Quoting (1)

- Was passiert, wenn das Trennzeichen in den Daten vorkommt? Beispiel:

```
with open("quoting.csv", "w", newline="") as fh:
    writer = csv.writer(fh, delimiter=".")
    writer.writerow([1.5, 2.5])
```

Ok, in diesem Beispiel kann man einfach das Trennzeichen ändern, z.B. in Strichpunkt.

- Wenn die Daten aber Strings sind? Beispiel:

```
with open("quoting.csv", "w", newline="") as fh:
    writer = csv.writer(fh, delimiter=",")
    writer.writerow(["ja; prima", "ja, prima", 133])
```

In diesem Fall ist das Ergebnis ja; prima,"ja, prima",133 d.h. Daten, die das Trennzeichen enthalten, werden in Hochkommas eingeschlossen.

Quoting (2)

- Das Quoting lässt sich durch die optionalen Parameter quoting und quotechar der Funktionen reader und writer beeinflussen:

	Der reader	Der writer quotet
quoting=csv.QUOTE_MINIMAL (Default)		nur Daten, die das Trennzeichen, das quotechar oder ein Zeilenendezeichen enthalten
quoting=csv.QUOTE_ALL		alle Daten
quoting=csv.QUOTE_NONNUMERIC	konvertiert alle nicht gequoteten Daten in float	alle nicht-numerischen Daten
quoting=csv.QUOTE_NONE		gar nichts

Das quotechar gibt an, in welche Zeichen Daten beim Quoten eingeschlossen werden. Der Default ist ". Beispiel:

```
quotechar="'"
```

Quoting (3)

- Probieren Sie alle 16 Kombination für quoting anhand des folgenden Beispiels aus!

```
with open("quoting.csv", "w", newline="") as fh:
    writer = csv.writer(fh, delimiter=",", quoting=csv.QUOTE_MINIMAL, quotechar="'")
    # writer = csv.writer(fh, delimiter=",", quoting=csv.QUOTE_ALL, quotechar="'")
    # writer = csv.writer(fh, delimiter=",", quoting=csv.QUOTE_NONNUMERIC, quotechar="'")
    # writer = csv.writer(fh, delimiter=",", quoting=csv.QUOTE_NONE, quotechar="'")
    writer.writerow(["ja; prima", "ja, prima", 133])

with open("quoting.csv", "r", newline="") as fh:
    reader = csv.reader(fh, delimiter=",", quoting=csv.QUOTE_MINIMAL, quotechar="'")
    # reader = csv.reader(fh, delimiter=",", quoting=csv.QUOTE_ALL, quotechar="'")
    # reader = csv.reader(fh, delimiter=",", quoting=csv.QUOTE_NONNUMERIC, quotechar="'")
    # reader = csv.reader(fh, delimiter=",", quoting=csv.QUOTE_NONE, quotechar="'")
    for row in reader:
        print(row)
```

Quoting (4)

- Funktionen des Moduls csv:

def reader(iterable, delimiter=",", quoting=csv.QUOTE_MINIMAL, quotechar='"')

The "iterable" argument can be any object that returns a line of input for each iteration, such as a file object or a list.

The returned object is an iterator. Each iteration returns a row of the CSV file (which can span multiple input lines).

def writer(fileobj, delimiter=",", quoting=csv.QUOTE_MINIMAL, quotechar='"')

The "fileobj" argument can be any object that supports the file API.

def writerow(row) Write the row parameter to the writer's file object.

def writerows(rows)

Write all elements in rows (an iterable of row objects as described above) to the writer's file object.

Die Funktionen haben weitere optionale Parameter, die hier nicht angegeben sind.

Kommandozeilenparameter (1)

- Meist ist es günstige einen Dateinamen nicht fest einzuprogrammieren, sondern variabel zu halten. Neben dem Einlesen von der Konsole, kann ein Dateiname auch als Kommandozeilenparameter übergeben werden. Beispiel:

  noten.py:

```
import sys
print(sys.argv[0])
if len(sys.argv) > 1:
    print(sys.argv[1])
```

  Aufruf in der Konsole:

```
python noten.py punkte.csv
```

  Konsolausgabe:

```
noten.py
punkte.csv
```

Kommandozeilenparameter (2)

- Die Liste sys.argv enthält
  - den Namen des aufgerufenen Skripts sys.argv[0]
  - und die Kommandozeilenparameter sys.argv[1], sys.argv[2], …
- Vor dem Zugriff auf die Kommandozeilenparameter unbedingt prüfen, wie viele vorhanden sind. Das Beispiel funktioniert auch, wenn kein Kommandozeilenparameter eingegeben wird:

  Aufruf in der Konsole:
  python noten.py

  Konsolausgabe:
  noten.py