make different output compare to expression 1.

Thus, expression 1 is one compression and expression 2 is two different comprehensions that's coded to have different output.

Part 2: Programming Exercises

Complete this part in the provided `a1_part2.py` starter file. Do **not** include your solution in this LaTeX file.

Part 3: Pytest Debugging Exercise

1. According to pytext report, "**test_single_bill**" has passed, but "**test_two_customers**" and "**test_just_food**" has failed.

2. **Note**:

   FAILED a1_part3.py::test_two_customers - KeyError: 'songs'

   FAILED a1_part3.py::test_just_food - assert 106.25 == 25.71

   What error is causing "test_two_customers" to fail?

   "test_two_customers" has failed due to KeyError. For the variable "bills", where it is equal to "['Food': 15.0, 'Songs': 3, 'Food': 16.2, 'Songs': 2]", it is used to call function "get_largest_bill". When the function "get_largest_bill" tries to call "calculate_total_cost", where it uses the variable "bills" as an argument, it is looking for key "songs" and "food". However, inside the list variable "bills", inside dict, it has key "Food" and "Songs", which, they're different from key that argument is looking for, due to capitalization at the start and the different order. Since the function couldn't find the key it was looking for, it raised a KeyError error.

   Solution

   To fix this, we should change bills' key from "Food" and "Songs" to "food" and "songs", and in "get_largest_bill", change the incorrect return code by changing the order of argument for calling "calculate_total_cost". For example, change "calculate_total_cost(bill['songs'], bill['food']" to "calculate_total_cost(bill['food'], bill['songs']".

   What error is causing "test_just_food" to fail?

   "test_just_food" is caused by AssertionError. Although this one's variable "bills" have correct capitalization, since the argument when calling "calculate_total_cost" in the return code for "get_largest_bill" is in a different order, it is causing miss calculation of "actual" to be different from "expected_value". Since the function couldn't have "actual" to be equal to "expected", it caused an AssertionError.

   Solution

   The same solution applies to this. In the return code of "get_largest_bill", just change the order of the argument for calling "calculate_total_cost". Reference the example I stated in last sentence of solution for "test_two_customers".

3. **Why did "test_single_bill" work with incorrect code?**

   The main reason other tests failed is that keys had different capitalization and different order of argument when calling "calculate_total_cost". Since other tests' variable "bills" had keys paired up with a different value of associated values, when the order of arguments changed, the calculation dealt with a different number it had to calculate. However, since "test_single_bill" has "bills = ['food': 10.0, 'songs': 10]", in which keys have the same identical associated

values, for example, food is 10.0 and songs is 10, when the order changed due to incorrect code, the calculation still remained same.

In conclusion, since associated values had identical numbers, they had identical actual values and expected values even though the order of argument was wrong. Thus, "test_single_bill" was able to pass the test even with the wrong code.

Part 4: Colour Rows

Complete this part in the provided `a1_part4.py` starter file. Do **not** include your solution in this LaTeX file.

Part 5: Working with Image Data

Complete this part in the provided `a1_part5.py` starter file. Do **not** include your solution in this LaTeX file.