CSC110 Fall 2022 Assignment 1: Written Questions

Shivesh Prakash

September 20, 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`: I would use floats to represent my total budget because the value is probably going to contain decimals, especially because of converting Euros to Canadian Dollars. Another idea here could be to use strings to avoid confusion regarding currencies; such as `"$10"` or `"€10"`. But I consider this selection inferior because it limits arithmetic operations such as addition and subtraction, which could be crucial in our program.

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

   `list`: I would use lists to store this data because lists are an unbound ordered collection. A slight inconvenience here would be that I would see the 4th best restaurant when I input `list[3]`, hoping to see the third best one. But this problem can be resolved by a simple self-made function.

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

   `int`: I would use integers to store this value because the number of places having laundry service can not have decimal values. Using integers over floats gives us the advantage of speed in calculations.

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

   `dict`: I would use dictionaries to store these values because dicts allow me store values in pairs. I would use keys to represent the names of cities I plan to visit and the corresponding values to represent the number of days I spend staying in the city.

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

   `bool`: I would use booleans to store this value because only two outcomes are possible- `True` or `False`. Using booleans over strings will keep my program open to more development. Also booleans use less bits so they work faster than strings.

2. **Exploring comprehensions.**

   (a) i. This expression evaluates to the list: `['B', 'l', 'u', 'e', 'b', 'e', 'r', 'r', 'y']`.

   ii. The data type of the expression is `list` and that of its elements is `str`.

(b) i. This expression evaluates to a set containing the letters of the word `'Blueberry'` in random order, but with only single iterations of repeated letters `'e'` and `'r'`. Such as: `{'e', 'b', 'y', 'B', 'l', 'r', 'u'}` or `{'B', 'y', 'u', 'l', 'b', 'r', 'e'}`.

ii. The data type of the expression is `set` and that of its elements is `str`.

iii. This value has 2 lesser elements than the value in part (a), this is because sets do not allow elements to repeat and automatically delete duplicate entries. Another key difference is the randomness of elements in part (b), this is because sets are unordered collections.

(c) Expression 1 evaluates to `['David!', 'Tom!', 'Mario!']`, while Expression 2 evaluates to `['David', 'Tom', 'Mario', '!', '!', '!']`. This is because lists are ordered data types, when they are added or concatenated they simply merge one after another. In expression 1 we defined each element as `name + '!'` so the list contains each name followed by an exclamation mark as the elements. Meanwhile in expression 2 we concatenated a list containing the names with a list containing exclamation marks, thus generating an odd result.

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. `test_single_bill(): Passed`
   `test_two_customers(): Failed`
   `test_just_food(): Failed`

2. `Error 1`- In the definition of function `calculate_total_cost(menu_amount: float, songs: int)` it is stated that the function takes the `menu_amount` as the first parameter and `songs` as the second parameter. The function body then goes on to make the calculations by finding tax and tips values using `menu_amount` and song amount by using parameter `songs`. However, the return statement of function `get_largest_bill()` states
   `"return max([calculate_total_cost(bill['songs'], bill['food']) for bill in bills])"`.
   Here the return statement calls the function `calculate_total_cost()` by providing `bill['songs']` as the first parameter and `bill['food']` as the second parameter, thus reversing the parameters and feeding in wrong values. This is the reason we get a actual value different from expected in `test_just_food()`, thus generation an `AssertionError`.

   `Error 2`- The body of `test_two_customers()` defines bills as `bills = [{'Food': 15.0, 'Songs': 3}, {'Food': 16.2, 'Songs': 2}]`. But the docstring of function `get_largest_bill()` tells us that the keys of bills are supposed to be `'food'` and `'songs'` which we use to extract the values from dictionary bills. Python is a case-sensitive language so it considers `'Food'` or `'Songs'` different from `'food'` or `'songs'`, thus it generates a `KeyError`.

3. `test_single_bill()` passed because `bill['food']` and `bill['songs']` have the same value, thus interchanging `menu_amount` and `songs` in function `calculate_total_cost()` does not mess up the calculations, giving the same actual and expected values.

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.