Files

Solve the following exercises and upload your solutions to Moodle until the specified due date.

Important Information!

Please try to *exactly match the output* given in the examples (naturally, the input can be different). We are running automated tests to aid in the correction and grading process, and deviations from the specified output lead to a significant organizational overhead, which we cannot handle in the majority of the cases due to the high number of submissions.

Make sure to use the *exact filenames* that are specified for each individual exercise.

Also, use the provided unit tests to check your scripts before submission (see the slides Handing in Assignments on Moodle). Feel free to copy the example text from the assignment sheet, and then change it according to the exercise task to match the output as best as possible.

In this assignment, it is of *particular importance* to wrap the printing that you see in the example outputs in `if __name__=='__main__:'`, as your exercises essentially only consist of a function definition. Example - let's say the task is to write a function that doubles the float value that is passed:

```
def double(var: float) -> float:
    return var*2

if __name__ == '__main__':
    print(double(3.4))
```

Unless explicitly stated otherwise, you can assume correct user input and correct arguments.

You are *not allowed* to use any concepts and modules that have not yet been presented in the lecture. Especially, in this assignment you *not allowed* to use the `csv` and `pandas` packages even though they are briefly mentioned in the Lecture code!

Exercise 1 – Submission: `a7_ex1.py` 20 Points

Write a function `analyze_log_file(filename: str, keyword: str)` that analyzes a log file and outputs information as follows:

- `filename` specifies the path to a log file.
- `keyword` is a string that represents the keyword to filter lines (e.g., "ERROR" or "WARNING").
- The function should:
  - Filter lines in the log file that contain the specified `keyword` (case-insensitive).
  - Write each filtered line to a new file named `"<file_stem>_<keyword>.<ext>"` where `file_stem` and `ext` are the file stem and extension of `filename`.
  - Return the total number of lines containing the `keyword`.
- Use the `utf-8` encoding when reading and writing files.
- If a `FileNotFoundError` is raised because `filename` does not specify a correct path, the function should return `None` and no output file should be created.

Example usage

Example execution of the program:

```
keyword = 'ERROR'
print(analyze_log_file("a7_ex1.log" , keyword))
```

Output: `15`
Expected content of output file in `a7_ex1_ERROR_expected.log`

Example execution of the program:

```
keyword = 'WARNING'
print(analyze_log_file("a7_ex1.log" , keyword))
```

Output: `10`
Expected content of output file in `a7_ex1_log_WARNING_expected.log`

Example execution of the program:

```
keyword = 'WARNING'
print(analyze_log_file("missing.log" , keyword))
```

Output: `None`

Exercise 2 – Submission: `a7_ex2.py` 30 Points

Write a function `organize_directory(src_path: str)` that organizes files in a specified directory (and its subdirectories) and logs the process as follows:

- `src_path` specifies the path to the directory containing files to be organized.
- The function should:
  - Create a new directory named `<src_path>_organized` where files will be moved.
  - For each file in `src_path`, determine its extension (e.g., `.txt`, `.jpg`).
  - Move each file to a subdirectory in `<src_path>_organized` named after the file's extension. For example, all `.txt` files go into `<src_path>_organized/txt`.
  - Log each move in a file called `move.log`, where each line contains the original file path and the new file location, in the format:

    ```
    Copied '<original_path>' to '<new_path>'
    ```

- Use the `utf-8` encoding when writing to the log file.
- If `src_path` does not specify a correct path or is not a directory, an error message should be written to `move.log` in the format:

  ```
  Error: '<src_path>' is not a valid directory.
  ```

- The function should handle nested subdirectories within `src_path`.

Hints

- The method `os.walk()` might be useful.

Example usage

Example execution of the program:

```
organize_directory("a7_ex2_dir")
```

Expected content of output folder in `a7_ex2_dir_expected`
Expected content of `move.log`[1]:

```
Copied 'a7_ex2_dir/document1.txt' to 'a7_ex2_dir_organized/txt/document1.txt'
Copied 'a7_ex2_dir/document2.txt' to 'a7_ex2_dir_organized/txt/document2.txt'
Copied 'a7_ex2_dir/image1.jpg' to 'a7_ex2_dir_organized/jpg/image1.jpg'
Copied 'a7_ex2_dir/image2.jpg' to 'a7_ex2_dir_organized/jpg/image2.jpg'
Copied 'a7_ex2_dir/script1.py' to 'a7_ex2_dir_organized/py/script1.py'
Copied 'a7_ex2_dir/subdir1/image3.jpg' to 'a7_ex2_dir_organized/jpg/image3.jpg'
Copied 'a7_ex2_dir/subdir2/document3.txt' to 'a7_ex2_dir_organized/txt/document3.txt'
Copied 'a7_ex2_dir/subdir2/script2.py' to 'a7_ex2_dir_organized/py/script2.py'
```

[1] These are examples of Unix paths. In Windows the `"/"` are replaced by `"\\"`. This is automatically taken into account in the unit tests.

Example execution of the program:

```
organize_directory("missing_dir")
```

Expected content of `move.log`:

```
Error: 'missing_dir' is not a valid directory.
```

Exercise 3 – Submission: `a7_ex3.py` 20 Points

Write a function `file_statistics(path: str, encoding: str = 'CP1252')` that reads a text file and returns character statistics as follows:

- `path` specifies the path to a text file (the file can be assumed to exist).
- `encoding` specifies the encoding to use when reading the file (default is CP1252).
- The function should:
  - Count the number of Latin alphabet characters, digits, and spaces.
  - Create a list with all other characters in the file.
  - Return a tuple containing these three counts, with the non-space-non-alphanumeric characters provided as a list.

Example usage

Example execution of the program:

```
print(file_statistics('a7_ex3_cp1252.txt'))
print(file_statistics('a7_ex3_utf8.txt',))
print(file_statistics('a7_ex3_utf8.txt', encoding='utf-8'))
```

Expected outputs can be found in `a7_ex3_examples.txt`

Exercise 4 – Submission: `a7_ex4.py` 30 Points

Write a function `analyze_and_append_logs(directory: str, output_file: str)` that analyzes log files in a given directory for occurrences of the keyword "ERROR" and appends the results to a pickle file. The function should:

- Accept the following arguments:
  - `directory`: A string specifying the path to the directory containing log files.
  - `output_file`: A string specifying the path to the pickle file where results will be stored.
- Search recursively in `directory` for all files with a `".log"` extension.
- For each log file, count the occurrences of the keyword "ERROR" (case-sensitive).
- Update the pickle file `output_file` with results as follows:
  - If `output_file` already exists, load the existing data, and append new results.
  - If `output_file` does not exist, initialize a new dictionary.
  - Use the relative path of each log file (from the current working directory) as the key and store the error count as the value.
- Write the updated dictionary back to `output_file` as a pickle file.

Example usage

Example execution of the program:

```
analyze_and_append_logs("a7_ex4_logs", "error_data.pkl")
```

Expected outputs can be found in `a7_ex4_expected_error_data.pkl`[2]

[2]These are examples of Unix paths. In Windows the `"/"` are replaced by `"\\"`. This is automatically taken into account in the unit tests.