Solutions DIT009 H24 - Exam 2 - 2025-01-09

Question 1: [15 pts] Tracing and Debugging

- `B: x = 7`
- `B: y = 4`
- `Main: x = 3 y =  4`
- `Main: z = 7`
- `A:  Result: Pear Kiwi`
- `Main: fruits = ['Apple', 'Pear', 'Kiwi', 'Lemon']`
- `C: total = 72.0`
- `Main: counter = 2`
- `B: x = 7`
- `B: y = 11`
- `Main: z = 7`
- `A:  Result: Kiwi`
- `Main: fruits = ['Apple', 'Kiwi', 'Strawberry']`
- `C: total = 18.0`
- `Main: counter = 12`

Question 2: [25 pts] Regular Expressions

Q2.1: [15 pts] There should be at least 3 of each of the regex (except for those that match only 2). **3 pts for each regex.**

\b\w{3}\b -- for -- the -- Ran -- gym -- new -- New -- out -- zen -- key -- the -- new -- Who	**\b[aeiou]{2}\w*\b** -- out -- outdoor **@\w+\b** -- @rkout -- @gym -- @nature **\s\w+\d{3}\b** -- runner_jane2023 -- fit_guru2024 **@\w+** -- @rkout -- @gym -- @nature

Q2.2: [10 pts] +5 in each: +2 for correct regex, +3 for each explanation.

The regex to extract all first sentences is:
^\w+.*?[!.?] (could also be ^.+?[!.?])

The regex begin by using an anchor for the beginning of the string (^), then matches anything (.*) but in a non-greedy way (?), and ends with any of the punctuations (! . or ?).

#- by user:.+$

The regex begin by matching the exact character sequence "- by user:" followed by anything at least once (.+), and then ending at the end of the string anchor ($)

Question 3 [40 pts]: Code Quality and Data Structures

Q3.1 [10 pts] Types

[4 pts] i) Python is dynamically and strongly typed. **Dynamically-typed** means that the language will infer the type of the variable based on the value assigned (or that we don't need to explicitly declare the type of the variable when declaring it). **Strongly-typed means** that Python can only operate on values of the same type, such as strings with strings and numbers with numbers.

[3 pts] One Advantages of Python's type. Some examples are:

- More flexibility when declaring and using variables since the programmer does not need to worry about the values inside the variable.
- Language becomes more readable and easier to learn.

[3 pts] One Disadvantages. Some examples are:

- Can introduce bugs **because of type inconsistencies.**
- Requires more error handling code to, e.g., check the types before operating on them (e.g., when using elements in a list).

Note: Vague justifications such as "easier to read, or more flexible" without proper justification will receive less or no points.

Q3.2 [5 pts]

[2 pts] Explanation: The function should fitler pokemons of two specified types into separate lists, then create a new list with all filtered pokemon names in a specific format (e.g., upper or lowercase).

[3 pts] Printing. 1 pts each:

- Filtered pokemon with Electric: ['Pikachu', 'Zapdos']
- Filtered pokemon with Grass: ['Bulbasaur', 'Vileplume']
- List of formatted pokemon: ['PIKACHU', 'ZAPDOS', 'BULBASAUR', 'VILEPLUME']

Q3.3 [15 pts] Code smells and Refactoring

[8 pts] Identifying code smells [2 pts per code smell] to a max of 8.

- Too many parameters in the function signature
- Unused variables such as format_length, data_type
- Monolith function (bloated, blob) since the function has 30 lines of code.
- Duplicated code for filtering pokemon (lines 1-8 and lines 10 - 16), and formatting pokemon (lines 18 - 23 and lines 25 - 29).

[7 pts] Refactoring needed

- [4 pts] Create two separate methods for filtering and formatting
- [1 pts] Fix variable names and function names.
- [1 pts] Remove unused variables
- [1 pts] Reduce the number of parameters to give only the list and one type.

Q3.4 [10 pts] Code smells and Refactoring

Readability:4 pts

- Organisation of the code, naming of variables and function, following conventions, easy to understand code.

Correctness: 6 pts

- [1 pts] Exception is raised correctly
- [1 pts] Right signature and parameters
- [4 pts] Iteration checking for first letter and adding to the **new dictionary**

```
def filter_per_name(my_pokemon, letter):
   if len(letter) != 1:
       raise ValueError("Letter must be a single character")
   else:
       filtered_pokemon = dict()
       for pokemon in my_pokemon:
           if pokemon[0] == letter:
               filtered_pokemon[pokemon] = my_pokemon[pokemon]
        return filtered_pokemon
```

Question 4 [20 pts]: Recursion and Algorithms

Q4.1 [10 pts] Recursive Find Item

Note: It must use recursion, otherwise the question received zero points.

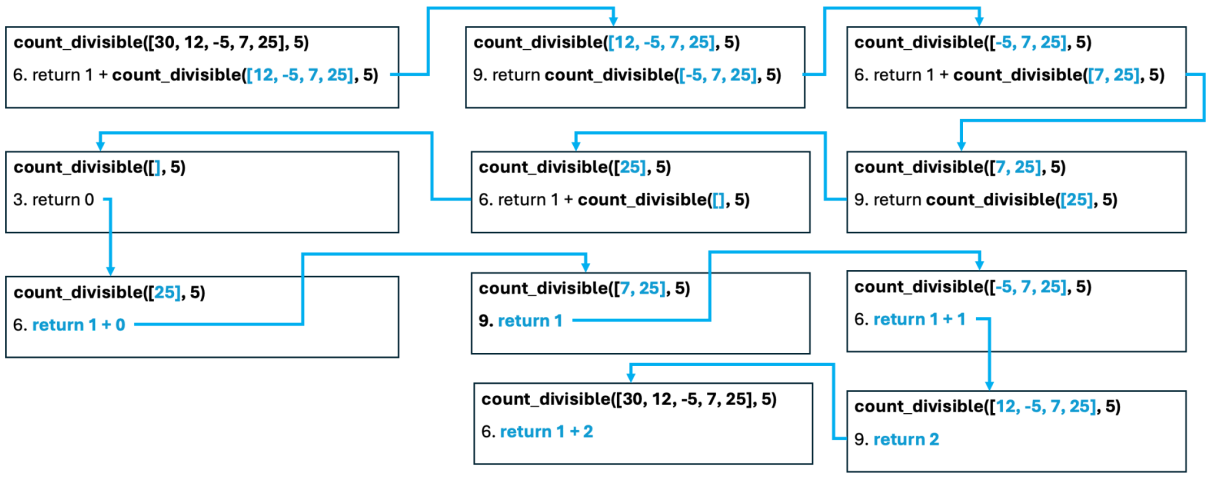
Readability:4 pts
- Organisation of the code, naming of variables and function, following conventions, easy to understand code.

Correctness: 6 pts
- [1 pts] Works for the base case of not found
- [1 pts] Works for the base case of only one element
- [4 pts] Recursive step is done correctly

Examples of penalties:
- [-2 pts] Any input or printing inside function.
- [-2 pts] Incorrect function parameters.
- [-2 pts] Uses Exceptions (they are unnecessary).
- [-3 pts] Missing returns in the function

```
1 def count_divisible(my_list, divisor):
2   if my_list == []:
3       return 0
4
5   elif (my_list[0] % divisor == 0):
6       return 1 + count_divisible(my_list[1:], divisor)
7
8   else:
9       return count_divisible(my_list[1:], divisor)
```

Q4.2 [10 pts] Call Stack [1 pt per correct step]


count_divisible([30, 12, -5, 7, 25], 5)
6. return 1 + count_divisible([12, -5, 7, 25], 5)
count_divisible([12, -5, 7, 25], 5)
9. return count_divisible([-5, 7, 25], 5)
count_divisible([-5, 7, 25], 5)
6. return 1 + count_divisible([7, 25], 5)
count_divisible([], 5)
3. return 0
count_divisible([25], 5)
6. return 1 + count_divisible([], 5)
count_divisible([7, 25], 5)
9. return count_divisible([25], 5)
count_divisible([25], 5)
6. return 1 + 0
count_divisible([7, 25], 5)
9. return 1
count_divisible([-5, 7, 25], 5)
6. return 1 + 1
count_divisible([30, 12, -5, 7, 25], 5)
6. return 1 + 2
count_divisible([12, -5, 7, 25], 5)
9. return 2