Hilfe:

- Dash (Mac) oder Zeal (Windows, Linux) nutzen.
- https://goalkicker.com/PythonBook/ von Stackoverflow
  - bei der schriftlichen Matura gibt es auch kein Internet.

Repository: `u02_python` im Ordner `useranlegen`.

A Script zum Anlegen von Linux Usern

A.1 Problembeschreibung

In den beiden Netzwerktechnik-Labors gibt es auf den Linux Rechnern lokale Benutzer für jede Klasse[1].

Die Liste der Klassen gibt es als Excel-Tabelle (*klassenräume*). Wir benötigen jetzt ein Programm zum Anlegen der Benutzer.

Tipps:

- es gibt einen Theorieteil am Ende dieser Angabe
- die restlichen Dateien gibt es in der zip-Datei.

Wichtig – Klarstellung:

- es gibt **zwei** Angabedateien und es sind **zwei Scripts** zu schreiben.
  - Klassenräume: Script mit ein Account je Klasse
  - Namen: Script mit einem Account je Schüler

A.2 Details

- Das Python-Programm `create_class.py` erzeugt mehrere Dateien:
  - ein Bash-Script[2] mit allen notwendigen Schritten/Befehlen zum Erzeugen der Benutzer[3]
  - ein Bash-Script zum Löschen der angelegten Benutzer
  - eine Liste mit Usernamen und Passwörtern zum Verteilen an die unterrichtenden Lehrer
  - ein Logfile mit sinnvollen Angaben
- dabei ist zu beachten
  - der Dateiname für die Eingabedatei soll auf der Kommandozeile angegeben werden (**ohne** vorangestelltes `-f` o.ä.).
  - mit den Optionen `-v` für *verbose* und `-q` für *quite* soll der Umfang des Loggings eingestellt werden können (*Loglevel*).
    - damit sollten sich auch die Bash-Scripts ändern
  - die Benutzernamen sollen
    - Klassen erhalten ein zusätzliches `k` als erstes Zeichen – der Name sollte nicht mit einer Ziffer beginnen
    - sollen in Kleinbuchstaben angegeben werden
    - etwaige Sonderzeichen (Umlaute) sollten ersetzt werden
    - der Kommentar (*gecos field*) sollte auf einen sinnvollen Wert gesetzt werden
  - es gibt zwei zusätzliche Benutzer: `lehrer` und `seminar`

[1] auch wenn diese jetzt weniger genutzt werden
[2] sinnvollerweise wird hier schon das `x` Recht gesetzt
[3] siehe SYT/BS Übung Linux User Mgmt

  - Die Home-Verzeichnisse der Benutzer soll unter `/home/klassen` bzw. `/home/lehrer/` angelegt werden, z.B. `/home/klassen/k1ai`
    * diese Ordner existieren eventuell nicht
  - Die Linux-User müssen in folgenden existierenden System-Gruppen sein: `cdrom,plugdev,sambashare`
  - Das Passwort besteht aus KZRZJZ[4]
    * K(lasse)
    * Z(ufall): ein zufälliges Zeichen aus `!%&(),._-=^#`[5]
    * R(aum), ohne dem `B` für Beamer
    * J(ahrgangsvorstand)
  - Wichtig: eventuell braucht man im Script ein Escape ("\") beim Passwort, aber nicht im Textfile!
  - Man sollte auch
    * den Benutzername ("Gecos")
    * eine sinnvolle Shell (`/bin/bash`)
    * das Passwort (mittels `chpasswd`)

    setzen[6].
  - bei bereits existierenden Benutzern sollte eine Fehlermeldung erfolgen.
  - die Bash-Scripts sollten bei Problemen abbrechen
  - Logging
    * Wir wollen einen `RotatingFileHandler` mit maximal 10.000 Bytes[7] und maximal 5 alte Versionen (*backup*).
    * Zusätzlich ist ein `StreamHandler` (=Ausgabe auf der Konsole) anzulegen.
    * Tipp: je nach Kommandozeilen-Parametern kann man zB. `logging.basicConfig(level=logging.DEBUG)` oder `logger.setLevel(logging.WARNING)` aufrufen.
    * Alternative: `logging.config.fileConfig('logging.conf')`
    * es sollte für jede *Art* von Meldung mindestens einen Aufruf geben
      · Fehler "Datei nicht gefunden" statt Exception
      · Debug: alle Details zur Fehlersuche
  - Datenstruktur
    * man kann die Daten (Klassen, Raum, …) als Tupel oder Liste "herumreichen"
    * oder als `dict`ionary mit sinnvollen Keys
    * oder als `namedtuple`[8]
    * oder als '@dataclass[9]

- jeder Fortschritt der Entwicklung wird im Git-Repository *commited*.
  - es gibt mindestens einen *Commit* je Feature

[4] und in jeder Ausgabedatei sollte das gleiche PW für den User stehen
[5] random.choice() *versteht* Strings
[6] `useradd` ist besser als das für interaktives Anlegen gedachte `adduser`
[7] etwas wenig, aber sonst sieht man das *Rotieren* nicht
[8] https://docs.python.org/3/library/collections.html#collections.namedtuple
[9] https://www.infoworld.com/article/3563878/how-to-use-python-dataclasses.html

B Verbesserung

B.1 Alle Schüler

Noch schöner sind natürlich *echte* Accounts für jeden Schüler. Aus der Schülerliste sollen nun *echte* Benutzer angelegt werden.

Neu:

- **neues**/zweites Script `create_user.py`[10]
- Als Login-Name wird der Familienname verwendet:
  - Nur Kleinbuchstaben.
  - Alle Sonderzeichen (z.B. Umlaute) werden ersetzt (zB aus Ä wird ae, aus ß wird ss).
  - Akzente müssen entfernt werden.
  - Leerzeichen im Namen werden durch Unterstrich ersetzt.
  - Bei mehreren gleichen Namen werden die Zahlen 1,2,3, … angehängt (Beispiel: `maier`, `maier1`, `maier2`, …).
- Das Passwort sollte zufällig gewählt werden.
- Für folgende Teilaufgaben sollten Tests enthalten sein:
  - Sonderzeichen im Benutzernamen
  - Doppelte Benutzernamen

B.2 Excel Output

Die einfache Textliste mit den angelegten Accounts reicht nicht mehr – die Ergebnisse sollen auch in eine Excel-Tabelle.

Ergänze dein Skript – mit einer Option die Auswahl der Art der Ausgabe beim Aufruf.

Tipp: Excel und Zellen mit `=` am Anfang verstehen sich schlecht…

C Python Theorie

C.1 Programmgerüst

Entweder kann man direkt in mehreren Schleifen die User ausgeben.

```
with open()...:
  for ...
    for ...
      user = ...
      print(...user...)
```

Oder mit Generatoren (`yield`) das Lesen und das Ausgeben *entkoppeln* – auch zum Testen hilfreich:

```
def get_user():
  for ...
    for ...
      user = ...
      yield user

...
with open()...
  for user in get_user():
    print(...user...)
```

[10] zusätzliche Option beim ersten Script ist viel zu kompliziert

Das Ergebnis kann man noch immer in eine Liste geben…

Wichtig:

- die *Arbeit* in verschiedene *Phasen*(=Unterprogramme) aufteilen
  - Einlesen (in Liste/Generator oder ähnliches)
  - Aufbereiten/Nachbessern:
    - PW erzeugen[11]
    - bei *echten Usernamen*: Korrektur des Namens (Umlaute), doppelte Namen, etc.
    - Ordnernamen
  - Ausgabe

C.2 Ausgabe

C.2.1 Formatstrings

siehe https://docs.python.org/3/library/string.html#format-string-syntax

Besonders die Variante mit `{name}` ist viel übersichtlicher als komplizierte `+` mit Strings.

Ab Python 3.6 gibt es `f"Wert x={x}"`. Ab Python 3.8 geht auch `f"Wert {x=}"`.

C.2.2 Dateien

immer `with` verwenden – damit erspart man sich das `close`[12].

```
with open("f1.txt", "r") as infile, open("f2.txt", "w") as outfile:
    line1 = infile.readline()
    print("Elegant in eine Datei schreiben.", file=outfile)
    outfile.write("Nicht so elegant")
    ...
```

C.3 Sonderzeichen

C.3.1 Umlaute im Benutzernamen

… sollen in diesem Beisiel nach diesem Schema ersetzt werden:
* ä wird zu ae, etc. * ß wird zu ss

C.3.2 Akzente

… sollen im Benutzernamen verschwidnen.

Eine Möglichkeit, Akzente zu entfernen (und dabei Sonderzeichen wie z.B. ß beizubehalten), ist dieses "Kochrezept" aus dem sehr empfehlenswerten Buch "Fluent Python"[13], Seite 121, `shave_marks()`[14]

- `NFD`[15]: decomposes by expanding composed characters into base characters and separate combining characters.
- `NFC`[16]: composes the code points to produce the shortest equivalent string

[11] ein PW für alle Ausgabedateien
[12] Das `with` statement ruft automatisch die beiden Methoden `__enter()__` und `__exit()__` auf. `__exit()__` schließt die geöffnete Datei.
[13] https://www.oreilly.com/library/view/fluent-python-2nd/9781492056348/
[14] Falls im einem Benutzernamen trotzdem Zeichen vorkommen, die diese Funktion nicht korrekt behandelt, darf das Programm nicht abstürzen sondern einen Logeintrag machen!
[15] **N**ormalization **F**orm **D**ecompose
[16] **N**ormalization **F**orm **C**ompose

```
import unicodedata
import string

def shave_marks(txt):
    """Remove all diacritic marks"""
    norm_txt = unicodedata.normalize('NFD', txt)
    shaved = ''.join(c for c in norm_txt
                     if not unicodedata.combining(c))
    return unicodedata.normalize('NFC', shaved)
```

C.4 Excel-Dateien

C.4.1 Install `openpyxl`

siehe https://openpyxl.readthedocs.io/en/stable/

Installation: Wir verwenden ein *Virtual Environment* dh. wir installieren nur für unser Script und nicht systemweit[17].

- https://docs.python.org/3/tutorial/venv.html
- https://docs.python.org/3/installing/
- https://pymotw.com/3/venv/

Ablauf[18]:

- `python3 -m venv venv` – im Ordner `venv` (Projektunterordner) eine neue Umgebung für dieses Projekt/Beispiel
- `venv/Scripts/activate` oder `source venv/bin/activate`: aktivieren
- in Pycharm: den Pythoninterpreter aus `venv` eintragen
- **Achtung**: der `venv` Ordner kommt **nicht** in das Git-Repository – sondern in `.gitignore` eintragen

  Was in das Repository gehört:

  – eine Liste der benötigten Pakete, kann man leicht erzeugen:

  ```
  pip freeze > requirements.txt
  ```

  – diese kann man später wieder leicht installieren:

  ```
  pip install -r requirements.txt
  ```

- `requirements.txt` kommt in das Repository

C.4.2 Muster

```
from openpyxl import load_workbook

wb = load_workbook(xlsfilename, read_only=True)
ws = wb[wb.sheetnames[0]]
for row in ws.iter_rows(min_row=7):
    x = row[3]     # Element in der Zeile lesen -- Objekt
    y = x.value    # Wert in der Zelle
    z = x.coordinate    # "Name" der Zelle
    ws['A1'] = 42 # Schreiben mit Zellenname
```

Tipp:

- man kann `python` interaktiv aufrufen und experimentieren

[17] xkcd zum Thema: https://xkcd.com/1987/
[18] bitte einmal *manuell* machen + Restart PyCharm; später auch direkt mit PyCharm

- man kann den Debugger benutzen und die Variablen *analysieren*.

C.5 Logging

siehe *"Python logging Cookbook"* (https://docs.python.org/3/howto/logging-cookbook.html) und und http://www.webcodegeeks.com/python/python-logging-example/ und https://pymotw.com/3/logging/.

C.5.1 Log Levels

Log levels are the levels of severity of each log line. Is a good solution to differentiate errors from common information in our log files.

The logging module provides 5 levels of severity: DEBUG, INFO, WARNING, ERROR, CRITICAL. Python's logging documentations defines each of them as:

- **DEBUG**: Detailed information, typically of interest only when diagnosing problems.
- **INFO**: Confirmation that things are working as expected.
- **WARNING**: An indication that something unexpected happened, or indicative of some problem in the near future (e.g. 'disk space low'). The software is still working as expected.
- **ERROR**: Due to a more serious problem, the software has not been able to perform some function.
- **CRITICAL**: A serious error, indicating that the program itself may be unable to continue running.

Loggers can be configured to output only lines above a threshold, which is configured as the global log level of that logger.

C.5.2 Handlers

As we said, a logger is an object which outputs strings to a handler. The handler receives the record and processes it, which means, in some cases, output to a file, console or even by UDP.

The logging module provides some useful handlers, and of course you can extend them and create your own. A list with handlers provided by Python below:

- **StreamHandler**: instances send messages to streams (file-like objects).
- **FileHandler**: instances send messages to disk files.
- **BaseRotatingHandler**: is the base class for handlers that rotate log files at a certain point. It is not meant to be instantiated directly. Instead, use RotatingFileHandler or TimedRotatingFileHandler.
- **RotatingFileHandler**: instances send messages to disk files, with support for maximum log file sizes and log file rotation.
- **TimedRotatingFileHandler**: instances send messages to disk files, rotating the log file at certain timed intervals.
- **SocketHandler**: instances send messages to TCP/IP sockets. Since 3.4, Unix domain sockets are also supported.
- **DatagramHandler**: instances send messages to UDP sockets. Since 3.4, Unix domain sockets are also supported.
- **SMTPHandler**: instances send messages to a designated email address.
- **SysLogHandler**: instances send messages to a Unix syslog daemon, possibly on a remote machine.
- **NTEventLogHandler**: instances send messages to a Windows NT/2000/XP event log.
- **MemoryHandler**: instances send messages to a buffer in memory, which is flushed whenever specific criteria are met.
- **HTTPHandler**: instances send messages to an HTTP server using either GET or POST semantics.
- **WatchedFileHandler**: instances watch the file they are logging to. If the file changes, it is closed and reopened using the file name. This handler is only useful on Unix-like systems; Windows does not support the underlying mechanism used.
- **QueueHandler**: instances send messages to a queue, such as those implemented in the queue or multiprocessing modules.

- **NullHandler**: instances do nothing with error messages. They are used by library developers who want to use logging, but want to avoid the 'No handlers could be found for logger XXX' message which can be displayed if the library user has not configured logging.

The NullHandler, StreamHandler and FileHandler classes are defined in the core logging package. The other handlers are defined in a sub-module, logging.handlers.