Lesen und Schreiben von CSV-Dateien in Python

Inhaltsverzeichnis

Seien wir ehrlich: Sie müssen Informationen in und aus Ihren Programmen über mehr als nur die Tastatur und die Konsole abrufen. Der Austausch von Informationen über Textdateien ist eine gängige Methode, um Informationen zwischen Programmen auszutauschen. Eines der beliebtesten Formate für den Datenaustausch ist das CSV-Format. Aber wie benutzt man es?

Lassen Sie uns eines klarstellen: Sie müssen (und werden) keinen eigenen CSV-Parser von Grund auf neu erstellen. Es gibt mehrere vollkommen akzeptable Bibliotheken, die Sie verwenden können. Die Python `csv`Bibliothek wird in den meisten Fällen funktionieren. Wenn Ihre Arbeit viele Daten oder numerische Analysen erfordert, ist die `pandas`Bibliothek verfügt auch über CSV-Parsing-Funktionen, die den Rest erledigen sollten.

In diesem Artikel erfahren Sie, wie Sie CSV aus Textdateien mit Python lesen, verarbeiten und parsen. Sie werden sehen, wie CSV-Dateien funktionieren, und das Wichtigste lernen `csv`Bibliothek, die in Python integriert ist, und sehen Sie, wie CSV-Parsing mit der funktioniert `pandas`Bibliothek .

Also lasst uns anfangen!

Was ist eine CSV-Datei?

Eine CSV-Datei (Comma Separated Values-Datei) ist eine Art Klartextdatei, die eine bestimmte Strukturierung verwendet, um tabellarische Daten anzuordnen. Da es sich um eine einfache Textdatei handelt, kann sie nur tatsächliche Textdaten enthalten – mit anderen Worten, druckbare ASCII- oder Unicode Zeichen.

Die Struktur einer CSV-Datei wird durch ihren Namen verraten. Normalerweise verwenden CSV-Dateien ein Komma, um jeden spezifischen Datenwert zu trennen. So sieht diese Struktur aus:

```
column 1 name,column 2 name, column 3 name
first row data 1,first row data 2,first row data 3
second row data 1,second row data 2,second row data 3
```

```
...
```

Beachten Sie, wie jedes Datenelement durch ein Komma getrennt ist. Normalerweise identifiziert die erste Zeile jedes Datenelement – mit anderen Worten, den Namen einer Datenspalte. Jede nachfolgende Zeile danach sind tatsächliche Daten und werden nur durch Dateigrößenbeschränkungen begrenzt.

Im Allgemeinen wird das Trennzeichen als Trennzeichen bezeichnet, und das Komma ist nicht das einzige, das verwendet wird. Andere beliebte Trennzeichen sind der Tabulator (`\t`), Doppelpunkt (`:`) und Semikolon (`;`) Figuren. Um eine CSV-Datei richtig analysieren zu können, müssen wir wissen, welches Trennzeichen verwendet wird.

Woher kommen CSV-Dateien?

CSV-Dateien werden normalerweise von Programmen erstellt, die große Datenmengen verarbeiten. Sie sind eine bequeme Möglichkeit, Daten aus Tabellenkalkulationen und Datenbanken zu exportieren sowie zu importieren oder in anderen Programmen zu verwenden. Beispielsweise können Sie die Ergebnisse eines Data-Mining-Programms in eine CSV-Datei exportieren und diese dann in eine Tabellenkalkulation importieren, um die Daten zu analysieren, Diagramme für eine Präsentation zu erstellen oder einen Bericht zur Veröffentlichung vorzubereiten.

CSV-Dateien können sehr einfach programmgesteuert bearbeitet werden. Jede Sprache, die die Eingabe von Textdateien und die Manipulation von Zeichenfolgen unterstützt (wie Python), kann direkt mit CSV-Dateien arbeiten.

Analysieren von CSV-Dateien mit der integrierten CSV-Bibliothek von Python

Das `csv`Die Bibliothek bietet Funktionen zum Lesen und Schreiben in CSV-Dateien. Es wurde entwickelt, um sofort mit Excel-generierten CSV-Dateien zu arbeiten, und lässt sich leicht an eine Vielzahl von CSV-Formaten anpassen. Das `csv`Die Bibliothek enthält Objekte und anderen Code zum Lesen, Schreiben und Verarbeiten von Daten aus und in CSV-Dateien.

Lesen von CSV-Dateien mit `csv`

Das Auslesen aus einer CSV-Datei erfolgt über die `reader`Objekt. Die CSV-Datei wird als Textdatei mit eingebautem Python geöffnet `open()`Funktion, die ein Dateiobjekt zurückgibt. Diese wird dann an die weitergegeben `reader`, die das schwere Heben erledigt.

Hier ist die `employee_birthday.txt`Datei:

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

Hier ist Code, um es zu lesen:

```
import csv

with open('employee_birthday.txt') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
```

```
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            print(f'\t{row[0]} works in the {row[1]} department, and was born in
{row[2]}.')
            line_count += 1
    print(f'Processed {line_count} lines.')
```

Dies führt zu folgender Ausgabe:

```
Column names are name, department, birthday month
    John Smith works in the Accounting department, and was born in November.
    Erica Meyers works in the IT department, and was born in March.
Processed 3 lines.
```

Jede Zeile, die von zurückgegeben wird `reader`ist eine Liste von `String`Elemente, die die durch Entfernen der Trennzeichen gefundenen Daten enthalten. Die erste zurückgegebene Zeile enthält die Spaltennamen, die auf besondere Weise gehandhabt werden.

Einlesen von CSV-Dateien in ein Wörterbuch mit `csv`

Anstatt sich mit einer Liste von Einzelpersonen zu befassen `String`-Elemente können Sie CSV-Daten auch direkt in ein Wörterbuch (technisch gesehen ein geordnetes Wörterbuch) einlesen.

Wieder unsere Eingabedatei, `employee_birthday.txt`ist wie folgt:

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

als Wörterbuch diesmal

```
import csv

with open('employee_birthday.txt', mode='r') as csv_file:
    csv_reader = csv.DictReader(csv_file)
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        print(f'\t{row["name"]} works in the {row["department"]} department, and
was born in {row["birthday month"]}.')
        line_count += 1
    print(f'Processed {line_count} lines.')
```

Dies ergibt die gleiche Ausgabe wie zuvor:

```
Column names are name, department, birthday month
    John Smith works in the Accounting department, and was born in November.
    Erica Meyers works in the IT department, and was born in March.
Processed 3 lines.
```

Woher stammen die Wörterbuchschlüssel? Es wird davon ausgegangen, dass die erste Zeile der CSV-Datei die Schlüssel zum Erstellen des Wörterbuchs enthält. Wenn Sie diese nicht in Ihrer CSV-Datei haben, sollten Sie Ihre eigenen Schlüssel angeben, indem Sie die `fieldnames`optionale Parameter zu einer Liste, die sie enthält.

Optionale Python-CSV `reader`Parameter

Das `reader`-Objekt kann verschiedene Stile von CSV-Dateien verarbeiten, indem zusätzliche Parameter , von denen einige unten gezeigt werden:

- `delimiter`gibt das Zeichen an, das zum Trennen der einzelnen Felder verwendet wird. Der Standardwert ist das Komma (`','`).
- `quotechar`gibt das Zeichen an, das verwendet wird, um Felder einzuschließen, die das Trennzeichen enthalten. Der Standardwert ist ein doppeltes Anführungszeichen (`' " '`).
- `escapechar`gibt das Zeichen an, das verwendet wird, um das Trennzeichen zu maskieren, falls keine Anführungszeichen verwendet werden. Der Standardwert ist kein Escape-Zeichen.

Diese Parameter bedürfen einer weiteren Erläuterung. Angenommen, Sie arbeiten mit Folgendem `employee_addresses.txt`Datei:

```
name,address,date joined
john smith,1132 Anywhere Lane Hoboken NJ, 07030,Jan 4
erica meyers,1234 Smith Lane Hoboken NJ, 07030,March 2
```

Diese CSV-Datei enthält drei Felder: `name`, `address`, und `date joined`, die durch Kommas getrennt werden. Das Problem ist, dass die Daten für die `address`enthält auch ein Komma zur Angabe der Postleitzahl.

Es gibt drei verschiedene Möglichkeiten, mit dieser Situation umzugehen:

- **Verwenden Sie ein anderes Trennzeichen**
  Auf diese Weise kann das Komma sicher in den Daten selbst verwendet werden. Sie verwenden die `delimiter`optionaler Parameter zur Angabe des neuen Trennzeichens.
- **Schließen Sie die Daten in Anführungszeichen ein**
  Die Besonderheit des von Ihnen gewählten Trennzeichens wird in Zeichenfolgen in Anführungszeichen ignoriert. Daher können Sie das für die Anführungszeichen verwendete Zeichen mit angeben `quotechar`optionaler Parameter. Solange dieses Zeichen auch nicht in den Daten erscheint, ist alles in Ordnung.
- **Escapezeichen für die Trennzeichen in den Daten**
  Escape-Zeichen funktionieren genauso wie in Format-Strings und machen die Interpretation des Escape-Zeichens (in diesem Fall das Trennzeichen) zunichte. Wenn ein Escape-Zeichen verwendet wird, muss es mit angegeben werden `escapechar`optionaler Parameter.

Schreiben von CSV-Dateien mit `csv`

Sie können auch mit a in eine CSV-Datei schreiben `writer`Objekt und die `.write_row()`Methode:

```
import csv

with open('employee_file.csv', mode='w') as employee_file:
    employee_writer = csv.writer(employee_file, delimiter=',', quotechar='"',
quoting=csv.QUOTE_MINIMAL)
```

```
    employee_writer.writerow(['John Smith', 'Accounting', 'November'])
    employee_writer.writerow(['Erica Meyers', 'IT', 'March'])
```

Das `quotechar`optionaler Parameter teilt dem mit `writer`welches Zeichen verwendet werden soll, um Felder beim Schreiben zu zitieren. Ob Zitieren verwendet wird oder nicht, wird jedoch von der bestimmt `quoting`optionaler Parameter:

- Wenn `quoting`ist eingestellt auf `csv.QUOTE_MINIMAL`, dann `.writerow()`wird Felder nur dann in Anführungszeichen setzen, wenn sie die enthalten `delimiter`oder der `quotechar`. Dies ist der Standardfall.
- Wenn `quoting`ist eingestellt auf `csv.QUOTE_ALL`, dann `.writerow()`wird alle Felder zitieren.
- Wenn `quoting`ist eingestellt auf `csv.QUOTE_NONNUMERIC`, dann `.writerow()`setzt alle Felder mit Textdaten in Anführungszeichen und wandelt alle numerischen Felder in um `float`Datentyp.
- Wenn `quoting`ist eingestellt auf `csv.QUOTE_NONE`, dann `.writerow()`wird Trennzeichen maskieren, anstatt sie zu zitieren. In diesem Fall müssen Sie auch einen Wert für angeben `escapechar`optionaler Parameter.

Das Zurücklesen der Datei im Klartext zeigt, dass die Datei wie folgt erstellt wird:

```
John Smith,Accounting,November
Erica Meyers,IT,March
```

Schreiben einer CSV-Datei aus einem Wörterbuch mit `csv`

Da Sie unsere Daten in ein Wörterbuch einlesen können, ist es nur fair, dass Sie sie auch aus einem Wörterbuch herausschreiben können sollten:

```
import csv

with open('employee_file2.csv', mode='w') as csv_file:
    fieldnames = ['emp_name', 'dept', 'birth_month']
    writer = csv.DictWriter(csv_file, fieldnames=fieldnames)

    writer.writeheader()
    writer.writerow({'emp_name': 'John Smith', 'dept': 'Accounting',
'birth_month': 'November'})
    writer.writerow({'emp_name': 'Erica Meyers', 'dept': 'IT', 'birth_month':
'March'})
```

nicht wie `DictReader`, das `fieldnames`Der Parameter ist beim Schreiben eines Wörterbuchs erforderlich. Das macht Sinn, wenn man darüber nachdenkt: ohne eine Liste von `fieldnames`, das `DictWriter`nicht wissen, welche Schlüssel zum Abrufen von Werten aus Ihren Wörterbüchern verwendet werden sollen. Es verwendet auch die Schlüssel in `fieldnames`um die erste Zeile als Spaltennamen auszuschreiben.

Der obige Code generiert die folgende Ausgabedatei:

```
emp_name,dept,birth_month
John Smith,Accounting,November
Erica Meyers,IT,March
```

Analysieren von CSV-Dateien mit der `pandas`Bibliothek

Natürlich ist die Python-CSV-Bibliothek nicht das einzige Spiel in der Stadt. Das Lesen von CSV-Dateien ist möglich in `pandas`auch. Es wird dringend empfohlen, wenn Sie viele Daten analysieren müssen.

`pandas`ist eine Open-Source-Python-Bibliothek, die leistungsstarke Datenanalysetools und einfach zu verwendende Datenstrukturen bietet. `pandas`ist für alle Python-Installationen verfügbar, aber es ist ein wichtiger Bestandteil der Anaconda Distribution und funktioniert sehr gut in Jupyter-Notebooks , um Daten, Code, Analyseergebnisse, Visualisierungen und erklärenden Text zu teilen.

Installieren `pandas`und seine Abhängigkeiten in `Anaconda`ist einfach gemacht:

```
$ conda install pandas
```

Genauso wie die Verwendung `pip`/ `pipenv`für andere Python-Installationen:

```
$ pip install pandas
```

Wir werden uns nicht mit den Einzelheiten des Wie befassen `pandas`funktioniert oder wie man es benutzt. Für eine eingehende Behandlung über die Verwendung `pandas`große Datensätze zu lesen und zu analysieren, lesen Shantnu Tiwari hervorragenden Artikel Arbeit mit großen Excel-Dateien in Pandas .

Lesen von CSV-Dateien mit `pandas`

Um etwas von der Macht zu zeigen `pandas`CSV-Fähigkeiten habe ich eine etwas komplizierter zu lesende Datei erstellt, genannt `hrdata.csv`. Es enthält Daten über Mitarbeiter des Unternehmens:

```
Name,Hire Date,Salary,Sick Days remaining
Graham Chapman,03/15/14,50000.00,10
John Cleese,06/01/15,65000.00,8
Eric Idle,05/12/14,45000.00,10
Terry Jones,11/01/13,70000.00,3
Terry Gilliam,08/12/14,48000.00,7
Michael Palin,05/23/13,66000.00,8
```

Einlesen der CSV in a `pandas` `DataFrame`geht schnell und unkompliziert:

```
import pandas
df = pandas.read_csv('hrdata.csv')
print(df)
```

Das war's: drei Zeilen Code, und nur eine davon erledigt die eigentliche Arbeit. `pandas.read_csv()`öffnet, analysiert und liest die bereitgestellte CSV-Datei und speichert die Daten in einem DataFrame . Drucken der `DataFrame`ergibt folgende Ausgabe:

```
             Name Hire Date   Salary  Sick Days remaining
0  Graham Chapman  03/15/14  50000.0                   10
1     John Cleese  06/01/15  65000.0                    8
2       Eric Idle  05/12/14  45000.0                   10
3     Terry Jones  11/01/13  70000.0                    3
4   Terry Gilliam  08/12/14  48000.0                    7
5   Michael Palin  05/23/13  66000.0                    8
```

Hier sind einige erwähnenswerte Punkte:

- Zuerst, `pandas`erkannte, dass die erste Zeile der CSV-Datei Spaltennamen enthielt, und verwendete sie automatisch. Ich nenne das Güte.
- Jedoch, `pandas`verwendet auch nullbasierte Integer-Indizes in der `DataFrame`. Das liegt daran, dass wir ihm nicht gesagt haben, was unser Index sein soll.
- Wenn Sie sich außerdem die Datentypen unserer Spalten ansehen, werden Sie sehen `pandas`hat das richtig konvertiert `Salary`und `Sick Days remaining`Spalten zu Zahlen, aber die `Hire Date`Spalte ist immer noch a `String`. Dies lässt sich im interaktiven Modus leicht bestätigen:
- ```
  >>> print(type(df['Hire Date'][0]))
  <class 'str'>
  ```

Lassen Sie uns diese Probleme einzeln angehen. Um eine andere Spalte als die zu verwenden `DataFrame`Index, fügen Sie die hinzu `index_col`optionaler Parameter:

```
import pandas
df = pandas.read_csv('hrdata.csv', index_col='Name')
print(df)
```

Jetzt die `Name`Feld ist unser `DataFrame`Index:

```
               Hire Date   Salary  Sick Days remaining
Name
Graham Chapman  03/15/14  50000.0                   10
John Cleese     06/01/15  65000.0                    8
Eric Idle       05/12/14  45000.0                   10
Terry Jones     11/01/13  70000.0                    3
Terry Gilliam   08/12/14  48000.0                    7
Michael Palin   05/23/13  66000.0                    8
```

Als nächstes fixieren wir den Datentyp der `Hire Date`aufstellen. Sie können zwingen `pandas`um Daten als Datum mit dem zu lesen `parse_dates`optionaler Parameter, der als Liste von Spaltennamen definiert ist, die als Datumsangaben behandelt werden sollen:

```
import pandas
df = pandas.read_csv('hrdata.csv', index_col='Name', parse_dates=['Hire Date'])
print(df)
```

Beachten Sie den Unterschied in der Ausgabe:

```
                Hire Date   Salary  Sick Days remaining
Name
Graham Chapman 2014-03-15  50000.0                   10
John Cleese    2015-06-01  65000.0                    8
Eric Idle      2014-05-12  45000.0                   10
Terry Jones    2013-11-01  70000.0                    3
Terry Gilliam  2014-08-12  48000.0                    7
Michael Palin  2013-05-23  66000.0                    8
```

Das Datum ist nun korrekt formatiert, was im interaktiven Modus einfach bestätigt werden kann:

```
>>> print(type(df['Hire Date'][0]))
<class 'pandas._libs.tslibs.timestamps.Timestamp'>
```
```

Wenn Ihre CSV-Dateien keine Spaltennamen in der ersten Zeile haben, können Sie die verwenden `names`optionaler Parameter, um eine Liste mit Spaltennamen bereitzustellen. Sie können dies auch verwenden, wenn Sie die in der ersten Zeile bereitgestellten Spaltennamen überschreiben möchten. In diesem Fall müssen Sie dies auch mitteilen `pandas.read_csv()`um vorhandene Spaltennamen mit dem zu ignorieren `header=0`optionaler Parameter:

```
import pandas
df = pandas.read_csv('hrdata.csv',
            index_col='Employee',
            parse_dates=['Hired'],
            header=0,
            names=['Employee', 'Hired','Salary', 'Sick Days'])
print(df)
```

Beachten Sie, dass, da sich die Spaltennamen geändert haben, die in der `index_col`und `parse_dates`optionale Parameter müssen ebenfalls geändert werden. Dies führt nun zu folgender Ausgabe:

```
                    Hired   Salary  Sick Days
Employee
Graham Chapman 2014-03-15  50000.0         10
John Cleese    2015-06-01  65000.0          8
Eric Idle      2014-05-12  45000.0         10
Terry Jones    2013-11-01  70000.0          3
Terry Gilliam  2014-08-12  48000.0          7
Michael Palin  2013-05-23  66000.0          8
```

Schreiben von CSV-Dateien mit `pandas`

Natürlich, wenn Sie Ihre Daten nicht herausbekommen `pandas`wieder, es tut dir nicht viel gut. Schreiben ein `DataFrame`in eine CSV-Datei ist genauso einfach wie das Einlesen. Schreiben wir die Daten mit den neuen Spaltennamen in eine neue CSV-Datei:

```
import pandas
df = pandas.read_csv('hrdata.csv',
            index_col='Employee',
            parse_dates=['Hired'],
            header=0,
            names=['Employee', 'Hired', 'Salary', 'Sick Days'])
df.to_csv('hrdata_modified.csv')
```

Der einzige Unterschied zwischen diesem Code und dem obigen Lesecode besteht darin, dass der `print(df)`Anruf wurde durch ersetzt `df.to_csv()`, wobei der Dateiname angegeben wird. Die neue CSV-Datei sieht folgendermaßen aus:

```
Employee,Hired,Salary,Sick Days
Graham Chapman,2014-03-15,50000.0,10
John Cleese,2015-06-01,65000.0,8
Eric Idle,2014-05-12,45000.0,10
Terry Jones,2013-11-01,70000.0,3
Terry Gilliam,2014-08-12,48000.0,7
Michael Palin,2013-05-23,66000.0,8
```

Fazit

Wenn Sie die Grundlagen zum Lesen von CSV-Dateien verstehen, werden Sie beim Importieren von Daten nie auf dem falschen Fuß erwischt. Die meisten CSV-Lese-, -Verarbeitungs- und -Schreibaufgaben können von der Basis problemlos erledigt werden `csv`Python-Bibliothek. Wenn Sie viele Daten lesen und verarbeiten müssen, ist die `pandas`Die Bibliothek bietet auch schnelle und einfache CSV-Handhabungsfunktionen.

Gibt es andere Möglichkeiten, Textdateien zu analysieren? Na sicher! Bibliotheken wie ANTLR , PLY und PlyPlus können alle schweres Parsing bewältigen, und wenn es einfach ist `String`Manipulation funktioniert nicht, es gibt immer reguläre Ausdrücke .

Aber das sind Themen für andere Artikel…