

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
2. Tests	4
2.1. Gültiges JSON-File	4
2.2. Gültiges Text-File	4
2.3. Gültiges CSV-File	4
2.4. Gültiges Datenbankfile	5
2.5. File existiert nicht	5
2.6. Eingabe über gesetzte Parameter	6
2.7. Eingabe mit einem Datenbankfile, wo Tabelle nicht existiert	6
2.8. Generierung des Information-Files	6
2.9. Erstellung des richtigen Verzeichnisses	7
3. Verwendete Bibliotheken	7
4. Quellcode	8

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> <tablename>`. Hier könnte die Eingabe verbessert werden, indem die Argumente mittels Name gesetzt werden, dies wurde aber nicht umgesetzt, wäre aber mit entsprechenden Paketen sicher leicht möglich. Wenn die Eingaben nicht gesetzt werden, müssen diese über Konsoleneingabe gesetzt werden. Zusätzlich wird überprüft, ob das File auch tatsächlich existiert. Dazu wurde das `os`-Package verwendet.

1.2. Überprüfung, ob File konvertiert werden kann

Grundsätzlich wird lediglich überprüft, ob bei einem SQL-File die Tabelle auch entsprechend existiert. 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.

2. Tests

2.1. Gültiges JSON-File

Listing 1. test.json

```
[
  {
    "Firstname": "Max",
    "Lastname": "Mustermann"
  },
  {
    "Firstname": "Maria",
    "Lastname": "Musterfrau"
  }
]
```

2.2. Gültiges Text-File

Listing 2. test.txt

```
Firstname;Lastname
Max;Mustermann
Maria;Musterfrau
```

Eingabe

```
Allowed file types are ('.csv', '.txt', '.json', '.db')
Enter filename: test.txt
Enter csv separator: ,
Enter sqlite table name: persons
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
```

2.3. Gültiges CSV-File

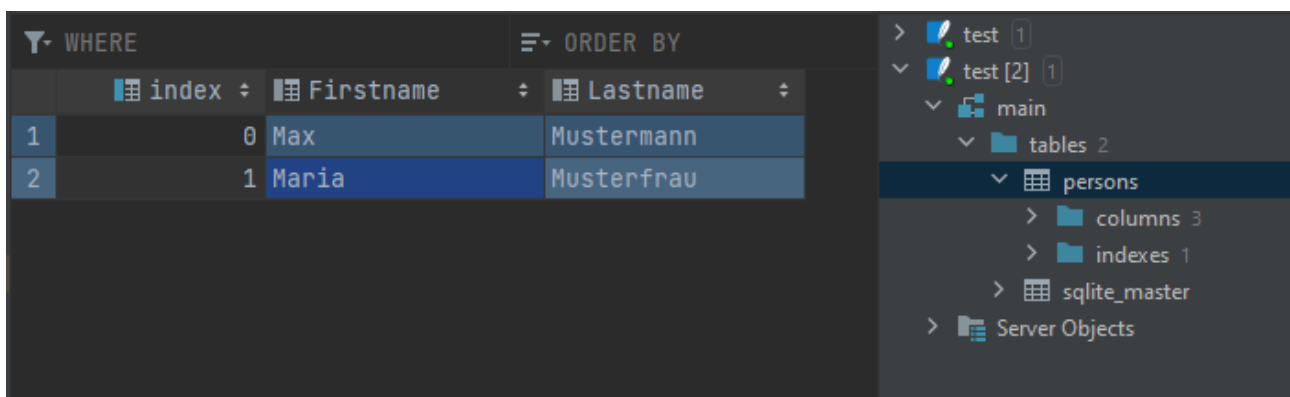
Listing 3. test.csv

```
Firstname;Lastname
Max;Mustermann
Maria;Musterfrau
```

Eingabe:

```
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
```

2.4. Gültiges Datenbankfile



The screenshot shows a database viewer interface. On the left, a table named 'persons' is displayed with two columns: 'Firstname' and 'Lastname'. The table contains two rows of data. On the right, a tree view shows the database structure, including 'test', 'test [2]', 'main', 'tables 2', 'persons', 'columns 3', 'indexes 1', 'sqlite_master', and 'Server Objects'.

	index	Firstname	Lastname
1	0	Max	Mustermann
2	1	Maria	Musterfrau

Figure 1. test.db

Eingabe:

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

2.5. 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
```

2.6. Eingabe über gesetzte Parameter

Eingabe mit gesetzten Parametern



Figure 2. 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
```

2.7. Eingabe mit einem Datenbankfile, wo Tabelle nicht existiert

Tabelle existiert nicht.

```
Allowed file types are ('.csv', '.txt', '.json', '.db')
Enter filename: test.db
Enter csv separator: ;
Enter sqlite table name: doesnotexist
Info: Creating dir test
Info: Copy file test.db
test.db has no table doesnotexist
```

2.8. 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 4. *information.txt*

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

2.9. Erstellung des richtigen Verzeichnisses

Verzeichnis wird richtig erstellt.

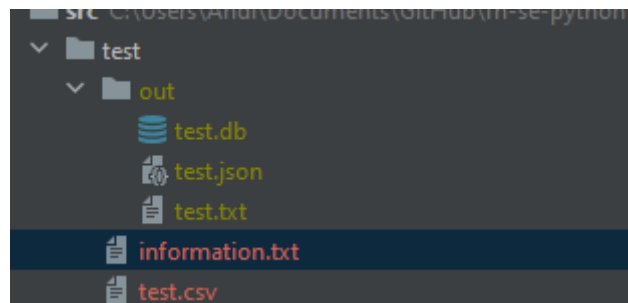


Figure 3. *Generated Directory*

3. Verwendete Bibliotheken

Listing 5. *requirements.txt*

```
pandas~=1.3.5
```

4. Quellcode

Listing 6. main.py

```
import os
import shutil
import sqlite3
import sys
from os.path import exists
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, table_name: str):
        self.filename = filename
        self.filetype = filetype
        self.separator = separator
        self.table_name = table_name

def validate_input() -> FileInput:

    arg_count = len(sys.argv)

    if arg_count > 4:
        print('<filename> <csv-delimiter> <sqlite-tablename>')
        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]

    if arg_count < 3:
        table_name = input("Enter sqlite table name: ")
    else:
        table_name = sys.argv[3]

    return FileInput(filename, filetype, separator, table_name)

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, table_name: str) -> DataFrame:
    conn = sqlite3.connect(filename)
    c = conn.cursor()
    c = c.execute(f"SELECT * FROM sqlite_master WHERE type='table' AND name='{table_name}';")
    if c.fetchone() is None:
        conn.close()
        raise ValueError(f'{filename} has no table {table_name}')

    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():
    try:
        # Get filename
        file_input = validate_input()
        filename = file_input.filename
        filetype = file_input.filetype
        name = filename.removesuffix(filetype)

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

        print(f'Info: Creating dir {name}')
        os.makedirs(name)
        print(f'Info: Copy file {filename}')
        copyfile(filename, f'{name}/{filename}')
        os.chdir(name)

        # Check and read Input
        print(f'Info: Reading {filename}')
        if filetype == '.csv' or filetype == '.txt':
            df = parse_csv(filename, separator=file_input.separator)
        elif filetype == '.db':
            df = parse_sqlite(filename, table_name=file_input.table_name)
        else:
            df = parse_json(filename)

        # Write to output
        print('Info: Creating out dir')
        os.makedirs("out")
        output_filetypes = [x for x in filetypes if x != filetype]
        for filetype in output_filetypes:
            out_filename = f'out/{name}{filetype}'
```

```
print(f'Info: Creating file {out_filename}')
if filetype == '.csv' or filetype == '.txt':
    df.to_csv(out_filename, file_input.separator, index=False)
elif filetype == '.db':
    conn = sqlite3.connect(f'out/{name}.db')
    df.to_sql(name=file_input.table_name, con=conn)
else:
    df.to_json(out_filename, orient='records')

print(f'Info: Creating information.txt')
with open('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()
```