Laboratory exercise 3

Python basics and file I/O

Name: Paola Glavić Sanković **JMBAG:** 0036522290

Preparation

- Review the Python lecture slides and be sure that you are familiar with the basic language features such as variable assignments, if statements, for loops, and list comprehensions.
- Additionally, examine how to use associative arrays (also known as dictionaries, dicts, or hashes). Dictionaries behave similarly to lists, but can use various types for indexing the array (notably, strings, or non-contiguous integers).
  Here is an example of creating an empty dictionary which will contain floats indexed by strings, that is, people's heights indexed by their names, and assigning a height to one person:
  ```
  heights = {}
  heights["John"] = 180.7
  ```
- Examine how to use `argv` from the `sys` module to read command-line arguments
- In all assignments, there is no need to perform error checking and/or handling.
  Assume that the input data is not malformed.
- **Add an appropriate shebang** (`#!/usr/bin/env python3`) at the beginning of your Python scripts, and make them executable using `chmod +x my_script.py`
- Follow the defined script names, the interface (command-line arguments) and the input/output format **exactly**, or your grade will be negatively impacted.
- Make sure you are using **Python 3**. Write clean, readable, easy to understand code. Give meaningful names to variables. Use classes or functions where you see fit. **Code clarity will impact your grade**.

Assignments

Task 1: **Warmup — introduction to standard I/O, reading, and writing files**

Write a Python script called `add_x.py` which will:

a) Take two **command-line** arguments: an input and output filename.
The input file should contain a list of floating point numbers, one per line.

Create such a file using a text editor.

(Note: do not leave an empty line at the end.)

Be sure to `open` the input file in the *read* `'r'` mode, and the output file in the *write* `'w'` mode.

NB: opening a file in the write mode **deletes all of its content**.

b) Ask the user (**on the standard input**) for a floating point number `x`.

c) Read all numbers from the input file.

Add `x` to each number.

Print the results on the screen, AND also write them into the given output file, one per line.

Task 2: **Delivery missions for an office snack robot**

You are programming a robot which will perform food, snack and refreshment order deliveries in an office environment. In order to be able to load the appropriate amount of food on the robot and to plan an optimal delivery route (not in scope of this task), the orders have to be read and processed, and the total amount of each type of food has to be calculated.

Your task is to write a Python script called `process_orders.py`. The script should take in a command-line argument of the input filename, read the orders from the given file, count the number of orders and calculate the total amount of each item type to be delivered.

Let us define the *order size* as the sum of the amounts of all different food items contained therein, and the *hungriest person* as the person with the largest order. The report should include the hungriest person of the day.

The results should be reported both on screen and written into a file called `orders_report.txt`.

The orders have been collected in a text file with the format as follows.

The input file contains a series of orders, where each order consists of the following:

1) Personal name
2) Number of items in the order, `n`
3) ...followed by `n` lines, one line per item in the order, in the following format:

   `Item type: amount (floating point)`

This is an example input file. For testing, create an input file of your own as well.

```
Peter
2
Ham sandwich: 1
Coke: 1.0
Alice
3
Veggie sandwich: 1
Dried apple slices: 1
Sprinkling water: 0.33
Hans
4
Sausage: 2
Mustard: 1
Donut: 1
Radler: 0.5
Bob
2
Ham sandwich: 1
Sprinkling water: 1
Joanna
3
Ham sandwich: 1
Coke: 0.5
Donut: 1
```

This is how the `process_orders.py` script should work when invoked on the file above.

Make sure you print the actual provided input filename, and not always `orders.txt`.

```
$ ./process_orders.py orders.txt
Processing orders.txt
Number of orders: 5
The hungriest person today was Hans!
The largest order size was 4.5.
Item totals:
  Ham sandwich: 3.0
  Coke: 1.5
  Veggie sandwich: 1.0
  Dried apple slices: 1.0
  Sprinkling water: 1.33
  Sausage: 2.0
  Mustard: 1.0
  Donut: 2.0
  Radler: 0.5
```

Advice:

- Use an infinite while loop for processing each order. Read the order name using the `readline` method of the file object. If `readline` returns an empty string which evaluates to `false` (i.e. `if not file.readline()`), the end of the file has been reached, so you can break out of the order processing loop.
- Use a dictionary to store the cumulatives for different item types. The `get` method will be helpful because it can accept a default argument in case the item does not yet exist in the dictionary. You can use a zero default value to easily initialize the sum, for example:
  `amount["Beer"] = amount.get("Beer", 0) + increment`

  Alternatively, you can use a `defaultdict`.
- When processing a line containing an order item, the `split` method of strings can be used to easily separate the item type from the amount.

Exercise submission

Create a zip archive containing `add_x.py`, `process_orders.py`, and your own original input file for Task 2. Upload the archive to Moodle.