Home Assignment 4

Files and Exceptions

General instructions:

- Read questions thoroughly and make sure your programs fulfill the required task.
- The assignment should be solved on your own!
- Follow the submission guidelines written on the website. In particular, all questions ought be submitted in the attached template file ex5_012345678.py, after replacing the digits 012345678 with your 9-digit ID number (Teudat Zehut if you have one, otherwise it's typically a number beginning with 9).
- Submission is due by: see website.
- Task: in this assignment you need to add your code to the given template file. Your input will be given to you in the variables predefined in the template file, currently assigned the invalid value ???. Your code should use these values, make a computation, and provide the required output.
- Self testing: for your convenience, the examples shown below appear as test cases in the template file. Each successful test prints True. Naturally, these examples don't cover all possible cases so you need to test your code on various inputs to verify that the output is correct and that your program never crashes.

- We often use automated testing of assignments, so your outputs must match the required format **exactly** (whitespace and capitalization matter!)
  Follow the output format provided in the examples.

- Do not remove comments from the template file.
- Do not change any variable or function name in the template file.
  Your code may use additional variables and functions, if needed.

- Some tests refer to sample input files attached to the assignment. For these to work, the sample input files need to be in the same folder as the .py file. Alternatively, you could place the input files elsewhere, and write the full path in the test code.

Question 1: summing numbers from a file

Implement the function `sum_num(file_name)`, which gets a string `file_name` specifying the name of an input file. You may assume that the file consists of a single line of text, containing a sequence of positive integers separated by a single space. The function needs to read the file and return the sum of the numbers in it.

Example: the attached file `q1_input_1.txt` consists of the following line.

```
4 55 3 67 10
```

The correct return value here is the integer 139.

In this question you may assume the input is correct, and no error-handling is required.

Question 2: copying lines containing a prescribed substring

Implement the function `copy_lines_with_str(in_file_name, out_file_name, target_str)`, which gets three input strings. The first two specifies file names, and the third is the substring we are interested at.

The function should copy into `out_file_name` all lines of `in_file_name` in which the string `target_str` occurs. The order of lines written to `out_file_name` should match the order these appear in `in_file_name`.

In case of an I/O error during reading from `in_file_name` or writing to `out_file_name`, you need to catch it (using `except`) and gracefully terminate the program without an error. Open resources should be closed, using `finally`. In case of an I/O error you can choose between leaving the output file partially written, empty, or non-existing.

Example 1:

The attached file `q2_input_1.txt` consists of the following lines.

Now somewhere in the Black Mountain Hills of Dakota There lived a young boy named Rocky Raccoon

Running the command `copy_lines_with_str("q2_input_1.txt", "q2_output_1_Rocky.txt", "Rocky")` should generate the output file `q2_output_1_Rocky.txt` which only contains the 2nd line from the input file (since it contains the red `Rocky` substring).

Running the command `copy_lines_with_str("q2_input_1.txt", "q2_output_1_ere.txt", "ere")` should generate the output file `q2_output_1_ere.txt` which contains both lines from the input file (since each of these contains the blue `ere` substring).

Example 2:

The attached file `q2_input_2.txt` is provided with three output files corresponding to the substrings `Rocky`, `boy`, `Nancy`.

Notes:

- You may assume `target_str` is non-empty, and consists of letters, digits, and punctuation characters only.
- If `target_str` does not occur anywhere in the input file, generate an empty output file.
- Text matching is case-sensitive, i.e., lowercase and uppercase differ. For instance, using `"rocky"` instead of `"Rocky"` as `target_str` would result in an empty output file.

Question 3: twin primes

Twin primes are a pair of prime numbers whose difference is 2; e.g., (3,5), (11,13), (59,61), etc. It is conjectured (but still open) that there exist infinitely many pairs of twin primes (for further reading, see: https://en.wikipedia.org/wiki/Twin_prime)

Implement the function `write_twin_primes(num, out_file_name)`, which writes the first `num` pairs of twin primes into the file `out_file_name`. Each pair should appear on a different line, and the two numbers should be separated by a comma (,).

You may (and should) use the implementation seen in class (2nd lecture) for primality testing. Implementing this test in a separate function is recommended (but not mandatory)

Error handling:

- If `num` is zero or negative, raise a `ValueError` with the error message Illegal value num=XXX, where XXX is replaced by the value of `num`.
  You may use the following command (it appears in the template file too):
  `raise ValueError("Illegal value num={}".format(num))`
- If an I/O error occurs while writing to the output file, you need to catch it (using `except`) and then raise a `ValueError` with the error message `Cannot write to XXX`, where XXX is replaced by file name.
  You may use the following command (it appears in the template file too):
  `raise ValueError("Cannot write to {}".format(out_file_name))`

Example:

Running `write_twin_primes(4, "q3_output_1.txt")` should generate the file `q3_output_1.txt` whose contents is:

```
3,5
5,7
11,13
17,19
```

Question 4: analyzing data from a file

In this question you need to analyze song rankings from the *US Billboard Hot 100 Chart*, for three bands: ABBA, The Beatles, Radiohead. Each line in the input file contains a song name, a band name (one of these three), and the highest rank of that song. The three fields in each line are separated by commas (,).

Example for a line of a possible input file:

```
The Winner Takes It All,ABBA,8
```

We would like to calculate the average song rank for each band.

Implement the function calc_avg_position_per_band(in_file_name), which gets an input file with the format described above, and returns a dictionary mapping each band name to the average rank of its songs.

Notes:

- If one of the three bands does not appear in the input file, you should raise a `ValueError` with the error message `At least one of the bands does not appear in the file XXX`, where XXX is replaced by the file name.
  You may use the following command (it appears in the template file too):
  `raise ValueError("At least one of the bands does not appear in the file {}".format(in_file_name))`
- Assume the input is legal; in particular, each line contains exactly two commas, separating the song name (having no commas in it), the band name (having no commas in it), and a positive integer. You may assume no bands appear in the input file except the three mentioned above.
- Store the average rank as an integer. If the average is not a whole number, round it to the nearest integer using Python's `round`.
- You may assume the file name is legal. No need to handle I/O errors while reading.

Example 1:

For the attached input file `q4_input_1.txt`, consisting of the following lines (color added for visual clarity):

```
The Winner Takes It All,ABBA,8
While My Guitar Gently Weeps,The Beatles,63
Fake Plastic Trees,Radiohead,11
Something,The Beatles,1
Here Comes The Sun,The Beatles,4
Dancing Queen,ABBA,1
```

the function should return the dictionary

```
{'The Beatles': 23, 'Radiohead': 11, 'ABBA': 4}
```

Example 2:

For the attached input file `q4_input_2.txt`, identical to the first file but without the Radiohead line, the function should raise a `ValueError`.

Good luck!