"Reading Binary Data / Functional Programming"

Practical Assignment 2

Informatik 1 für Biomedical Engineering

Technische Universität Graz

---

1. Tasks

1.1. Funktionen (1pt)

Verwenden Sie eine saubere main Funktion für Ihr Programm. Des Weiteren sind die Tasks 1.4 und 1.5 jeweils als eigene Funktionen zu implementieren.

1.2. Kommandozeilenparameter (2pt)

Ihr Programm soll mit einer Reihe von Kommandozeilenparametern umgehen müssen. Anders als bei Assignment 1 verwenden Sie für dieses Assignment das `argparse`[1] package. Lesen sie, wie die Funkionen von `argparse` funkiontieren. Sie benötigen insbesondere `add_argument()` und `parse_args()`.

Um die Variablen aus dem Rückgabewert von `argparse.parse_args()` zu extrahieren, bietet sich die Funktion `vars()` an.

Folgende Parameter sind zu implementieren:

`-i / --in` Pfad zur Input Datei. Default: ass_1.p

`-o / --out` Pfad zur Output Datei. Default: ass_2.p

`-d / --data` Pfad zur Ordner mit den Binärdateien. Default: DATA

`--verify` Angabe ob die Daten verifiziert werden sollen. (siehe Task 1.4)

`--gsr` Angabe, ob die GSR Korrelation gerechnet werden soll. (siehe Task 1.5)

Selbstverständlich ist zu überprüfen, ob die angegebene Input Datei vorhanden ist. Wenn nicht, dann soll Ihr Programm eine entsprechende Ausgabe machen und sich sauber (mit `exit()`) beenden. Die Default-Werte sind zu verwenden, sollte der Parameter nicht im Aufruf vorhanden sein.

Hinweis: `argparse` Fügt automatisch die `-help` und `-h` Kommandos hinzu. Dies ist einfach so beizubehalten! Überprüfen Sie beim –data Parameter ob er bereits mit einem slash (`/`) endet (also `DATA` oder `DATA/`) , oder ob Sie diesen per Hand hinzufügen müssen.

Beispielaufrufe:

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

```
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
```

1.3. Einlesen der Signale aus den Binärdateien (5pt)

Lesen Sie die Binärdateien ein und schreiben Sie die Werte als Liste in ihre Datenstruktur aus Assignment 1. Legen Sie hierfür einen Key namens `data` an. Zum Beispiel: `<dict>['drive01']['signals']['ecg']['data'] = []`

Die Daten sind in Frames gespeichert. Im ersten Frame finden sich die Werte aller Sensoren. Die Reihenfolge ist definiert durch das header file (und in unserem Fall alphabetisch). Mit `sorted()` können Sie daher sichergehen, dass sie die in der richten Reihenfolge lesen. Die Anzahl der Bits pro Wert ist definiert durch `data_format`. In unserem Fall immer 16. (Sie können dies fest annehmen!) Mithilfe des `struct`[2] packages können Sie einfach byteweise Einlesen und dies direkt in 16Bit Zahlen (so-genannte short integer) konvertieren. Die Benötigte Funktion ist hier `struct.unpack()` und die Flag für unsigned short ist `'h'`.

Beachten sie hierbei aber die Anzahl der Samples pro Frame (In der Datenstruktur gespeichert als `'samples_per_frame'`. Manche Sensoren haben eine höhere Auflösung und speichern daher mehrere Werte in einen Frame. Diese sind immer direkt hintereinander. Folgendes Beispiel beschreibt die Anordnung der Daten im ersten Frame.

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

Speichern Sie die Daten in der Datenstruktur auch pro Frame. Jeweils als Tuple. **Hinweis:** `struct.unpack()` kann zum Beispiel `'hh'` als Format entgegennehmen und returniert dann 2 short integer im richtigen Format. Die Liste mit den einzelnen Frames pro Signal schreiben Sie an die korrekte Stelle in der Datenstruktur.

Sollte die Binärdatei nicht vorhanden sein, dann speichern Sie eine leere Liste in der Datenstruktur.

Hinweis: Die Data Files sind im Binärformat gespeichert. Verwenden Sie beim Öffnen daher die Flags `'rb'`.

1.4. Verifizierung der Signalwerte (4pt)

In den header Files wurde eine Kontrollsumme definiert. In unserem Dataset ist diese als `checksum` gespeichert. Summieren Sie alle Werte des dazugehörigen Sensors auf. Konvertieren Sie das Ergebnis zu einem signed 16 Bit Integer. Vergleichen Sie dies schlussendlich mit dem gespeicherten `checksum` Wert. Das `struct` package kann auch hier hilfreich sein. (Flag für signed short: `'H'`)

Hinweis: Die Summe ist über alle Sensorwerte zu machen. Sollten also in einem Frame mehrere Werte(`samples_per_frame`) sein, summieren Sie auch hier alle auf.

Sollte ein Fehler auftreten (also die Kontrollsumme nicht mit den Werten übereinstimmen) geben Sie folgende Ausgabe aus:

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

```
CHECKSUM FAILED <record> <signal>

#Example:
CHECKSUM FAILED drive01 ecg
```

Implementieren Sie dies als eigene Funktion und führen Sie sie nur aus, wenn der optionale Kommandozeilenparameter `--verify` verwendet wurde.

Hinweis: Modifizieren Sie Ihre eingelesenen Signalwerte um per Hand Fehler zum Testen einzuführen.

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

Berechnen Sie den Pearson Correlation Coefficient[3] für alle Records bei denen die beiden Signale `'hand gsr'` und `'foot gsr'` vorhanden sind.

Als Input für die Korrelation verwenden Sie den Wert pro Frame. Da diese beiden Sensoren aber 2 Werte pro Frame haben, berechnen und verwenden Sie den Durchschnittswert pro Frame. Um den Pearson Correlation Coefficient zweier Serien zu berechnen, können Sie die Implementation von scipy[4] verwenden.(Import: `from scipy.stats import pearsonr`)

Machen Sie pro Record folgende Ausgabe:

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

Limitieren Sie bei der Ausgabe die Nachkommastellen auf 4.

Implementieren Sie dies als eigene Funktion und führen Sie sie nur aus, wenn der optionale Kommandozeilenparameter `--gsr` verwendet wurde.

1.6. Speichern als Pickle (1pt)

Verwenden Sie `pickle` um das erweiterte Dictionary zu speichern. Der Pfad ist definiert durch den Kommandozeilenparameter `-o` oder `--out`.

2. Packages

Folgende Packages sind für dieses Assignment zugelassen und/oder notwendig:

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

3. Beschränkungen

- Fügen Sie keine nutzlosen Komponenten Ihrer Abgabe hinzu
- Importieren Sie keine extra zusätzlichen packages
- Verwenden Sie keine nicht angeführten Kommandozeilenparameter

[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

4. Datei Header

All Ihre Quelldateien in Ihrer Abgabe müssen gleich zu Beginn einen Kommentar mit folgenden Informationen enthalten:

- Author: – Ihr Name
- MatNr: – Ihre Matrikelnummer
- Description: – Generelle Beschreibung der Datei
- Comments: – Kommentare, Erklärungen, usw

Bitte einfach den folgenden Code in Ihre Dateien kopieren und den Inhalt anpassen. Je nach PDF-Reader müssen Sie eventuell die Leerzeichen/Einrückungen per Hand anpassen.

Beispielheader:

```
##############################################################################
# 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

Für diese Lehrveranstaltung orientieren Sie sich am offiziellen PEP 8 Standard[5]. Dieser Beschreibt grundsätzliche Formalitäten im Bezug auf Ihren Code. Folgendes ist besonders zu Beachten:

Sprache. Code schreibt man in Englisch. Im internationalen Zeitalter ist es notwendig, dass auch jemand am anderen Ende der Welt verstehen kann, was Sie programmiert haben. Ihr gesamter Quellcode muss daher auf Englisch geschrieben sein. Dies betrifft sowohl die Kommentare also auch Variablennamen und Ähnliches.

Leerzeichen statt Tabulatoren. Python basiert auf Einrückungen, anstatt auf geschwungenen Klammern . Theoretisch gibt es die Möglichkeit, Leerzeichen (spaces) oder Tabulatoren (tabs) zu verwenden. PEP8 schreibt aber, 4 Leerzeichen als Einrückungen vor. Die meisten Python Programmierumgebungen werden automatisch 4 Leerzeichen einfügen, wenn Sie auf die Tabulator-Taste drücken.

Sprechende Namen. Verwenden Sie kurze, aber sprechende Namen für Ihre Variablen, Funktionen, usw. Es muss eindeutig aus dem Namen hervorgehen, was die Aufgabe des Elements ist. Für simple Iterationen kann ein einfaches `i` ausreichend sein, aber dies kann schnell zu Chaos führen. Sollten Sie Variablen haben, die keine Aufgabe haben und nicht verwendet werden, schreibt PEP 8 vor, einen einfachen Unterstrich (_) zu verwenden.

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

72 **Zeichen Zeilenlänge.** In Ihrem Code darf keine Zeile mehr als 72 Zeichen haben. Dies ist Voraussetzung, um den Code ausdrucken, oder auf allen Geräten darstellen zu können. PEP 8 limitiert in manchen Fällen die Länge auf 79 Zeichen und in anderen auf 72. Bitte halten Sie sich in Ihrem Code an die untere Schranke (72). **Hinweis:** Dies inkludiert die Einrückungen mittels Leerzeichen!

6. Automatisierte Tests

Ihr Programm wird mit dem folgenden Befehl ausgeführt:

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

Vergewissern Sie sich, dass Ihre Abgabe allen Beschränkungen, die in dieser Angabe erwähnt sind, konform ist. Beim Ausführen erhält Ihr Programm 2 Minuten Laufzeit. Sollte dieses Limit überschritten werden, gilt ihr Programm als unausführbar. (Die bereitgestellte Laufzeit ist stets sehr großzügig bemessen!)

7. Abgabe

7.1. Deadline

05. Dezember 2017 um 23:59:59

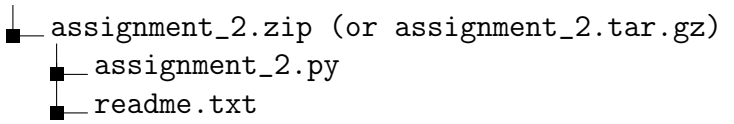
Eine Spätabgabe ist nicht vorgesehen. Ausnahme bilden hier Notfälle. Sollte das Abgabesystem nicht online sein, verlängert sich die Deadline automatisch um 24 Stunden. Sollten Sie, aus diversen Gründen, nicht in der Lage sein, Ihre Abgabe hochzuladen, kontaktieren Sie Ihren Tutor *VOR* der Deadline. (**Hinweis:** Urlaub oder Ähnliches wird nicht als Grund akzeptiert!)

7.2. Hochladen der Abgaben

Assignments werden stets als Archive abgegeben. Erlaubt sind hier die Formate *.zip*, und *.tar.gz*.
Zusätzlich zu ihren Quelldateien, soll Ihre Abgabe auf eine Datei namens `readme.txt` beinhalten. Das Vorhandensein der Datei ist Pflicht, ihr Inhalt aber optional. Sie soll folgenden Inhalt haben: (i) Die Zeit, die Sie benötigt haben, um die Aufgabenstellung zu absolvieren. (ii) Feedback, wo Sie Probleme hatten. Ihr Feedback ermöglicht es, verbreitete Probleme zu erkennen und die Vorlesung und Tutoriumseinheiten entsprechend anzupassen. Abgaben erfolgen auf der Palme Website. Bitte prüfen Sie vor der Abgabe diese Kriterien:

- Datei- und Ordnerstruktur (siehe unten)
- Kommentarheader in jeder Quelldatei
- Coding Standard

7.3. Struktur der Abgabe



Bitte fügen Sie den Datensatz und das input pickle **NICHT** Ihrer Abgabe hinzu. Diese werden automatisch bei den Tests in die entsprechenden Ordner kopiert.