Energy profile converter

The goal of this project is to create a **Python** program that can read energy profiles and perform operations on them.

Energy profile

An energy profile is a JSON file describing the generated or consumed power of any system over a year. For example a file could describe how much energy was generated by a solar tracker (like the one in front of the FH JOANNEUM in Kapfenberg) over a year.

Format

An energy profile looks like this:

```
{
    "name": "Solar Tracker",
    "interval_in_minutes": 15,
    "unit": "kWh",
    "data": [0.0,0.0,0.1,...]
}
```

- `"name"`: name of the system
- `"interval_in_minutes"`: interval of data. For example, 15 means there is a data entry for every 15 minutes (35040 for a year).
- `"unit"`: the unit of the data.
- `"data"`: array with the data (`floats`) itself.

For the example above this means: The data is an array with 35040 values. Each value describes the generated energy for the next **15** minutes in **kWh**.

ToDo

Your program should be called from the command line, read the input file, perform changes and write the changes back to an output file.

> You're allowed to use the whole Python standard library, but **NOT** to use any third party modules that you have to install via `pip` etc!

1. Calling the program

Calling your program could look like this:

```
$ python profile_converter.py input_files/example.json output.json --
interval 60 --unit KJ
```

Calling the program like this should do the following:

1. Read the profile from `input_files/example.json`.
2. change the `"interval_in_minutes"` from the current value (15) to 60.
3. modify the **`"data"`**, so the interval matches the 60 minutes.
4. change the `"unit"` to "KJ" (Kilojoules).
5. modify the data, so it is converted from the current unit (kWh) to KJ.
6. write the resulting new profile to `output.json`

Ensure that you catch any wrong inputs and prevent your program from crashing when arguments are missing or wrong.

2. Reading the file

You can use the json library provided in the standard libs (`import json`) to load the file and parse the values.

3. Changing the interval

Your program should be able to convert between the following intervals:

- 1 minute
- 15 minutes
- 30 minutes
- 1 hour
- 1 day

For that read the wanted interval from the command line and compare it with the current interval in the json. If it is different, change the value in the json to the new interval and convert the data.

1. When converting from a more frequent interval to a less frequent interval use the average of the old values for the new one. For example `[0.1,0.5,0.6,0.4,...]` in a 15 minutes interval should give `[0.3,0.5,...]` in a 30 minutes interval.
2. When converting from a less frequent interval to a more frequent interval just repeat the values. For example `[0.3,0.5,...]` in a 30 minutes interval should give `[0.3,0.3,0.5,0.5,...]` in a 15 minutes interval.

3. Changing the unit

Your program should be able to convert between the following units:

- kWh (kilowatt-hours)
- Wh (watt-hours)
- KJ (kilojoule)
- J (joule)

For that read the wanted unit from the command line and compare it with the current unit in the json. If it is different, change the value in the json to the new unit and convert the data. For converting the data just multiply each value in the array with the needed conversion rate.

4. Writing the output

Your program should read the path of the output file from the command line and save the new json with the changed values to the file system.

5. Working with git

You should use `git` to track your progress during development. Therefore create a local git repository and do regular commits. At the end, you should not have only one or two commits! The git repository and it's history is part of the exercise.

Submission

Submit your finished program as a zip containing all your python files, the example input file, an output file that you generated using your program and a README file. The README file should contain a short documentation on how to use the program and a description of how it works. Also the `.git`-directory which is the local representation of your git-repository.

The zip has to follow the naming **comp_ws23_<YOUR-LASTNAME>_<YOUR-FIRSTNAME>.zip**, e.g. **comp_ws23_schwab_harald.zip**.

OPTIONAL: Create a remote git repository on GitHub or git-iit.fh-joanneum.at and add the link to the repository in your README.