Solutions DIT009 H24 - Exam 1 - 2024-10-30

Question 1: [15 pts] Tracing and Debugging

- `Result: Alan Margareth Ada Barbara`
- `B: total = 45`
- `B: counter = 6`
- `A: x = 9`
- `A: y = 13`
- `A: z = 4`
- `Main: names = ['Alan', 'Margareth', 'Ada', 'Grace']`
- `Main: counter = 10`
- `Result: Alan Margareth Ada Grace Peter`
- `Main: names = ['Alan', 'Margareth', 'Ada', 'Grace', 'Lynn']`
- `B: total = 30`
- `B: counter = 5`
- `A: x = 5`
- `A: y = 7`
- `A: z = 3`

Question 2: [15 pts] Regular Expressions

Q2.1: [10 pts] There should be at least 3 of each of the regex. **2 pts for each regex.**

\d{4} - 2023 - 2020 - 2024 - 1987 - 2022 **[_@]\w+?\b** - _joe2023 - _travel_bella - @dev_guru - @code_mike2020 - _jane - _1987 - @game_master - _fanatic	**\b\w+$** - fit_joe2023 - _travel_bella - bookworm123 - hackathon - oldtimer67 - foodie_jane - soundwave_1987 - sports - tech_fanatic - freshface22 **\bf.*?\b** - fitnesslife - fit_joe2023 - food - foodie_jane - fan - freshface22	**[A-Z]\w+\b** - Just - Exploring - Latest - Python - Essentials - Coding - Can - New - Check - Music - Great - What - Loving - Who

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

The regex to extract all sentences with exclamation marks is:
[^.?!]*?\!

The regex begin by matching all characters that are not a punctuation [^.?!], and anything followed by it (*) and avoiding a greedy match (?), and then checking if the match ends with an exclamation point (!).

The regex to extract all hashtags is:
#\w+\b

The regex matches all characters that begin with a hashtag character (#) followed by one or more alphanumeric characters (\w+) and matching those characters until finding a boundary character.

Question 3 [45 pts]: Data Structures and Code Quality

Q3.1 [15 pts] Describing Data Structures

[2 pts] For overall clarity and quality of writing.

[5 pts] i) The main difference between lists and dictionaries is that: **Lists store individual elements in a specific sequence, therefore, lists iterated using an index**. Dictionaries, dictionaries **store a key-value relationship, or a mapping** between a key and a corresponding value**;** moreover, **dictionaries don't have indices**, instead **we use the keys to iterate over its elements**.

[2 pts] Examples of Advantages unique to Lists

- Since they are mutable, we can change the elements in the list using the index of the elements (or, pop() them). Also, you can mention the varied operations.
- We can use the index to directly access elements in specific positions without having to iterate through the list.

[2 pts] Examples of Disadvantages of Lists:

- We should check the types of values inside the list as they can be of different types and introduce inconsistencies in the way we operate on values.
- Can be inefficient if we need to retrieve specific elements without knowing its index, as we might need to go through the entire list.

[2 pts] Examples of Advantages of Dictionaries:

- Allows to store complex relationships, by mapping keys to values. Since each key is unique, we ensure that the relationship is also unique.
- If we know the key, accessing or retrieving elements is very fast.

[2 pts] Examples of Disadvantages of Dictionaries:

- Require more memory than lists as they store more information (e.g., the keys AND values).
- Since they are not ordered, finding specific keys can also be inefficient as we need to iterate over the keys themselves.

Q3.2 [8 pts] Finding bugs

[4 pts] i) The bug, in **line 21**, and the **program is trying to update a tuple**, and the bug happens because Tuples are immutable, i.e., **we cannot update a tuple once we create it**.

[4 pts] ii) The way to fix this is to: change the value of the dictionary from tuples to Lists. Or, prevent the program from changing the tuples.

Note: If someone said that there were more bugs than the one above, then **deduct 2 pts per additional bug** for listing an incorrect bug.

Q3.3 [7 pts] Tracing function calls. [1 pt per correct printing]

```
A0B26 ['Alan', 5000, 'Ana']
SBS02 ['Ada', 200, 'Alex']
21234 ['Grace', 700, 'John']

Value: 5000 SEK. From: Alan. To: Ana
Value: 1000 SEK. From: Alice. To: Maria
Value: 300 SEK. From: Ada. To: Alex
Value: 800 SEK. From: Grace. To: John
Value: 1000 SEK. From: Ada. To: Alex
```

Q3.4 [15 pts] Code Quality and Naming conventions

- +3 pts for overall clarity and cohesion in the explanation of the purpose.
- The explanations should not be close to the code constructs. For instance, describe iterations, or “go over” instead of a “for loop in the dictionary”. Avoid referring to specific variable names, or if-else blocks. It should be okay to refer to Lists, Tuples, Key, etc.

Function a: Easier to explain, focus on iteration.

- **[+3 pts]** Function A **goes through the dictionary** printing the values that have **keys ending with two digits.**

- **[+2 pts]** Should be a verb on printing or showing, and include the aspect of ID or transactions in it. Examples: filter_transactions(), print_id_matches(), show_transactions_per_id(), etc.

Function b: Harder due to if-else. Should explain all cases.

- [+5 pts] Function B **updates the values of the transactions that are below 1000,** by **adding 100 as a bonus. If the bonus would go over the limit of 1000,** then the function rounds the value to the limit (1000).

- [+2 pts]. Should be a verb on updating the transaction value. Examples: update_transactions(), calculate_bonus(), add_bonus(), etc.

Question 4 [20 pts]: Coding and Algorithms

Q4.1 [10 pts] Nested for loops and modulo.

Readability:4 pts

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

Correctness: 6 pts

- [2 pts] Works for the Invalid input
- [4 pts] Passes all other cases.

Examples of penalties:

- [-2 pts] Using recursion
- [-2 pts] Printing, instead of returning a string.
- [-2 pts] Incorrect function parameters.
- [-2 pts] Uses Exception (they are unnecessary).
- [-2 pts] Reads the numbers inside the function.
- [-3 pts] Missing returns in the function

```
def find_divisibles(numbers):
  result = ""
  if len(numbers) < 2:
    result = "Invalid input. We require at least 2 numbers."

  else:
    n = len(numbers)
```

```
    for i in range(n - 1):
      current_number = numbers[i]
      result += f"{current_number} - "
      for j in range(i+1, n):
        if numbers[j] % current_number == 0:
          result += f"{numbers[j]} "
          result += "\n"

  return result
```

Q4.2 [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. OK if index=0.
- [-2 pts] Uses Exceptions (they are unnecessary).
- [-3 pts] Missing returns in the function

```
def find_index(my_list, target, index):
    # Base case: if we've checked all elements, return -1
    if index == len(my_list):
      return -1

    # Check if the current element is the target
    elif my_list[index] == target:
      return index

    else:
        # Recursive case: move to the next index
        return find_index(my_list, target, index + 1)
```