Dateiverarbeitung

- Definition Datei
- Öffnen/Schließen einer Textdatei
- Zugriffsmodi
- Zeilenweise lesen aus einer Textdatei
- Gesamte Datei auf einmal lesen
- Texte in eine Datei schreiben
- with/as-Anweisung (inkl. Exception Handling)
- Textcodierung
- Hausübungsbeispiel

Defintion einer Datei

Eine Datei ist eine Menge von logisch zusammenhängenden und meist sequentiell geordneten Daten, die auf einem Speichermedium dauerhaft gespeichert werden und mittels eines Bezeichners bzw. Namens wieder identifizierbar und damit ansprechbar sind. Die Daten einer Datei existieren also über die Laufzeit eines Programms hinaus. Deswegen werden sie auch als nicht flüchtig oder persistent bezeichnet.

1. Öffnen/Schließen einer Textdatei


```
open(filename, mode)
```

Im folgenden Beispiel öffnen wir die Datei "blindtext.txt" zum Lesen, da der Modus auf "r" (read) gesetzt ist:

```
fobj = open("blindtext.txt", "r")
```

Alternativ hätte man die obige Anweisung auch ohne die Angabe des "r" schreiben können. Fehlt der Modus, wird automatisch lesen zum Defaultwert:

```
fobj = open("blindtext.txt")
```

Auf keinen Fall darf man die *close()*-Methode vergessen, damit die Daten in der Datei konsistent bleiben. Es bedarf dieser Anweisung, um Datei-Objekte ordnungsgemäß zu schließen.

```
fobj.close()
```

Zugriffsmodi der open()-Methode

"r" - read (lesen) = Standardwert:	Öffnet eine Datei zum Lesen. Liefert einen Error, wenn Datei nicht existiert.
"a" – append (anhängen):	Öffnet die Datei für Erweiterungen. Erstellt die Datei, wenn sie nicht existiert.
"w" – write (schreiben):	Öffnet die Datei fürs Schreiben. Überschreibt alle bestehenden Inhalte. Erstellt die Datei, wenn sie nicht existiert
"x" – create (erstellen):	Erstellt eine neue Datei. Liefert einen Error, wenn Datei bereits existiert.

Außerdem kann Binär- oder Textmodus festgelegt werden:

"t" – text = Standardwert:	Textmodus
"b" – binär:	Binärmodus (z.B. Bilder)

Beispiele Dateien öffnen

Folgende Anweisungen zum Öffnen einer Datei sind somit gleichbedeutend, da die Standardwerte weggelassen werden können:

```
f = open("demofile.txt")
f = open("demofile.txt", "r")
f = open("demofile.txt", "rt")
```

2. Zeilenweise lesen aus Textdatei

Schleife über Fileobjekt

Sehr häufig bearbeitet man eine Datei zeilenweise, d.h. man liest eine Datei Zeile für Zeile. (Hinweis: wird inkl. Zeilenumbruch eingelesen.) Das folgende Programm demonstriert, wie man eine Datei zeilenweise einliest und anschließend ausgibt. Um keinen doppelten Zeilenumbruch auszugeben (in der gelesenen Zeile sowie durch die print-Anweisung selbst) werden durch die String-Methode rstrip() vor der Ausgabe etwaige Leerzeichen und Newlines vom rechten Rand der gelesene Zeile entfernt. Alternativ kann kann man auch durch die Option `end=""` den Umbruch in der print-Ausgabe unterbinden.

```
fobj = open("blindtext.txt")
for line in fobj:
    print(line.rstrip())   # print(line, end="")
fobj.close()
```

Übung: Probiere diesen Code aus. Die Datei "blindtext.txt" steht dir als Datei im Arbeitsauftrag zur Verfügung.

2. Zeilenweise lesen aus Textdatei

Methode readline() - einzeln verwenden

Alternativ kann man eine Datei mit der Methode readline() zeilenweise auslesen. readline() liefert immer auch das Zeilenendezeichen jeder Zeile mit \n. Wenn das Ende der Datei erreicht ist, gibt readline() eine leere Zeichenkette zurück. Da leere Zeilen innerhalb der Datei zumindest aus \n bestehen, gibt es hier keine Doppeldeutigkeit.

```
f = open("blindtext.txt", "r")
print(f.readline())     # 1. Zeile
print(f.readline())     # 2. Zeile
print(f.readline(5))    # 5. Zeichen der 3. Zeile
```

2. Zeilenweise lesen aus Textdatei

Methode readline() - in Schleife verwenden

Um händisch eine Datei mit readline() bis zum Ende auszulesen kann folgender Python-Code verwendet werden.

```
fobj = open("blindtext.txt")
zeile = fobj.readline()
while zeile!="":
    print(zeile, end="")
    zeile = fobj.readline()
```

Übung: Probiere diesen Code aus. Die Datei “blindtext.txt” steht dir als Datei im Arbeitsauftrag zur Verfügung.

3. Gesamte Datei auf einmal lesen

Methode readlines() - Inhalte in eine Liste lesen

Bis jetzt wurde die Datei Zeile für Zeile mit einer Schleife verarbeitet.
Aber es kommt öfters vor, dass man eine Datei gerne in eine komplette Datenstruktur einlesen will. z.B. einen String oder eine Liste. Auf diese Art kann die Datei schnell wieder geschlossen werden, und man arbeitet anschließend nur noch auf der Datenstruktur weiter.

```
f = open("blindtext.txt")
lst = f.readlines()          # Gesamter Inhalte in Liste lst gelesen
print(lst[3])                # z.B. 3. Element (= 3. Zeile) ausgeben
```

Listcomprehension, um alle Leerzeichen und Umbrüche gleich zu entfernen:

```
f = open("blindtext.txt")
lst = [line.rstrip() for line in f.readlines()]
```

3. Gesamte Datei auf einmal lesen

Methode readlines() - Inhalte in eine Liste lesen

kompakte Schreibweise in einer Zeile:

```
f = open("blindtext.txt").readlines()
print(f[3])
```

Übung: Probiere die diversen Codes mit der Methode readlines() aus.
Kann dies ebenfalls funktionieren?

```
print(open("blindtext.txt").readlines()[3])
```

3. Gesamte Datei auf einmal lesen

Methode read() - Inhalte in einen String lesen

Optional kann man aber auch die read-Methode verwenden.
Hierbei würde die komplette Datei in einen String eingelesen.

```
blindstr = open("blindtext.txt").read()
print(blindstr)
```

Übung: Probiere diesen Code aus und ändere den Code so um, dass du überprüfen kannst, ob die zurückgelieferte Variable wirklich vom Typ str ist.

4. Texte in eine Datei schreiben

Schreiben in eine Datei lässt sich analog bewerkstelligen.
Beim Öffnen der Datei benutzt man lediglich "w" statt "r" als Modus.
Zum Schreiben der Daten in die Datei benutzt man die Methode write des Dateiobjektes.

```
file = open("test.txt","w")     # Datei zum Schreiben öffnen/erstellen
file.write("Test1\n")           # Zeile1 mit “Test1” und Umbruch schreiben
file.write("\n")                # Zeile2 nur Umbruch schreiben
file.write("Test2\n")           # Zeile 3 mit “Test2” und Umbruch schreiben
file.close()                    # Datei schließen
```

Übung: Probiere diesen Code aus. Die Datei “test.txt” musst du vorher nicht erstellen. Dies passiert bei Nichtvorhandensein durch den open-Befehl von selbst.

4. Texte in eine Datei schreiben

In diesem Beispiel wird zeilenweise aus einer Datei (blindtext.txt) gelesen und neu aufbereitet in eine neue Datei (blindtext_nr.txt) geschrieben:

```
fobj_in = open("blindtext.txt")
fobj_out = open("blindtext_nr.txt","w")
i = 1
for line in fobj_in:
    print(line.rstrip())
    fobj_out.write(str(i) + ": " + line)
    i = i + 1
fobj_in.close()
fobj_out.close()
```

Übung: Überlege vorher, wie der Inhalt der neuen Datei aussehen wird, bevor du den Code testest.

4. Texte in eine Datei schreiben

Achtung auf mögliches Überschreiben!

Im write-Modus wird alles überschrieben, sollte die Datei bereits existieren und Inhalte vorhanden sein.

```
f = open("demofile.txt", "w")
f.write("Woops! Jetzt ist alles weg!")
f.close()
```

Übung: Probiere diese Codes aus. Das “demofile.txt” mit Inhalt steht wieder zur Verfügung.

Im append-Modus kann man Texte an bestehende Inhalte anhängen:
(Hinweis: write fügt im Gegensatz zu print nicht automatisch ein Zeilenende ein!)

```
f = open("demofile.txt", "a")
f.write("\nJetzt hat die Datei mehr Inhalt!\n")
f.close()
```

5. with/as Anweisung

Sollte beim Lese- oder Schreibvorgang ein Fehler passieren, kann es sein, dass die Datei am Ende nicht mehr geschlossen wird und die Datei eventuell unbrauchbar wird. Um dies zu verhindert, gibt es die with/as-Anweisung, ein Konstrukt, das wir verwenden können, um sicherzustellen, dass die Datei immer geschlossen wird. Ein explizites close ist dann nicht mehr nötig!

```
with open("blindtext.txt") as fobj
for line in fobj:
    print(line.rstrip())
```

5. with/as Anweisung

Bereits verwendetes Beispiel nun mit with/as-Anweisung:

```
with open("blindtext.txt") as fobj_in:
    with open("blindtext_nr.txt","w") as fobj_out:
        i = 1
        for line in fobj_in:
            print(line.rstrip())
            fobj_out.write(str(i) + ": " + line)
            i = i + 1
```

5. with/as Anweisung

Vergleiche hierzu die vorherige Folie:
Optimierung open-Anweisungen. Die Tiefe der Einrückungen wird dadurch reduziert:

```
with open("blindtext.txt") as fobj_in, \
    open("blindtext_nr.txt","w") as fobj_out:
    i = 1
    for line in fobj_in:
        print(line.rstrip())
        fobj_out.write(str(i) + ": " + line)
        i = i + 1
```

6. Exception-Handling

With/as garantiert, dass die Datei auch beim Auftreten eines Fehlers geschlossen wird, der eigentliche Fehler (z.B. fehl. Zugriffsrechte, voller Datenträger, nicht vorhand. Datei, etc.) tritt aber anschließend dennoch auf. with/as ersetzt also nicht try/except!

```
try:
    # in eine Textdatei schreiben
    with open('testfile.txt', 'w') as f1:
    f1.write('Hier wird ein Text geschrieben!\n')
    f1.write('Unicodes äöüß\n')

    # Textdatei zeilenweise auslesen
    with open('testfile.txt', 'r') as f2:
    for line in f2:
        print(line, end='')

except BaseException as err:
    print('Fehler:', err)
```

Übung: Teste nun die with/as-Anweisung inkl. des Exception-Handlings und probiere einen Fehler zu verursachen, der abgefangen wird. (Hinweis: Dateiname zum lesen ändern!)

7. Textcodierung

Wenn kein encoding-Parameter beim open-Befehl angeführt sind, verwendet MU die Encodierung “cp1252” (Windows), weshalb die vorliegende Beispieldatei “blindtext.txt” in dieser Codierung abgespeichert wurde, um nicht gleich eine Verwirrung herbeizuführen.

Standardmäßig gilt für Textdateien (z.B. Editor-Textdateien) die Codierung Unicode UTF-8. Um diese nun im MU-Editor richtig auslesen zu können, benötigen wir beim open-Befehl den Parameter encoding=”utf-8”.

```
with open("eigeneDatei.txt","r",encoding="utf-8") as f:
    print(f.read())
```

Übung: Erstelle nun selbst eine Textdatei “eigeneDatei.txt” inkl. Sonderzeichen/Umlaute im Editor und lies diese wie oben angeführt aus.

8. Hausübung
bis zur nächsten Theorie-Einheit

Dieses Übungsbeispiel muss bis zur nächsten Theorie-Einheit für die Mitarbeitsnote abgegeben werden:

Schreibe ein Python-Programm, dass die vorhandene Datei “mitarbeiter.txt” ausliest und die Daten aufbereitet in eine neue Datei “mitarbeiter_neu.txt” schreibt.

Drei Zeilen müssen jeweils zu einer Zeile in der Form “Vorname Nachname (Beruf)” zusammengefasst und in eine neue Datei geschrieben werden.

Inhalt der neuen Datei sollte wie folgt aussehen:

```
Johannes Huber (Architekt)
Matthias Mayer (Pfarrer)
...
```