Comma-Separated Values (CSV)

Håndtering, Søgning og Analyse af CSV-Filer

Arinbjörn Brandsson
Benjamin Rotendahl

April 4, 2017

Opbygningen af en CSV-Fil

Første linje beskriver headernavnene, resten er data

```
id,name,description,menus_appeared
1,Consomme printaniere royal,,8
2,Chicken gumbo,,111
3,Tomato aux croutons,,13
```

CSV-Filer kan visualiseres som en tabel

id	name	description	menus_appeared
1	Consomme printaniere royal		8
2	Chicken gumbo		111
3	Tomato aux croutons		13

Indlæsning af CSV-Filer

En måde man kan loade CSV-Filer

```
# Importer csv biblioteket
import csv
# Aaben filen, og kald den csv_file
with open('filename.csv') as csv_file:
    # Opret en laeser til csv_file,
    # og kald den reader
    reader = csv.DictReader(csv_file)
```

Et godt tip er at loade filen ind i sin main funktion, og give læseren som argument til andre funktioner.

Eksempel på en main funktion

```
# Importer csv biblioteket
import csv

# Kode her...

def main():
    with open('file.csv') as csv_file:
        file_reader = csv.DictReader(csv_file)
        print(findNoget(file_reader))

# Denne if-statement bliver altid kaldt
if __name__ == '__main__':
    main()
```

Bearbejdning af CSV-Data

Hvordan arbejder man med læseren?

- Læseren kan læse filens linjer
- Læseren kan huske headersne, og har hver linje i en dictionary
- Læseren kan ikke ændre i CSV-filen
- Så vi skal kunne gennemgå linjerne via læseren

Den oplagte løsning:

Vi kan bruge løkker!

```
def findNoget(reader):
    for row in reader:
        print(row['name'])
```

(Simple) Eksempler

Find retter, givet ved name, og returnér deres ID (ikke case-sensitive)

```
def findName(reader, name):
    results = []
    for row in reader:
        if row['name'].lower() == name.lower():
            results.append(row['id'])
    return results
```

Problem ved denne løsning:

Tager ikke højde for specialtegn som vi måske vil ignorere, såsom komma (mere om dette senere)

Find retter som var var sidst i et menukort 1942

```
def find1942(reader):
    results = []
    for row in reader:
        if int(row['last_appeared']) == 1942:
            results.append(row['name'])
    return results
```

Hvad `int()` gør

- Filens data bliver læst ind som tekststrenge
- Uden `int()` ville sammenligningen være `'1942' == 1942`
- Hvorimod `int('1942') = 1942`
- Det samme kan gøres med `floats`, for eksempel `float('0.8')`

Indskrivning af CSV-Filer

Men hvad hvis vi vil lave nye CSV-filer?

- En læser kan kun læse rækker
- Så vi skal bruger en skriver til at skrive i en fil
- Virker næsten på samme måde
- Man kan enten skrive ind en række ad gangen, eller skrive mange ræker ind på samme tid (`writerow()` og `writerows()`)

Eksempel på at Gemme en Fil

Eksempel: Gem retter som har været i en menu 10 eller flere gange

```
def save15(reader, fileName):
    with open(fileName, 'w') as save_file:
        headers = reader.fieldnames
        writer = csv.DictWriter(save_file,
                                fieldnames=headers)
        writer.writeheader()
        for row in reader:
            if int(row['times_appeared']) > 14:
                writer.writerow(row)
    return True
```

Mulige problemer man kan løbe ind i:

- Man har kun brug for ét resultat
- Manglende data i datasættet
- Dataen er forkert formatteret
- Dataen indeholder specialtegn

Ét Resultat

Men hvad nu hvis vi kun vil have ét datapunkt?

- Python har en del kontrolstrukturer - for eksempel til at afbryde løkker tidligt.
- Siden vores læser bruger en løkke, kan vi bruge kommandoen `break` til at bryde ud af en løkke øjeblikkeligt

Eksempel - stopper når vi finder id 3

```
def foo(reader):
    for row in reader:
        if int(row['id']) == 3:
            temp = row
            break
    return temp
```

Manglende Data

Hvad gør man hvis der mangler data?

- Nogle gange mangler datasættet datapunkter
- Nogle sammenligninger med manglende datapunkter kan få programmet til at crashe
- For at forhindre dette, skal man tage højde for manglende datapunkter (I Python denoteret med den tomme streng, `''`). Her har man to muligheder:
  1. Inkludere rækken alligevel (nemt)
  2. Ignorere rækken (også nemt)
- I første tilfælde gør man som man har gjort før
- I andet tilfælde bruger man kommandoen `continue`, som afbryder den nuværende iteration, og går direkte til den næste iteration.

Eksempel fra Før

```
def findName(reader, name):
    results = []
    for row in reader:
        temp = row['name']
        if temp == '':
            continue
        if temp.lower() == name.lower():
            results.append(row['id'])
    return results
```

Bemærk:

`continue` kan blive simuleret med `if:...  else:...`

Forkert-Formatteret Data

Nogle gange er dataen ikke komma-separeret

- Nogle gange er dataen separeret med noget andet end et komma
- Løsning: Fortæl `csv.DictReader` hvad den nye delimiter er

Eksempel: CSV-fil med punktum i stedet for kommaer

```
with open('punktummer.csv') as csv_file:
    reader = csv.DictReader(csv_file, delimiter='.')
    # Kode her
```

Specialkarakterer

Specialkarakterer i ord

- Nogle gange indeholder dataen specialkarakterer som man ikke er interesseret i
- Problematik ved dette: `'hello,' == 'hello'`
- Løsning: Skrub specialkaraktererne fra
  - Vi kommer til at bruge sort magi til dette

Eksempel:

```
import csv
import re
def findName(reader, name):
    results = []
    for row in reader:
        # [^\w] fanger alle tegn som
        # ikke er bogstaver eller tal
        temp = re.sub(r'[^\w]', '' ,row['name'])
        if temp == '':
            continue
        if  temp.lower() == name.lower():
            results.append(row['id'])
    return results
```