

Übung 1

Table of Contents

1. Lösungsidee	2
1.1. Eingabe mit Argumenten/Konsoleneingabe	2
1.2. Überprüfung, ob File konvertiert werden kann	2
1.3. Erstellung Dateiverzeichnis	2
1.4. Einlesen von Files	2
1.5. Ausgeben von Files	3
1.6. Erstellung des Info-Files	3
1.7. Ausnahmebehandlungen	3
1.8. Spezieller Test-Modus	3
1.9. Konsolenausgaben	3
2. Tests	5
2.1. Testen aller Beispielfiles	5
2.2. File existiert nicht	7
2.3. Eingabe über gesetzte Parameter	7
2.4. Eingabe mit einem Datenbankfile, wo Tabelle nicht existiert	7
2.5. Generierung des Information-Files	8
2.6. Erstellung des richtigen Verzeichnisses	8
3. Verwendete Bibliotheken	9
4. Quellcode	10

1. Lösungsidee

1.1. Eingabe mit Argumenten/Konsoleneingabe

Mittels Eingabeparameter können Filename, CSV-Separator & Tabellename (für SQLite-Tabellename) angegeben werden. Alle Eingaben sind verpflichtet und können entweder über Kommandozeilenparameter oder Konsoleneingabe gesetzt werden. Die Eingabe sieht mit der Konsole folgendermaßen aus: ``main.py <filename> <delimiter>``. Hier könnte die Eingabe verbessert werden, indem die Argumente mittels Name wie z.B. `--file` oder `--delimiter` gesetzt werden, dies wurde aber nicht umgesetzt. Wenn die Eingaben nicht gesetzt werden, müssen diese über Konsoleneingabe gesetzt werden. Zusätzlich wird überprüft, ob das File auch tatsächlich existiert. Dazu wurde das Standard-Paket `os` verwendet.

1.2. Überprüfung, ob File konvertiert werden kann

Grundsätzlich wird lediglich überprüft, ob bei einem SQL-File genau eine Tabelle existiert. Der Name kann mit der `sqlite_master` Tabelle ermittelt werden. Bei JSON und CSV wird beim Einlesen mittels Pandas automatisch auf die Gültigkeit überprüft, hier ist zusätzlich nichts mehr nötig. Leere Felder im CSV werden z.B. automatisch als `null` erkannt.

1.3. Erstellung Dateiverzeichnis

Zu Beginn wird überprüft, ob bereits ein Ordner mit Namen des Files existiert. Wenn dies der Fall ist, wird mittels `shutil.rmtree` der Ordner inklusive der enthaltenen Files gelöscht. Anschließend wird ein neues Verzeichnis erstellt und das File hineinkopiert. Zusätzlich wird auch in das entsprechende Verzeichnis gewechselt.

1.4. Einlesen von Files

Txt-Files und Csv-Files werden hier gleich behandelt, es wird bei beiden versucht diese mit der Methode von pandas `read_csv` einzulesen. Es wird auch ein entsprechendes Trennzeichen gesetzt, je nachdem welches festgelegt wird.

Bei der Datenbank wird versucht, eine Verbindung aufzubauen und anschließend das File mittels `read_sql` einzulesen. Der Tabellename wird dabei explizit angegeben.

Für Json Files wird `read_json` verwendet, mit der Option `orient=records` werden außerdem die Spaltennamen entsprechend gesetzt.

1.5. Ausgeben von Files

Dazu werden alle anderen Dateitypen durchlaufen und je nach Dateityp `to_csv`, `to_sql` oder `to_json` verwendet. Diese werden in ein eigenes Verzeichnis out gespeichert, daher muss vorher `makedirs` aufgerufen werden.

1.6. Erstellung des Info-Files

Mittels `with open as f` wird ein neues File erstellt, welches anschließend mit entsprechenden Informationen über das eingelesene File befüllt wird.

Folgende Informationen werden gesetzt:

- Eingabefiletyp
- Zeilenanzahl
- Spaltenanzahl
- Spaltennamen
- Ausgabefiletypen

1.7. Ausnahmebehandlungen

Für falsche Eingaben werden Exceptions geworfen, welche entsprechend abgearbeitet und ausgegeben werden. Daher gibt es in der Main-Methode einen `try except` Block.

1.8. Spezieller Test-Modus

Um das Testen mit den ganzen Beispielfiles zu vereinfachen, wurde in der Main-Methode ein Bool-Flag ergänzt, mit dem automatisch alle Files im Verzeichnis "exercise_test_cases" ausgeführt werden.

1.9. Konsolenausgaben

Um Informationen und Fehler besser anzuzeigen, wurden entsprechend

Informationen mit `print` ausgegeben und mit `Error:` oder `Info:` deklariert.

2. Tests

2.1. Testen aller Beispielfiles

Getestet wurden alle Beispielfiles aus Moodle. Zusätzlich wurde für jedes Dateiformat ein kurzes eigenes File erstellt. Zum Testen aller Files des Verzeichnisses

`exercise_test_cases` muss das Flag `test_files` in der Main-Methode auf `true` gesetzt werden.

Alle Files wurden korrekt erstellt abgesehen von `blog.json`, da hier die Encodierung nicht passen dürfte. Beim File `annual_gold_rate.txt` wird eine Warning ausgegeben, dass die Tabellenspalten mit Leerzeichen erstellt werden. Dies führt aber zu keinen Problemen.

```
----- AnnualTicketSales.csv -----
Info: Creating dir AnnualTicketSales
Info: Copy file AnnualTicketSales.csv
Info: Reading AnnualTicketSales\AnnualTicketSales.csv
Info: Creating out directory
Info: Creating file AnnualTicketSales.txt
Info: Creating file AnnualTicketSales.json
Info: Creating file AnnualTicketSales.db
Info: Creating information.txt
----- annual_gold_rate.txt -----
Info: Creating dir annual_gold_rate
Info: Copy file annual_gold_rate.txt
Info: Reading annual_gold_rate\annual_gold_rate.txt
C:\Users\Andi\Documents\GitHub\fh-se-python\exercise\Part2\Ue01\src\venv\lib\site-
packages\pandas\core\generic.py:2872: UserWarning: The spaces in these column names will not be
changed. In pandas versions < 0.14, spaces were converted to underscores.
  sql.to_sql(
Info: Creating out directory
Info: Creating file annual_gold_rate.csv
Info: Creating file annual_gold_rate.json
Info: Creating file annual_gold_rate.db
Info: Creating information.txt
----- belgium.db -----
Info: Creating dir belgium
Info: Copy file belgium.db
Info: Reading belgium\belgium.db
Info: Creating out directory
Info: Creating file belgium.csv
Info: Creating file belgium.txt
Info: Creating file belgium.json
Info: Creating information.txt
----- blog.json -----
Info: Creating dir blog
Info: Copy file blog.json
Info: Reading blog\blog.json
Error: 'utf-8' codec can't decode byte 0xe4 in position 163: invalid continuation byte
----- canada.db -----
```

```
Info: Creating dir canada
Info: Copy file canada.db
Info: Reading canada\canada.db
Info: Creating out directory
Info: Creating file canada.csv
Info: Creating file canada.txt
Info: Creating file canada.json
Info: Creating information.txt
----- credit_data.txt -----
Info: Creating dir credit_data
Info: Copy file credit_data.txt
Info: Reading credit_data\credit_data.txt
Info: Creating out directory
Info: Creating file credit_data.csv
Info: Creating file credit_data.json
Info: Creating file credit_data.db
Info: Creating information.txt
----- DOGE-INR.csv -----
Info: Creating dir DOGE-INR
Info: Copy file DOGE-INR.csv
Info: Reading DOGE-INR\DOGE-INR.csv
Info: Creating out directory
Info: Creating file DOGE-INR.txt
Info: Creating file DOGE-INR.json
Info: Creating file DOGE-INR.db
Info: Creating information.txt
----- One Piece json.json -----
Info: Creating dir One Piece json
Info: Copy file One Piece json.json
Info: Reading One Piece json\One Piece json.json
Info: Creating out directory
Info: Creating file One Piece json.csv
Info: Creating file One Piece json.txt
Info: Creating file One Piece json.db
Info: Creating information.txt
----- test1.csv -----
Info: Creating dir test1
Info: Copy file test1.csv
Info: Reading test1\test1.csv
Info: Creating out directory
Info: Creating file test1.txt
Info: Creating file test1.json
Info: Creating file test1.db
Info: Creating information.txt
----- test2.db -----
Info: Creating dir test2
Info: Copy file test2.db
Info: Reading test2\test2.db
Info: Creating out directory
Info: Creating file test2.csv
Info: Creating file test2.txt
Info: Creating file test2.json
Info: Creating information.txt
----- test3.txt -----
Info: Creating dir test3
Info: Copy file test3.txt
Info: Reading test3\test3.txt
Info: Creating out directory
Info: Creating file test3.csv
Info: Creating file test3.json
```

```
Info: Creating file test3.db
Info: Creating information.txt
----- test4.json -----
Info: Creating dir test4
Info: Copy file test4.json
Info: Reading test4\test4.json
Info: Creating out directory
Info: Creating file test4.csv
Info: Creating file test4.txt
Info: Creating file test4.db
Info: Creating information.txt
```

2.2. File existiert nicht

Eingabe eines Files, welches nicht existiert.

```
Allowed file types are ('.csv', '.txt', '.json', '.db')
Enter filename: thisfiledoesnotexist.txt
File thisfiledoesnotexist.txt does not exist
```

2.3. Eingabe über gesetzte Parameter

Eingabe mit gesetzten Parametern

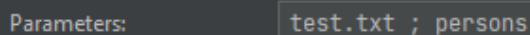


Figure 1. Argumente

```
Info: Creating dir test
Info: Copy file test.txt
Info: Reading test.txt
Info: Creating out dir
Info: Creating file out/test.csv
Info: Creating file out/test.json
Info: Creating file out/test.db
Info: Creating information.txt
```

2.4. Eingabe mit einem Datenbankfile, wo Tabelle nicht existiert

Test mit File, wo Tabelle nicht existiert.

```
Allowed file types are ('.csv', '.txt', '.json', '.db')
Enter filename: test2.db
Enter csv separator: ,
----- test2.db -----
Info: Creating dir test2
Info: Copy file test2.db
Info: Reading test2\test2.db
Error: test2\test2.db has no table
```

2.5. Generierung des Information-Files

Folgende Eingabe führt zu entsprechendem Information-File.

```
Allowed file types are ('.csv', '.txt', '.json', '.db')
Enter filename: test.csv
Enter csv separator: ;
Enter sqlite table name: persons
Info: Creating dir test
Info: Copy file test.csv
Info: Reading test.csv
Info: Creating out dir
Info: Creating file out/test.txt
Info: Creating file out/test.json
Info: Creating file out/test.db
Info: Creating information.txt
```

Listing 1. information.txt

```
Input file type: .db
Total row count: 2
Total column count: 2
Columns: ['Firstname', 'Lastname']
Output file types: ['.txt', '.json', '.db']
```

2.6. Erstellung des richtigen Verzeichnisses

Verzeichnis wird richtig erstellt.

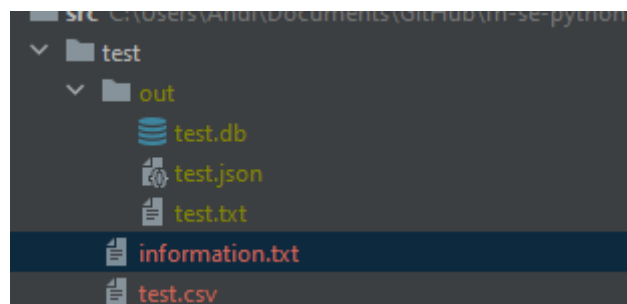


Figure 2. Generated Directory

3. Verwendete Bibliotheken

Listing 2. requirements.txt

```
pandas~=1.3.5
```

4. Quellcode

Listing 3. main.py

```
import os
import shutil
import sqlite3
import sys
from os import listdir
from os.path import exists, isfile, join
from shutil import copyfile
import pandas as pd
from pandas import DataFrame

filetypes = (".csv", ".txt", ".json", ".db")

class FileInput:

    def __init__(self, filename: str, filetype: str, separator: str):
        self.filename = filename
        self.filetype = filetype
        self.separator = separator

def validate_input() -> FileInput:

    arg_count = len(sys.argv)

    if arg_count > 4:
        print("<filename> <csv-delimiter>")
        raise ValueError(f"Invalid arguments {str(sys.argv)}")
    elif arg_count == 1:
        print(f"Allowed file types are {(str(filetypes))}")
        filename = input("Enter filename: ")
    else:
        filename = sys.argv[1]

    if not exists(filename):
        raise IOError(f"File {filename} does not exist")

    filetype = get_filetype(filename)

    if arg_count < 2:
        separator = input("Enter csv separator: ")
    else:
        separator = sys.argv[2]

    return FileInput(filename, filetype, separator)

def get_filetype(name: str) -> str:
    for file_type in filetypes:
        if name.lower().endswith(file_type):
            return file_type

    raise ValueError(f"File {name} has invalid type")
```

```
def parse_sqlite(filename: str) -> DataFrame:
    conn = sqlite3.connect(filename)
    c = conn.cursor()
    c = c.execute(f"SELECT name FROM sqlite_master WHERE type='table';")
    results = c.fetchall()
    if len(results) == 0:
        conn.close()
        raise ValueError(f"{filename} has no table")
    elif len(results) == 2:
        conn.close()
        raise ValueError(f"{filename} has more than one table")

    # Get first column in first row
    table_name = results[0][0]
    conn = sqlite3.connect(filename)
    df = pd.read_sql(f"select * from {table_name}", con=conn)
    conn.close()
    return df

def parse_csv(filename: str, separator: str) -> DataFrame:
    return pd.read_csv(filename, delimiter=separator)

def parse_json(filename) -> DataFrame:
    return pd.read_json(filename, orient="records")

def main(test_files: bool = False):
    if test_files:
        os.chdir("exercise_test_cases")
        files = [FileInput(f, get_filetype(f), ",") for f in listdir() if isfile(f)]
    else:
        files = [validate_input()]

    for file in files:
        print(f"----- {file.filename} -----")
        try:
            filename = file.filename
            filetype = file.filetype
            name = filename.removesuffix(filetype)

            target_folder = name
            if os.path.isdir(target_folder):
                shutil.rmtree(target_folder)

            print(f"Info: Creating dir {target_folder}")
            os.makedirs(f"{target_folder}")
            print(f"Info: Copy file {filename}")
            copyfile(filename, join(target_folder, filename))

            # Check and read Input
            print(f"Info: Reading {join(target_folder, filename)}")
            if filetype == ".csv" or filetype == ".txt":
                df = parse_csv(join(target_folder, filename), separator=file.separator)
            elif filetype == ".db":
                df = parse_sqlite(join(target_folder, filename))
            else:
                df = parse_json(join(target_folder, filename))
```

```
# Write to output
print("Info: Creating out directory")
out_folder = join(target_folder, "out")
os.makedirs(out_folder)
output_filetypes = [x for x in filetypes if x != filetype]
for filetype in output_filetypes:
    out_filename = join(out_folder, f"{name}{filetype}")
    print(f"Info: Creating file {name}{filetype}")
    if filetype == ".csv" or filetype == ".txt":
        df.to_csv(out_filename, file.separator, index=False)
    elif filetype == ".db":
        conn = sqlite3.connect(join(out_folder, f"{name}.db"))
        df.to_sql(name=name, con=conn)
        conn.close()
    else:
        df.to_json(out_filename, orient="records")

print(f"Info: Creating information.txt")
with open(join(target_folder, "information.txt"), "w") as f:
    f.write(f"Input file type: {filetype}\n")
    f.write(f"Total row count: {len(df)}\n")
    f.write(f"Total column count: {len(df.columns)}\n")
    f.write(f"Columns: {list(df.columns)}\n")
    f.write(f"Output file types: {output_filetypes}")
except (IOError, ValueError) as e:
    print(f"Error: {e}")
except BaseException as e:
    print(f"Error: {e}")

if __name__ == "__main__":
    main(test_files=False)
```