"Reading Binary Data / Functional Programming"

Practical Assignment 2

Informatics 1 for Biomedical Engineering

Graz University of Technology

---

1. Tasks

1.1. Functions (1pt)

Your program should contain a proper main function. Further, the tasks 1.4 and 1.5 should be implemented as functions.

1.2. Command Line Parameters (2pt)

Your program should accept a number of command line parameters. Unlike the previous assignment, this time you should use the `argparse`[1] package. Read the documentation to understand how the individual functions of this package work. In particular, you will need `add_argument()` and `parse_args()`.

To extract the variables from the return value of `argparse.parse_args()`, you may use the builtin `vars()` function.

Implement the following parameters

`-i / --in` Path to the input file. Default: ass_1.p

`-o / --out` Path to the output file. Default: ass_2.p

`-d / --data` Path to the directory of the binary data files. Default: DATA/

`--verify` Decider if the verify task should be executed (see Task 1.4)

`--gsr` Decider the GSR calculation should be run. (see Task 1.5)

Naturally, you should verify if the input file is actually present on the system. Else, your program should print a descriptive error message and terminate cleanly (using `exit()`. Use the default values for the parameters should the user not utilize them.

Note: `argpase` automatically adds the `--help` and `-h` parameters. Do not change this. Further, check the value of the `--data` parameters to determine if it already contains a trailing slash (`/`), or if you need to add it manually when opining files.

Example program calls:

```
python assignment_2.py --in DATA/assignment_1.p --out DATA/assignment_2.p --data DATA
python assignment_2.py --in=DATA/assignment_1.p --out=DATA/assignment_2.p --verify
python assignment_2.py -i DATA/assignment_1.p -o DATA/assignment_2.p --gsr
python assignment_2.py -iDATA/assignment_1.p -oDATA/assignment_2.p --verify --gsr
```

[1] https://docs.python.org/3.6/library/argparse.html

1.3. Reading the Signal Data from Binary Files (5pt)

Open the pickle from Assignment 1. Read the binary data files and write their values as lists in your data structure. Create a key called `data` for each signal. For example: `<dict>['drive01']['signals']['ecg']['data'] = []`

The data is encoded in Frames. In the first frame you find the first value for each signal. The order is defined by the header file (in our case alphabetically). Thus, you can use `sorted()` to ensure you read in the correct order. The number of Bits per value is defined by the `data_format`. Here, it will always be 16. (You can assume this for this task!). Use the `struct`[2] package to read the data byte-wise and convert directly into the required 16Bit integers (so-called short integers). The function you will need is `struct.unpack()` and the flag for signed shorts is `'h'`.

Keep an eye on the number of samples per frame (saved in your data structure under texttt'samples_per_frame'). Some sensors have a higher resolution and save multiple values in a single frame. These are always stored one after another. The following example described the order of the signal values in the first frame:

```
32*[ECG] 128*[EMG] 2*[foot gsr] 2*[hand gsr] [hr] [resp]
```

Save this data in your data structure based on frames. Each entry is a tuple with all the values of this frame. **Hint:** `struct.unpack()` can use `'hh'` as format and in this case would return 2 short integers in the correct format. Store the read data into the `data` field of the corresponding signal in your data structure.

Should the binary file be missing, save an empty list in your data structure.

Hint: These files are stored in a binary format. When opening use the flags `'rb'`.

1.4. Signal Verification (4pt)

The header files contain a checksum for each singal. We stored these in the dataset as `checksum`. Add up all the values of a sensor and convert the resulting number to a signed 16 bit integer (ignoring all overflow). Finally, compare this value with the value stored in the `checksum` field. Again, the struct package can be helpful here. (Flag for unsigned short: `'H'`)

Note: Calculate the sum over all signal values. Should there be multiple values per frame, add them up as well.

Should there be an error (as in the values do not match) print the following to the output:

```
CHECKSUM FAILED <record> <signal>

#Example:
CHECKSUM FAILED drive01 ecg
```

Implement this as separate function and execute it only when the optional command line parameter `--verify` is set.

Hint: Modify the read signal data manually to intoduce errors and check your calculation.

[2]https://docs.python.org/3.6/library/struct.html

1.5. Correlation 'hand gsr' und 'foot gsr' (2pt)

Calculate the Pearson Correlation Coefficient[3] for all records containing the signals `'hand gsr'` and `'foot gsr'` .

As Input use the values per Frame. Since both signals contain multiple values per frame, calculate their mean. To calculate the Pearson Correlation Coefficient of two series (lists), you can use the implementation provided by scipy[4](Import: `from scipy.stats import pearsonr`)

Print the following for each record:

```
GSR <record> <correlation>
```

Limit the number of digits after the comma to 4.

Implement this as separate function and execute it only when the optional command line parameter `--gsr` is set.

1.6. Save as Pickle (1pt)

Use `pickle` to save the extended dictionary. The path is defined by the value of the command line parameter `-o` or `--out`.

2. Packages

The following Python packages are allowed or required in this assignment.

- argparse, collections, math, os, pickle, scipy, struct, sys

3. Restrictions

- Do not add any bloat to your submission
- Use built-in functions where possible
- Do NOT load extra packages (no imports)

4. File Headers

All Python source files have to contain a comment header with the following information:

- Author: – Your name
- MatNr: – Your matriculate number
- Description: – Purpose of the file
- Comments: – Any comments as to why if you deviate from the task or restrictions

[3] http://www.statsoft.com/Textbook/Statistics-Glossary/P/button/p#Pearson%20Correlation
[4] https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.pearsonr.html

Please copy and paste the example and change its contents to your data. (Depending on your PDF-viewer you may have to fix the whitespaces)!

Example Header:

```
########################################################################
# Author:      Patrick Kasper
# MatNr:       0730294
# Description: The main file. Assignment only has 1 file...
# Comments:    This is the example comment. I just made it a bit
#              longer so it spans across multiple lines.
########################################################################
```

5. Coding Standard

For this lecture and the practicals we use PEP 8 [5] as a coding standard guideline. Please note that whilst we do not strictly enforce all these rules we will not tolerate messy or unreadable code. Please concentrate in particular on the following aspects:

Language Programming is done in English. This is to ensure someone else at the other end of the world can read your code. Please make sure all sources you submit are thus written in English. This covers both variable names and comments!

Spaces, not tabs. Indent your files with 4 spaces instead of tabs. PEP 8 forces this in Python 3 (whilst allowing more freedom in Python 2). Most editors have an option to indent with 4 spaces when you press the tab button!

Descriptive names. Use descriptive names for variables where possible! Whilst a simple i can be enough for a simple single loop code can become very messy really fast. If a variable has no purpose at all, use a single underscore (_) for its name.

Max 72 characters. Do not have lines longer than 72 characters. Whilst PEP 8 allows 79 in some cases we ask you to use the lower cap of 72 characters per line. Please note that this includes indentation.

6. Automated Tests

Your file will be executed with the following command:

```
>python assignment_2.py <command line params>
```

Please make sure your submission follows all the restrictions defined in this document. Your program will have a limited runtime of 2 minutes. If your submission exceeds this threshold, then it will be considered non-executable. Please note that this is a very generous amount of time for any of the tasks in this course.
If your submission fails any of the automated tests, we will look into your code to ensure the reason was not on our end and to evaluate which parts of your code are correct and awards points accordingly.

[5] https://www.python.org/dev/peps/pep-0008/

7. Submission

7.1. Deadline

05. December 2017 at 23:59:59

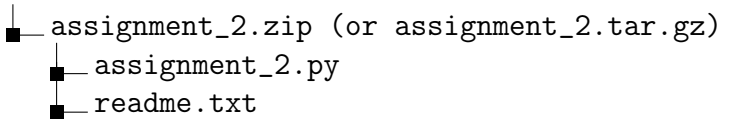
Any submission handed in too late will be ignored with the obvious exception of emergencies. In case the submission system is globally unreachable at the deadline it will be pushed back for 24 hours. If for whatever reason you are unable to submit your work, contact your tutor *PRIOR* to the deadline.

7.2. Uploading your submission

Assignment submissions in this course will always be archives. You are allowed to use .zip or .tar.gz formats.
In addition to your Python source files, please also pack a *readme.txt*. The file is mandatory but its contents are optional. In this file please write down how much time you spent on the assignment and where you ran into issues. Your feedback allows for immediate adjustments to the lecture and tutor sessions.
Upload your work to the Palme website. Before you do please double-check the following aspects:

- File and folder structure (see below)
- Comment header in every source file
- Coding Standard

7.3. Your Submission File



Please do **NOT** submit the dataset or the input pickle. These paths will automatically be linked to your submission during tested!