# Praktikumsaufgabe 3

## *Messdaten-Verarbeitung*[1]

**Formalia:**

1. Zur Programmentwicklung **verwenden Sie als Modul-Namen (Datei-Namen) für das dritte Praktikum den Namen** `p03NameVorname.py` **und halten sich strikt an die in der Aufgabe vorgegebenen Namen für die Funktionen (Grund: Unit-Test s.u.)**.
2. Sie können mit dem in Moodle bereitgestellten Unit-Test `p03unit.py` überprüfen, ob Ihre eigene Lösung die richtigen Ergebnisse liefert:
   a) Im Modul `p03unit.py` passen Sie die 1-te Zeile gemäss Ihrem Namen an.
   b) Im Anaconda-Prompt können Sie den Unit-Test ausführen durch die Anweisung: `python p03unit.py`
3. **Erst wenn Ihre Lösung den Unit-Test besteht, können Sie Ihre Lösung in der Zoom-Sitzung (gleicher Link wie Online-Sprechstunde/Vorlesung) abnehmen lassen**.
4. Falls Sie selber Ihre Fehler im Unit-Test nicht beheben können, haben Sie in der Zoom-Sitzung zum Praktikum die Möglichkeit, die Fehler mit den anderen TeilnehmerInnen im Breakout-Raum zu besprechen und sich helfen zu lassen.
5. Falls Sie in der Breakout-Raum-Gruppe nicht weiter kommen rufen Sie Dozenten-Hilfe. Beachten Sie dass aufgrund der Anzahl der Breakout-Räume es etwas dauern kann, bis die *Dozenten-Hilfe* kommt. **Bringen Sie deshalb Geduld und Verständnis mit in die Praktikums-Zoom-Sitzung**.
6. **Die Abnahme Ihrer Praktikums-Lösung kann nur in der für Ihre Gruppe stattfindenden Zoom-Sitzung (gleicher Link wie Online-Sprechstunde/Vorlesung) erfolgen. Die Teilnahme an allen Praktikums-Zoom-Sitzungen für Ihre Gruppe ist Pflicht für den Erhalt der Bonus-Punkte.**
7. Sollten Sie in der vorgesehenen Zoom-Sitzung die Abnahme nicht schaffen haben Sie **am Ende** der nächsten Zoom-Sitzung für Ihre Gruppe die Möglichkeit eine zurückliegende Praktikums-Aufgabe abnehmen zu lassen.
8. Die Gruppeneinteilung und die Termine für die Zoom-Praktikums-Sitzungen werden über den Moodle-Kurs bekannt gegeben.

**Bem.:**
Die Praktikums-Anleitung ist immer als ein Pflichtenheft zu verstehen. Deshalb **müssen alle vorgegebenen Namen und die Anzahl und Art der Parameter von Funktionen im Code genau übernommen werden und dürfen nicht willkürlich geändert werden!**

Lesen Sie die ganze Anleitung **mehrfach genau durch** und **überlegen** Sie sich ggf. auf Papier die algorithmische Lösung bevor Sie mit dem Codieren beginnen!

[1] In Anlehnung an die Praktikums-Aufgaben der Kollegen Schöttl und Tasin aus dem SoSem 2018.

Das Bayerische Landesamt für Umwelt stellt die aktuellen Messdaten, die von den in Bayern aufgestellten Umweltmessstationen ermittelt wurden, zum Herunterladen zur Verfügung.
Auf der Internetseite

`https://www.lfu.bayern.de/luft/immissionsmessungen/messwerte/stationen/detail/803/172`

finden Sie die Messdaten für die Lothstraße der letzten 48 Stunden.
Laden Sie die aktuellen NO2 (Stickstoffdioxid, Dateiname `csv_803_172.csv`) und über den Reiter Feinstaum PM10 (Deiteiname `csv_803_132G.csv`) die Feinstaub-Werte jeweils als **csv-Datei** von der Seite (siehe `"Download"` auf den jeweiligen Seiten immer ganz unten unter *Weiterführende Informationen*) herunter.
Dateien im CSV-Format *(engl. Comma Separated Values)* sind Textdateien, in denen die einzelnen Einträge durch Kommas, die einzelnen Datensätze durch Returns *(new lines)* voneinander getrennt sind. Im deutschsprachigen Raum ist auch ein Semikolon statt eines Kommas als Trennzeichen verbreitet.
Überprüfen und interpretieren Sie den Inhalt der Dateien mit einem Texteditor. Beachten Sie, dass jede Datei **drei Spalten** zur Verfügung stellt. Die ersten beiden Spalten enthalten zeitliche Angaben, die dritte den zugehörigen Messwert.

*Tipp:*
Für eine erfolgreiche Programmentwicklung ist es wichtig, jede Teilfunktionalität zu testen, bevor Sie die nächste Teilaufgabe angehen!
**Damit die betreuenden Dozenten leicht Ihre Ergebnisse überprüfen können verwenden Sie bitte zum Testen die bereitgestellten Dateien `TESTcsv_803_172.csv` ($NO_2$ Werte) und `TESTcsv_803_132G.csv` (Feinstaubwerte).**

1. Erstellen Sie eine Funktion `read_data(fn, val_name="value")`, die die Daten einer Datei mit Dateiname `fn` einliest und in einem geeigneten Tabellen-Objekt (s.u.) abspeichert und dieses zurück gibt.

   Da Dateien im CSV-Format Textdateien sind, können Sie diese Dateien genau so einlesen, wie Sie das Einlesen von Text-Datein im 1-ten Praktikum kennengelernt haben. In Python gibt es auch ein spezielles Modul (`csv`) für die Verarbeitung von Dateien im CSV-Format, dieses soll hier **nicht** verwendet werden.

   Es gibt verschiedene Möglichkeiten der Repräsentation. So könnte man die Daten als Liste von Dictionaries oder als Dictionary von Listen oder Tupel darstellen. Auch eine Liste von Tupel oder ein Tupel von Tupel wäre möglich.

   **Wir entscheiden uns hier für ein Dictionary von Listen.**

   Die Spalte mit den Datums-Werten soll unter den Schlüssel `"date"` als Liste im Dictionary abgespeichert werden.

   Die Spalte mit den Zeit-Werten soll unter dem Schlüssel `"time"` als Liste im Dictionary abgespeichert werden.

   Der Schlüsselname für die Spalte mit den Messwerten kann über den optionalen Parameter `val_name` beim Aufruf von `read_data` angegeben werden. Wenn er beim Aufruf nicht angegeben wird, erhält diese Spalte den Schlüssel `"value"`. Dieses Dictionary wird im Folgenden **Tabelle** (*table)* genannt.

   *Hinweis:*

   Mit `s.split(";")` wird ein String `s` in einzelne Teil-Strings zerlegt, wobei ein Semikolon als Trennzeichen interpretiert wird.

   Können Sie in der Python-Konsole sich die Hilfe für diese Methode anzeigen lassen?

   Warum spricht man in diesem Fall von einer Methode und nicht von einer Funktion?

   Gehen Sie zunächst in einzelnen Schritten vor und überprüfen Sie Ihr Ergebnis immer in der Python-Shell.

*Hinweis für Profis:*

Es gibt eine nicht ganz einfach verständliche einzeilige Lösung (die die Funktion zip und das Entpacken einer Liste mit `*` nutzt). Man beobachte hierfür in der Python-Shell die Wirkungsweise der folgenden Anweisungen:

```
M = [[ 1, 10, 100], [2, 20, 200], [3, 30, 300], [4, 40, 400]]
Mt = list(zip(*M))
```

Warum wird das obige Ergebnis mit `Mt` bezeichnet?

2. Erstellen Sie eine Funktion `stats(table, val_name)`, die die Anzahl, das Minimum, das Maximum und den arithmetischen Mittelwert aller Werte des angegebenen Schlüssels `val_name` der Tabelle `table` als Tupel zurückliefert.

   Sie können davon ausgehen, dass zumindest ein Messwert vorhanden ist und dieser als String, der in eine gültige Fließkommazahl gewandelt werden kann, vorliegt.

   Testen Sie diese Funktion mit den Werten aus den zur Verfügung gestellten Dateien. Geben Sie dabei die Ergebnisse **geeignet formatiert** aus, um auch die **Formatierte Ausgabe** mit der Methode `format()` zu üben.

3. Aus den beiden Tabellen für die Messwerte von $NO_2$ bzw. Feinstaub wird durch ein Zusammenfügen *(Merge)* (s. u. Funktion `merge()`) eine neue Tabelle (Dictionary) erzeugt, die neben den Spalten für das Datum (Schlüssel `"date"`) und die Zeit (Schlüssel `"time"`) auch die Spalten mit den Messwerten für $NO_2$ bzw. Feinstaub enthält.

   Hierfür wird die **Hilfsfunktion**

   ```
   add_entry(table, date, time, val_name0, val_name1, val0, val1)
   ```

   benötigt, die eine Tabelle `table` um einen Eintrag für den durch `date` und `time` angegebenen Zeitpunkt und für die Messwerte `val0` und `val1` mit den Schlüsseln `val_name0` bzw. `val_name1` ergänzt.

   Sie müssen nicht überprüfen ob in `table` bereits Einträge für den angegebenen Zeitpunkt existieren, sondern können die Werte des neuen Eintrages einfach **am Ende der jeweiligen Spalten einfügen**.

   Testen Sie diese Hilfsfunktion, indem Sie Ihr beim ersten Aufruf ein `dict`-Objekt übergeben, das schon die Schlüssel `"date"`, `"time"`, und geeignete Schlüssel-Namen für die Parameter `val_name0` und `val_name1` besitzt und **alle zugehörigen Werte aber leere Listen sind**.

   Anschließend befüllen Sie dieses `dict`-Objekt durch wiederholte (mindestens 2 weitere) Aufrufe von `add_entry()` mit weiteren Einträgen. Geben Sie dann diese Tabelle aus.

4. Die Funktion `check_time(table0, table1, i0, i1)` ist bereits vorbereitet und in der zur Verfügung gestellten Datei `versuch3.py` vorhanden.

   Sie ermittelt, ob der zu Zeile mit Index `i0` von `table0` gehörige Zeitpunkt vor/gleichzeitig/nach dem zu Zeile mit Index `i1` von `table1` gehörigen Zeitpunkt ist. Die Funktion gibt entsprechend die Werte `-1/0/1` zurück.

   Ist die Zeilennummer (Zeilen-Index) in einer Tabelle ungültig (zu groß), wird der Eintrag der anderen Tabelle als zeitlich davor eingestuft.

   Importieren Sie diese Funktion mit einer notwendigen `import`-Anweisung in Ihrem Programm. Anschließend testen Sie die Funktion, indem Sie die beiden heruntergeladenen Tabellen einlesen (nennen Sie die Schlüssel für die Messwerte z.B. `"no2"` bzw. `"fs"`) und dann `check_time` für verschiedene Zeilenkombinationen (verschiedene Werte für die Indices `i0` und `i1`, die auch ausserhalb des gültigen Bereiches liegen können) aufrufen. Die zum Testen gewählten Zeilen-Kombinationen sollten natürlich so gewählt werden, dass man auch alle möglichen Ergebnisse (-1, oder 0 oder 1) erhält.

5. Erstellen Sie eine Funktion `merge(table0, table1)`, die zwei eingelesene Tabellen `table0` und `table1` akzeptiert und in eine gemeinsame neue Tabelle kombiniert und diese zurück gibt.

   Es kann davon ausgegangen werden, dass die Tabellen nach der Messungs-Zeit sortiert sind. In der kombinierten Tabelle sollen Messwerte, die zur selben Zeit gehören, in derselben Zeile erscheinen. Liegt für einen Zeitpunkt ein Messwert nicht vor, soll in die Tabelle stattdessen der Wert `None` vom Typ `NoneType` (und **nicht** `str`) eingetragen werden. Die neue Tabelle soll dabei zeitlich sortiert bleiben.

   *Hinweise:*

   a) Nutzen Sie die Funktionen `check_time()` und `add_entry()`.

   b) Schaffen Sie es, die Schlüssel-Namen für die Messwerte nur aus der jeweiligen Tabelle zu extrahieren so dass sie nicht zusätzlich anzugeben sind?

      Tipp: Hierfür könnten Operationen, die Sie für Objekte vom Typ `set` kennen, nützlich sein.

   c) Der klassische Merge-Algorithmus, z.B. auch angewendet von **Merge-Sort** (ein Vergleichsbasiertes Sortierverfahren, das die asymptotisch optimale worst-case Laufzeit von $O(n \log(n))$ erreicht und somit im worst-case besser als **Quick-Sort** ist) läuft angewendet auf das Zusammenfügen der Tabellen wie folgt ab:

      – Es werden zwei Indizes `i0` und `i1`, mit deren Hilfe man sich die Position des nächsten einzufügenden Eintrages aus Tabelle `table0` bzw. `table1` merkt, verwendet.

      – Solange (`while`-Schleife) in beiden Tabellen noch Einträge vorhanden sind, die noch eingefügt werden müssen, wählt man immer den *Zeit-kleinern* Eintrag und fügt diesen in die neue Tabelle ein und erhöht `i0` oder `i1` entsprechend.
      Sind die Einträge aus den beiden Tabellen zeitgleich, fügt man die Messwerte aus beiden Tabellen zu diesem gleichen Zeitpunkt in die neue Tabelle ein und erhöht sowohl `i0` als auch `i1`.

      – Nach Abbruch der `while` Schleife sind noch die restlichen Elemente der Tabelle, die noch einzufügende Elemente besitzt, einzufügen.

      *Für Profis:*
      Dies kann auf einen Schlag mit einer einzigen Python-Anweisung ohne die Verwendung einer Schleife erfolgen.

   d) Testen Sie diese Funktion indem Sie die zur Verfügung gestellten Dateien
      `TESTcsv_803_172.csv` ($NO_2$ Werte) und
      `TESTcsv_803_132G.csv` (Feinstaubwerte)
      einlesen und daraus eine neue Tabelle erzeugen, die die beide Messwert-Spalten für die $NO_2$ und die Feinstaub Werte enthält.

6. Ändern Sie die Funktion `stats()` so ab, dass sie auch mit der in 5.d) erstellten Tabelle funktioniert (d.h. `None`-Einträge werden einfach ignoriert).

   Geben Sie für diese Tabelle die Ergebnisse wieder **geeignet formatiert** aus.