Datenverarbeitung mit Python

Dateien, JSON und Reguläre Ausdrücke

Jowan Sulaiman

GitHub Repository: py-journey

3. Juni 2025

Inhaltsverzeichnis

Was ist File Handling?

- **Definition:** Lesen von Daten aus Dateien und Schreiben von Daten in Dateien.
- **Warum ist das wichtig?**
  - Persistente Speicherung von Daten
  - Datenaustausch zwischen Programmen
  - Verarbeitung großer Datenmengen

Grundlegende Dateioperationen

Die typischen Schritte im Umgang mit Dateien sind:

1. Datei **öffnen**
2. Daten **lesen** oder **schreiben**
3. Datei **schließen**

Weitere Operationen können sein:

- Datei **löschen**
- Dateiinformationen abfragen

Dateitypen im Überblick

Textdateien

- `.txt`, `.csv`, `.json`, `.py`
- Menschenlesbar
- Zeilenweise Struktur
- Wichtig: Zeichenkodierung (z.B. UTF-8)

Binärdateien

- `.jpg`, `.mp3`, `.exe`, `.pdf`
- Nicht direkt lesbar
- Spezifische Programm-Interpretation
- Exakte Strukturkenntnis nötig

Dateien öffnen: open()

Die Funktion `open()` ist der Schlüssel zum Dateizugriff.

Syntax

```
file_object = open("dateiname.txt", "modus")
```

Wichtige Modi:

- `"r"`: **R**ead (Lesen) - Datei muss existieren. (Standard)
- `"w"`: **W**rite (Schreiben) - Erstellt Datei / Überschreibt Inhalt.
- `ä"`: **A**ppend (Anhängen) - Erstellt Datei / Fügt am Ende hinzu.
- `*"`: **E**xclusive creation (Exklusives Erstellen) - Fehler, wenn Datei existiert.
- `"+"` (z.B. `"r+"`): Lesen **und** Schreiben.
- `"b"` (z.B. `"rb"`): **B**inärmodus.

Dateien sicher schließen: Der `with`-Block

Wichtig!

Geöffnete Dateien müssen immer geschlossen werden, um Datenverlust oder Ressourcenprobleme zu vermeiden.

Der `with`-Block (Context Manager) erledigt das automatisch:

```
1 with open("beispiel.txt", "r") as f:
2     inhalt = f.read()
3     print(inhalt)
4 # Datei f ist hier automatisch geschlossen!
```

Vorteile:

- Kein vergessenes `f.close()`
- Sauberer Code
- Automatische Schließung auch bei Fehlern

Lesemethoden im Überblick

Angenommen, `meine_datei.txt` enthält:

```
Zeile 1
Zweite Zeile
Ende
```

1. Gesamten Inhalt lesen: `read()`

```python
with open("meine_datei.txt", "r") as f:
    inhalt = f.read()
    # inhalt ist "Zeile 1\nZweite Zeile\nEnde"
```

Lesemethoden (Forts.)

2. Einzelne Zeile lesen: `readline()`

```
with open("meine_datei.txt", "r") as f:
    zeile1 = f.readline() # "Zeile 1\n"
    zeile2 = f.readline() # "Zweite Zeile\n"
```

3. Alle Zeilen als Liste: `readlines()`

```
with open("meine_datei.txt", "r") as f:
    zeilen_liste = f.readlines()
    # zeilen_liste ist ['Zeile 1\n', 'Zweite Zeile\n', 'Ende']
```

Lesemethoden: Effizientes Iterieren

4. Zeilenweise Iteration (bevorzugt für große Dateien):

```
1 with open("meine_datei.txt", "r") as f:
2     for zeile in f: # Jede Zeile wird einzeln geladen
3         print(zeile.strip()) # .strip() entfernt \n
```

Vorteil

Diese Methode ist speichereffizient, da nicht die gesamte Datei auf einmal in den Speicher geladen wird.

Übungen: Dateien lesen

Übung 1: Erstelle `gedicht.txt`. Schreibe ein Skript, das den gesamten Inhalt liest und ausgibt.

Übung 2: Lies `gedicht.txt` zeilenweise und gib jede Zeile mit Zeilennummer aus.

Übung 3: Datei `daten.txt` (z.B. `Apfel,Rot`): Lies sie und gib nur die Fruchtnamen aus.

Quiz: Dateien lesen – Frage 1

Welche Methode liest den gesamten Inhalt einer Datei als einen einzigen String?

A) `readlines()`
B) `read()`
C) `readline()`
D) `open()`

Antwort: Dateien lesen – Frage 1

Frage 1: Welche Methode liest den gesamten Inhalt einer Datei als einen einzigen String?
Korrekte Antwort: B) `read()`

Quiz: Dateien lesen – Frage 2

Was ist der Vorteil der Verwendung des `with open(...)` Konstrukts?

A) Öffnet im Schreibmodus.
B) Schließt die Datei automatisch.
C) Liest schneller.
D) Konvertiert in JSON.

Antwort: Dateien lesen – Frage 2

Frage 2: Was ist der Vorteil der Verwendung des `with open(...)` Konstrukts? **Korrekte Antwort: B) Es schließt die Datei automatisch.**

Quiz: Dateien lesen – Frage 3

Welcher Modus wird verwendet, um eine Datei nur zum Lesen zu öffnen und einen Fehler auszulösen, wenn sie nicht existiert?

A) `"w"`

B) `ä"`

C) `"r+"`

D) `"r"`

Antwort: Dateien lesen – Frage 3

Frage 3: Welcher Modus wird verwendet, um eine Datei nur zum Lesen zu öffnen und einen Fehler auszulösen, wenn sie nicht existiert? **Korrekte Antwort: D)** `"r"`

Quiz: Dateien lesen – Frage 4

Wie kann man am besten zeilenweise über eine große Datei iterieren, um Speicher zu sparen?

A) `file.readlines()` + Schleife

B) `for line in file:`

C) `file.read().splitlines()`

D) `file.readline()` in `while`

Antwort: Dateien lesen – Frage 4

Frage 4: Wie kann man am besten zeilenweise über eine große Datei iterieren, um Speicher zu sparen? **Korrekte Antwort: B)** `for line in file:`

Schreibmodi im Detail

- `"w"` (Write):
  - Erstellt Datei (wenn nicht existent).
  - **Überschreibt** bestehenden Inhalt!
- ä" (Append):
  - Erstellt Datei (wenn nicht existent).
  - Fügt Daten am **Ende** hinzu.
- *" (Exclusive Creation):
  - Erstellt Datei.
  - Fehler (`FileExistsError`), wenn Datei bereits existiert.

Schreibmethoden

1. Einzelnen String schreiben: `write()`

```
1 with open("ausgabe.txt", "w") as f:
2     f.write("Hallo Welt!\n")
3     f.write("Zweite Zeile.")
```

Inhalt von `ausgabe.txt`:

```
Hallo Welt!
Zweite Zeile.
```

2. Liste von Strings schreiben: `writelines()`

```
1 zeilen = ["Erste Zeile\n", "Zweite Zeile\n"]
2 with open("mehrzeilig.txt", "w") as f:
3     f.writelines(zeilen)
```

Wichtig

`writelines()` fügt **keine** Zeilenumbrüche (`\n`) automatisch hinzu! Sie müssen in den Strings enthalten sein, wenn gewünscht.

Übungen: Dateien schreiben/erstellen

Übung 1: Skript: Erstelle `meine_notizen.txt`. Frage nach 3 Notizen, schreibe jede in neue Zeile.

Übung 2: Skript: Schreibe Einkaufsliste (Liste von Strings) in `einkaufsliste.txt`. Jeder Artikel neue Zeile. Bei Existenz: anhängen.

Übung 3: Skript: Erstelle `log.txt` exklusiv (`"x"`). Schreibe Startnachricht. Bei Existenz: Meldung.

Quiz: Dateien schreiben/erstellen – Frage 1

Welcher Modus überschreibt eine existierende Datei oder erstellt eine neue, wenn sie nicht existiert?

A) `"r"`

B) `ä"`

C) `*"`

D) `"w"`

Antwort: Dateien schreiben/erstellen – Frage 1

Frage 1: Welcher Modus überschreibt eine existierende Datei oder erstellt eine neue, wenn sie nicht existiert? **Korrekte Antwort: D)** `"w"`

Quiz: Dateien schreiben/erstellen – Frage 2

Was bewirkt `file.writelines(["Hallo", "Welt"])`?

A) Schreibt "Hallo\nWelt\n"
B) Schreibt "Hallo Welt"
C) Schreibt "HalloWelt"
D) Fehler (fehlende \n)

Antwort: Dateien schreiben/erstellen – Frage 2

Frage 2: Was bewirkt `file.writelines(["Hallo", "Welt"])`? **Korrekte Antwort: C)**

Schreibt "HalloWelt"

Quiz: Dateien schreiben/erstellen – Frage 3

Sie möchten Daten an das Ende einer bestehenden Datei anhängen, ohne den vorhandenen Inhalt zu löschen. Welchen Modus verwenden Sie?

A) `"w"`

B) `"r+"`

C) `ä"`

D) `*"`

Antwort: Dateien schreiben/erstellen – Frage 3

Frage 3: Sie möchten Daten an das Ende einer bestehenden Datei anhängen, ohne den vorhandenen Inhalt zu löschen. Welchen Modus verwenden Sie? **Korrekte Antwort: C)** ä"

Quiz: Dateien schreiben/erstellen – Frage 4

Welcher Fehler wird ausgelöst, wenn man versucht, eine Datei im Modus `*"` zu erstellen, die bereits existiert?

A) `IOError`

B) `FileExistsError`

C) `FileNotFoundError`

D) `ValueError`

Antwort: Dateien schreiben/erstellen – Frage 4

Frage 4: Welcher Fehler wird ausgelöst, wenn man versucht, eine Datei im Modus `*"` zu erstellen, die bereits existiert? **Korrekte Antwort: B)** `FileExistsError`

Dateien löschen: os.remove()

Das os-Modul wird für Dateioperationen auf Systemebene benötigt.

```
1 import os
```

Datei löschen:

```
1 # Zuerst prüfen, ob die Datei existiert (gute Praxis!)
2 if os.path.exists("zu_loeschen.txt"):
3     os.remove("zu_loeschen.txt")
4     print("Datei gelöscht.")
5 else:
6     print("Datei nicht gefunden.")
```

Mögliche Fehler

`FileNotFoundError`: Datei nicht da.

`PermissionError`: Keine Berechtigung.

`IsADirectoryError`: Versuch, Ordner zu löschen.

Verzeichnisse löschen

Leeres Verzeichnis löschen: `os.rmdir()`

```
1 import os
2 # os.mkdir("leerer_ordner") # Zum Testen
3 if os.path.exists("leerer_ordner") and \
4    not os.listdir("leerer_ordner"): # Prüfen ob leer
5     os.rmdir("leerer_ordner")
6     print("Leerer Ordner gelöscht.")
```

Löst `OSError` aus, wenn der Ordner nicht leer ist. **Verzeichnis (auch nicht leer) löschen:** `shutil.rmtree()`

```
1 import shutil # 'Shell Utilities' Modul
2 import os     # Für os.path.exists und os.makedirs
3
4 # Zum Testen erstellen:
5 # if not os.path.exists("voller_ordner"):
6 #     os.makedirs("voller_ordner/sub", exist_ok=True)
7 #     with open("voller_ordner/datei.txt", "w") as f:
8 #         f.write("Testinhalt")
9
10 if os.path.exists("voller_ordner"):
11     shutil.rmtree("voller_ordner")
12     print("Ordner samt Inhalt gelöscht.")
13 else:
14     print("Ordner 'voller_ordner' nicht gefunden zum Löschen.")
```

Übungen: Dateien löschen

Sicherheitshinweis

Arbeiten Sie bei Löschübungen immer mit Testdateien/-ordnern!

Übung 1: Skript: Erstelle `temp_datei.txt`. Prüfe Existenz, dann lösche sie. Gib Meldungen aus.

Übung 2: Skript: Erstelle leeren Ordner `temp_ordner`. Lösche ihn nur, wenn er leer ist.

Übung 3 (Vorsicht!): Skript: Erstelle `ordner_zum_loeschen` mit einer Datei darin. Nutze `shutil.rmtree()` zum Löschen. Gib eine Warnung aus.

Quiz: Dateien löschen – Frage 1

Welche Funktion wird verwendet, um eine einzelne Datei in Python zu löschen?

A) `os.delete("datei.txt")`

B) `os.rmdir("datei.txt")`

C) `os.remove("datei.txt")`

D) `shutil.remove("datei.txt")`

Antwort: Dateien löschen – Frage 1

Frage 1: Welche Funktion wird verwendet, um eine einzelne Datei in Python zu löschen?
Korrekte Antwort: C) `os.remove("datei.txt")`

Quiz: Dateien löschen – Frage 2

Was passiert, wenn man `os.remove()` auf eine nicht existierende Datei anwendet?

A) Es passiert nichts.

B) Es wird ein `FileExistsError` ausgelöst.

C) Es wird ein `FileNotFoundError` ausgelöst.

D) Die Datei wird erstellt.

Antwort: Dateien löschen – Frage 2

Frage 2: Was passiert, wenn man `os.remove()` auf eine nicht existierende Datei anwendet?
Korrekte Antwort: C) Es wird ein `FileNotFoundError` ausgelöst.

Quiz: Dateien löschen – Frage 3

Welche Funktion kann ein Verzeichnis inklusive seines gesamten Inhalts löschen?

A) `os.rmdir()`

B) `os.removeall()`

C) `shutil.rmtree()`

D) `os.delete_tree()`

Antwort: Dateien löschen – Frage 3

Frage 3: Welche Funktion kann ein Verzeichnis inklusive seines gesamten Inhalts löschen?
Korrekte Antwort: C) `shutil.rmtree()`

Quiz: Dateien löschen – Frage 4

Was ist eine gute Praxis, bevor man versucht, eine Datei zu löschen?

A) Die Datei zuerst umbenennen.

B) Den Inhalt der Datei auslesen.

C) Mit `os.path.exists()` prüfen, ob die Datei existiert.

D) Die Zugriffsrechte der Datei ändern.

Antwort: Dateien löschen – Frage 4

Frage 4: Was ist eine gute Praxis, bevor man versucht, eine Datei zu löschen? **Korrekte Antwort: C) Mit `os.path.exists()` prüfen, ob die Datei existiert.**

Was ist JSON?

Java**S**cript **O**bject **N**otation

- Leichtgewichtiges Datenaustauschformat.
- Für Menschen: Einfach zu lesen und zu schreiben.
- Für Maschinen: Einfach zu parsen und zu generieren.
- Textbasiert und sprachunabhängig.

Typische Anwendungsfälle:

- APIs (Daten zwischen Server und Webanwendung)
- Konfigurationsdateien
- Speicherung strukturierter Daten

JSON Syntax: Bausteine

JSON basiert auf zwei Strukturen:

- **Objekte:** Sammlung von Schlüssel-Wert-Paaren.
  - In `{}` (geschweifte Klammern).
  - Schlüssel: Strings in **doppelten** Anführungszeichen.
  - Beispiel: `{"name: "Max", älter: 30}`
- **Arrays (Listen):** Geordnete Liste von Werten.
  - In `[]` (eckige Klammern).
  - Beispiel: `[Äpfel", "Banane", 100]`

JSON Syntax: Wertetypen

Die erlaubten Werte in JSON sind:

- **String:** In doppelten Anführungszeichen (`"Hallo"`).
- **Zahl:** Ganzzahl oder Fließkommazahl (`123`, `3.14`).
- **Boolean:** `true` oder `false` (kleingeschrieben!).
- **Array:** `[...]`.
- **Objekt:** `{...}`.
- `null`: Repräsentiert einen leeren Wert (wie `None` in Python).

Beispiel JSON-Struktur

```
1 {
2   "id": 1, "name": "Anna", "active": true,
3   "courses": ["Math", "Physics"], "info": null
4 }
```

Python und JSON: Das `json`-Modul

Python bietet das `json`-Modul für die Arbeit mit JSON-Daten.

```
1 import json
```

Wichtige Funktionen:

- `json.dumps(py_obj)`: Python-Objekt → JSON-String (Serialisieren).
- `json.loads(json_str)`: JSON-String → Python-Objekt (Deserialisieren).
- `json.dump(py_obj, file_obj)`: Python-Objekt → JSON-Datei.
- `json.load(file_obj)`: JSON-Datei → Python-Objekt.

Der Parameter `indent` bei `dumps` und `dump` sorgt für eine schön formatierte ("pretty-printed") Ausgabe.

Beispiel: Python ↔ JSON String

Python-Dict zu JSON-String (dumps):

```python
import json
py_data = {"name": "Kurs", "teilnehmer": 15, "aktiv": True}

json_str = json.dumps(py_data, indent=4) # Mit Einrückung
print(json_str)
```

Ausgabe:

```
{
    "name": "Kurs",
    "teilnehmer": 15,
    "aktiv": true
}
```

JSON-String zu Python-Dict (loads):

```python
json_input = '{"produkt": "Laptop", "preis": 1200.50}'
py_obj = json.loads(json_input)
print(py_obj["produkt"]) # Ausgabe: Laptop
```

Beispiel: Python ↔ JSON Datei

Python-Daten in JSON-Datei schreiben (`dump`):

```python
import json
user_data = {"id": 7, "username": "coder"}

with open("user_config.json", "w") as f:
    json.dump(user_data, f, indent=2)
# Datei user_config.json wird erstellt/überschrieben
```

JSON-Datei in Python-Daten laden (`load`):

```python
import json
with open("user_config.json", "r") as f:
    loaded_config = json.load(f)
    print(loaded_config["username"]) # Ausgabe: coder
```

Typen-Mapping: Python ↔ JSON

Python

- `dict`
- `list`, `tuple`
- `str`
- `int`, `float`
- `True`
- `False`
- `None`

JSON

- `object`
- `array`
- `string`
- `number`
- `true`
- `false`
- `null`

Übungen: JSON

Übung 1: Python-Dict (Buch: Titel, Autor, Jahr, ISBN) → JSON-String (Einrückung: 2). Gib aus.

Übung 2: Gegebener JSON-String (Mitarbeiter-Daten) → Python-Dict. Gib Name und 2. Projekt aus.

Übung 3: Liste von Dicts (Personen: Name, Stadt) → `personen.json`. Dann Datei lesen und Namen der 1. Person ausgeben.

Quiz: JSON – Frage 1

Welche Python-Funktion wird verwendet, um ein Python-Dictionary in einen JSON-String zu konvertieren?

A) `json.load()`

B) `json.read()`

C) `json.dumps()`

D) `json.convert()`

Antwort: JSON – Frage 1

Frage 1: Welche Python-Funktion wird verwendet, um ein Python-Dictionary in einen JSON-String zu konvertieren? **Korrekte Antwort: C)** `json.dumps()`

Quiz: JSON – Frage 2

Was ist das JSON-Äquivalent zum Python-Wert `None`?

A) `undefined`

B) `NIL`

C) `empty`

D) `null`

Antwort: JSON – Frage 2

Frage 2: Was ist das JSON-Äquivalent zum Python-Wert `None`? **Korrekte Antwort: D)** `null`

Quiz: JSON – Frage 3

Sie haben eine JSON-Datei `data.json` und möchten deren Inhalt in ein Python-Objekt laden. Welche Funktion verwenden Sie (angenommen die Datei ist geöffnet als ‘f‘)?

A) `json.loads(f.read())`

B) `json.dump(f)`

C) `json.load(f)`

D) `json.parse(f)`

Antwort: JSON – Frage 3

Frage 3: Sie haben eine JSON-Datei `data.json` und möchten deren Inhalt in ein Python-Objekt laden. Welche Funktion verwenden Sie (angenommen die Datei ist geöffnet als ‘f‘)? **Korrekte Antwort: C)** `json.load(f)`

Quiz: JSON – Frage 4

Welches der folgenden JSON-Schlüssel-Wert-Paare ist syntaktisch korrekt?

A) `'name': "Max"`

B) `äge: '30'`

C) `"city: "Berlin"`

D) `isActive: true`

Antwort: JSON – Frage 4

Frage 4: Welches der folgenden JSON-Schlüssel-Wert-Paare ist syntaktisch korrekt? **Korrekte**

Antwort: C) `"city: "Berlin"` (Schlüssel und Strings in doppelten Anführungszeichen)

Was sind Reguläre Ausdrücke (RegEx)?

RegEx sind **Suchmuster** in Texten.

- Finden, Ersetzen, Validieren von Textteilen.
- Extrem mächtig und flexibel.
- Syntax kann anfangs komplex wirken.

Beispiele für Anwendungsfälle:

- E-Mail-Validierung: `name@domain.com`
- Extraktion von Telefonnummern oder URLs
- Suchen & Ersetzen von Wörtern

RegEx Grundlagen: Einfache Metazeichen

Metazeichen haben eine spezielle Bedeutung:

- `.` Beliebiges Zeichen (außer oft Zeilenumbruch)
- `^` Anfang des Strings
- `$` Ende des Strings
- `|` Oder (z.B. `rot|grün`)

Quantifizierer (wie oft?):

- `*` Null oder öfter (z.B. `ab*` → a, ab, abb)
- `+` Einmal oder öfter (z.B. `ab+` → ab, abb)
- `?` Null oder einmal (z.B. `colou?r` → color, colour)
- `{n}` Genau n-mal (z.B. `x{3}` → xxx)
- `{n,}` Mindestens n-mal
- `{n,m}` Mindestens n, maximal m-mal

RegEx Grundlagen: Zeichenklassen und Gruppen

Zeichenklassen `[]`**:** Definiert eine Menge erlaubter Zeichen.

- `[abc]` 'a', 'b' oder 'c'
- `[a-z]` Jeder Kleinbuchstabe
- `[0-9]` Jede Ziffer
- `[âeiou]` Kein Vokal (Negation mit ^ *innerhalb* der Klasse)

Gruppierung `()`**:**

- Fasst Teile eines Musters zusammen (z.B. `(ab)+`).
- Ermöglicht Extraktion von Teil-Matches.

Escape-Zeichen `\`**:** Hebt Sonderbedeutung auf oder leitet Spezialsequenz ein.

- `\.` `\?` Literal für Punkt, Fragezeichen.

RegEx Grundlagen: Spezielle Sequenzen

Nützliche Abkürzungen:

- `\d` : Jede Ziffer (wie `[0-9]`)
- `\D` : Jedes Nicht-Ziffer-Zeichen
- `\w` : Jedes alphanumerische Zeichen (Buchstabe, Zahl, _) (wie `[a-zA-Z0-9_]`)
- `\W` : Jedes nicht-alphanumerische Zeichen
- `\s` : Jedes Whitespace-Zeichen (Leerz., Tab, \n)
- `\S` : Jedes Nicht-Whitespace-Zeichen
- `\b` : Wortgrenze (z.B. `\bcat\b` findet "cat", nicht "caterpillar")
- `\B` : Keine Wortgrenze

Python und RegEx: Das re-Modul

```
1 import re
```

Kernfunktionen:

- `re.match(pattern, string)`: Sucht Muster **am Anfang** des Strings.
- `re.search(pattern, string)`: Sucht Muster **irgendwo** im String (erster Treffer).
- `re.findall(pattern, string)`: Findet **alle** Treffer (als Liste).
- `re.finditer(pattern, string)`: Wie `findall`, aber gibt Iterator von Match-Objekten.
- `re.sub(pattern, repl, string)`: Suchen und **Ersetzen**.
- `re.split(pattern, string)`: **Teilt** String am Muster.
- `re.compile(pattern)`: Kompiliert Muster für effizientere Wiederverwendung.

Tipp: Raw Strings

Nutze Raw Strings `r"..."` für RegEx-Muster, um Probleme mit Backslashes zu vermeiden (z.B. `r"\d"`).

Das Match-Objekt

`re.search()` und `re.match()` geben bei Erfolg ein **Match-Objekt** zurück, sonst `None`.

```python
import re
text = "Nummer: 0123-45678"
match = re.search(r"(\d{4})-(\d{5})", text)

if match:
    print("Gefunden!")
```

Methoden des Match-Objekts:

- `match.group(0)` oder `match.group()`: Gesamter gematchter Text (`"0123-45678"`).
- `match.group(1)`: Erste geklammerte Gruppe (`"0123"`).
- `match.group(2)`: Zweite geklammerte Gruppe (`"45678"`).
- `match.groups()`: Tupel aller Gruppen (`('0123', '45678')`).
- `match.start()`: Startindex des Matches.
- `match.end()`: Endindex des Matches.

Beispiele: `findall` und `sub`

`re.findall()`: Alle Treffer finden

```python
import re
text = "E-Mails: test@example.com, user@domain.org"
emails = re.findall(r"[\w\.-]+@[\w\.-]+\.\w+", text)
print(emails) # ['test@example.com', 'user@domain.org']
```

`re.sub()`: Suchen und Ersetzen

```python
import re
text = "Hallo Herr Müller, hallo Frau Meier."
neuer_text = re.sub(r"(Herr|Frau)\s", "Person ", text)
print(neuer_text) # Hallo Person Müller, hallo Person Meier.
```

Übungen: RegEx in Python

Übung 1: Skript: Prüfe, ob Eingabe eine gültige dt. PLZ ist (5 Ziffern). Nutze `re.match()`.

Übung 2: Text: "Meeting am 2024-06-15 um 14:30. Nächster Termin 2024-07-01.Ëxtrahiere alle Daten (YYYY-MM-DD) mit `re.findall()`.

Übung 3: String: "Dies Ist Ein TestSATZ". Ersetze alle Großbuchstaben mit 'X' via `re.sub()`.

Übung 4: HTML-String: `"Das ist <b>wichtig</b> und <b>auch fett</b>."` Extrahiere Inhalte der <b>-Tags. Ergebnis: `['wichtig', 'auch fett']`.

Quiz: RegEx in Python – Frage 1

Welches Metazeichen passt auf "null oder mehr"Wiederholungen des vorhergehenden Zeichens/Gruppe?

A) `+`

B) `?`

C) `*`

D) `{0,}`

Antwort: RegEx in Python – Frage 1

Frage 1: Welches Metazeichen passt auf "null oder mehr"Wiederholungen des vorhergehenden Zeichens/Gruppe? **Korrekte Antwort: C)** $*$ *(D ist äquivalent, C ist das primäre Metazeichen)*

Quiz: RegEx in Python – Frage 2

Welche `re`-Funktion ist am besten geeignet, um zu prüfen, ob ein Muster *irgendwo* in einem String vorkommt und das erste Vorkommen als Match-Objekt zurückzugeben?

A) `re.match()`

B) `re.findall()`

C) `re.search()`

D) `re.fullmatch()`

Antwort: RegEx in Python – Frage 2

Frage 2: Welche `re`-Funktion ist am besten geeignet, um zu prüfen, ob ein Muster *irgendwo* in einem String vorkommt und das erste Vorkommen als Match-Objekt zurückzugeben? **Korrekte**

Antwort: C) `re.search()`

Quiz: RegEx in Python – Frage 3

Was ist der Zweck von Raw Strings (`r"..."`) in Python bei der Verwendung von Regulären Ausdrücken?

A) Sie machen den Ausdruck schneller.

B) Sie erlauben Kommentare im Ausdruck.

C) Sie verhindern, dass Backslashes als Escape-Sequenzen von Python interpretiert werden.

D) Sie funktionieren nur mit ASCII-Zeichen.

Antwort: RegEx in Python – Frage 3

Frage 3: Was ist der Zweck von Raw Strings (`r"..."`) in Python bei der Verwendung von Regulären Ausdrücken? **Korrekte Antwort: C) Sie verhindern, dass Backslashes als Escape-Sequenzen von Python interpretiert werden.**

Quiz: RegEx in Python – Frage 4

Der Reguläre Ausdruck `r"\d{3}-\d{3}"` passt auf welchen der folgenden Strings?

A) `abc-def`
B) `1234-567`
C) `12-345`
D) `123-456`

Antwort: RegEx in Python – Frage 4

Frage 4: Der Reguläre Ausdruck `r"\d{3}-\d{3}"` passt auf welchen der folgenden Strings?
Korrekte Antwort: D) `123-456`

Das Wichtigste in Kürze

File Handling:

- Dateien öffnen (`open()`), lesen, schreiben, schließen.
- Sicher mit `with`-Block.
- Modi (`"r"`, `"w"`, `ä"`).
- Löschen mit `os.remove()`, `os.rmdir()`, `shutil.rmtree()`.

JSON:

- Standard für Datenaustausch (Objekte, Arrays, Werte).
- Python-Modul `json` (`dumps`, `loads`, `dump`, `load`).

RegEx:

- Mächtige Suchmuster für Text.
- Python-Modul `re` (`search`, `findall`, `sub`, etc.).
- Metazeichen, Klassen, Quantifizierer.

Wie geht es weiter?

- **Üben, üben, üben!**
- Weitere Dateiformate erkunden: CSV, XML.
- Fortgeschrittene RegEx-Konzepte (Lookarounds, non-greedy).
- Solide Fehlerbehandlung (`try-except`) integrieren.
- Praktische Projekte umsetzen!
- Alle Codebeispiele und Materialien finden Sie auch im Kurs-Repository: jowansulaiman/py-journey

Vielen Dank!

Für Ihre Teilnahme und Aufmerksamkeit

Code & Materialien
https://github.com/jowansulaiman/py-journey