csv — Leitura e escrita de arquivos CSV

Código-fonte: Lib/csv.py

O chamado formato CSV (Comma Separated Values) é o formato mais comum de importação e exportação de planilhas e bancos de dados. O formato CSV foi usado por muitos anos antes das tentativas de descrever o formato de maneira padronizada em **RFC 4180**. A falta de um padrão bem definido significa que existem diferenças sutis nos dados produzidos e consumidos por diferentes aplicativos. Essas diferenças podem tornar irritante o processamento de arquivos CSV de várias fontes. Ainda assim, enquanto os delimitadores e os caracteres de citação variam, o formato geral é semelhante o suficiente para que seja possível escrever um único módulo que possa manipular eficientemente esses dados, ocultando os detalhes da leitura e gravação dos dados do programador.

O módulo `csv` implementa classes para ler e gravar dados tabulares no formato CSV. Ele permite que os programadores digam “escreva esses dados no formato preferido pelo Excel” ou “leia os dados desse arquivo gerado pelo Excel”, sem conhecer os detalhes precisos do formato CSV usado pelo Excel. Os programadores também podem descrever os formatos CSV entendidos por outros aplicativos ou definir seus próprios formatos CSV para fins especiais.

Os objetos de `reader` e `writer` do módulo `csv` leem e escrevem sequências. Os programadores também podem ler e gravar dados no formato de dicionário usando as classes `DictReader` e `DictWriter`.

Ver também:

PEP 305 - API de arquivo CSV
A proposta de melhoria do Python que propôs essa adição ao Python.

Conteúdo do módulo

O módulo `csv` define as seguintes funções:

`csv.`**`reader`**(*csvfile*, *dialect='excel'*, ***fmtparams*)

Retorna um objeto leitor que irá iterar sobre as linhas no *csvfile* fornecido. *csvfile* pode ser qualquer objeto que possua suporte ao protocolo iterador e retorne uma string sempre que o método `__next__()` for chamado — arquivos objeto e objetos lista são ambos adequado. Se *csvfile* for um objeto de arquivo, ele deverá ser aberto com `newline=''`. [1] Pode ser fornecido um parâmetro opcional *dialect*, usado para definir um conjunto de parâmetros específicos para um dialeto CSV específico. Pode ser uma instância de uma subclasse da classe `Dialect` ou uma das strings retornadas pela função `list_dialects()`. Os outros argumentos nomeados opcionais *fmtparams* podem ser dados para substituir parâmetros de formatação individuais no dialeto atual. Para detalhes completos sobre os parâmetros de dialeto e formatação, consulte a seção Dialetos e parâmetros de formatação.

Cada linha lida no arquivo csv é retornada como uma lista de cadeias. Nenhuma conversão automática de tipo de dados é executada, a menos que a opção de formato `QUOTE_NONNUMERIC` seja especificada (nesse caso, os campos não citados são transformados em pontos flutuantes).

Um pequeno exemplo de uso:

```
>>> import csv
>>> with open('eggs.csv', newline='') as csvfile:
...     spamreader = csv.reader(csvfile, delimiter=' ', quotechar='|')
...     for row in spamreader:
```

```
Spam, Lovely Spam, Wonderful Spam
```

csv.**writer**(*csvfile*, *dialect='excel'*, ***fmtparams*)

Retorna um objeto de escrita responsável por converter os dados de usuário em strings delimitadas no objeto arquivo ou similar. *csvfile* pode ser qualquer objeto com um método `write()`. Se *csvfile* for um objeto arquivo, ele deverá ser aberto com `newline=''` [1]. Pode ser fornecido um parâmetro opcional *dialect*, usado para definir um conjunto de parâmetros específicos para um dialeto CSV específico. Pode ser uma instância de uma subclasse da classe `Dialect` ou uma das strings retornadas pela função `list_dialects()`. Os outros argumentos nomeados opcionais *fmtparams* podem ser dados para substituir parâmetros de formatação individuais no dialeto atual. Para detalhes completos sobre os parâmetros de dialeto e formatação, consulte a seção Dialetos e parâmetros de formatação. Para tornar o mais fácil possível a interface com os módulos que implementam a API do DB, o valor `None` é escrito como uma string vazia. Embora isso não seja uma transformação reversível, torna mais fácil despejar valores de dados SQL NULL em arquivos CSV sem preprocessar os dados retornados de uma chamada `cursor.fetch*`. Todos os outros dados que não são de strings são codificados com `str()` antes de serem escritos.

Um pequeno exemplo de uso:

```
import csv
with open('eggs.csv', 'w', newline='') as csvfile:
    spamwriter = csv.writer(csvfile, delimiter=' ',
                            quotechar='|', quoting=csv.QUOTE_MINIMAL)
    spamwriter.writerow(['Spam'] * 5 + ['Baked Beans'])
    spamwriter.writerow(['Spam', 'Lovely Spam', 'Wonderful Spam'])
```

csv.**register_dialect**(*name*[, *dialect*[, ***fmtparams*]])

Associa *dialect* a *name*. *name* deve ser uma string. O dialeto pode ser especificado passando uma subclasse de `Dialect` ou por *fmtparams* argumentos nomeados, ou ambos, com argumentos nomeados substituindo os parâmetros do dialeto. Para detalhes completos sobre os parâmetros de dialeto e formatação, consulte a seção Dialetos e parâmetros de formatação.

csv.**unregister_dialect**(*name*)

Exclui o dialeto associado ao *name* do registro do dialeto. Um `Error` é levantado se *name* não for um nome de dialeto registrado.

csv.**get_dialect**(*name*)

Retorna o dialeto associado a *name*. Um `Error` é levantado se *name* não for um nome de dialeto registrado. Esta função retorna uma classe `Dialect` imutável.

csv.**list_dialects**()

Retorna os nomes de todos os dialetos registrados

csv.**field_size_limit**([*new_limit*])

Retorna o tamanho máximo atual do campo permitido pelo analisador sintático. Se *new_limit* for fornecido, este se tornará o novo limite.

O módulo `csv` define as seguintes classes:

class csv.**DictReader**(*f*, *fieldnames=None*, *restkey=None*, *restval=None*, *dialect='excel'*, **args*, ***kwds*)

Cria um objeto que funcione como um leitor comum, mas mapeia as informações em cada linha para `dict` cujas chaves são fornecidas pelo parâmetro opcional *fieldnames*.

determinados, o dicionário preserva sua ordem original.

Se uma linha tiver mais campos que nomes de campo, os dados restantes serão colocados em uma lista e armazenados com o nome do campo especificado por *restkey* (o padrão é `None`). Se uma linha que não estiver em branco tiver menos campos que nomes de campo, os valores ausentes serão preenchidos com o valor *restval* (o padrão é `None`).

Todos os outros argumentos nomeados ou opcionais são passados para a instância subjacente de `reader`.

Alterado na versão 3.6: Linhas retornadas agora são do tipo `OrderedDict`.

Alterado na versão 3.8: As linhas retornadas agora são do tipo `dict`.

Um pequeno exemplo de uso:

```
>>> import csv
>>> with open('names.csv', newline='') as csvfile:
...     reader = csv.DictReader(csvfile)
...     for row in reader:
...         print(row['first_name'], row['last_name'])
...
Eric Idle
John Cleese

>>> print(row)
{'first_name': 'John', 'last_name': 'Cleese'}
```

class csv.**DictWriter**(*f*, *fieldnames*, *restval=''*, *extrasaction='raise'*, *dialect='excel'*, **args*, ***kwds*)

Cria um objeto que funcione como um de escrita comum, mas mapeia dicionários nas linhas de saída. O parâmetro *fieldnames* é uma `sequência` de chaves que identificam a ordem na qual os valores no dicionário transmitidos para o método `writerow()` são escritos no arquivo *f*. O parâmetro opcional *restval* especifica o valor a ser escrito se o dicionário estiver com falta de uma chave em *fieldnames*. Se o dicionário transmitido para o método `writerow()` contiver uma chave não encontrada em *fieldnames*, o parâmetro opcional *extrasaction* indica qual ação executar. Se estiver definido como `'raise'`, o valor padrão, a `ValueError` é levantada. Se estiver definido como `'ignore'`, valores extras no dicionário serão ignorados. Quaisquer outros argumentos nomeados ou opcionais são passados para a instância subjacente de `writer`.

Observe que, diferentemente da classe `DictReader`, o parâmetro *fieldnames* da classe `DictWriter` não é opcional.

Um pequeno exemplo de uso:

```
import csv

with open('names.csv', 'w', newline='') as csvfile:
    fieldnames = ['first_name', 'last_name']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

    writer.writeheader()
    writer.writerow({'first_name': 'Baked', 'last_name': 'Beans'})
    writer.writerow({'first_name': 'Lovely', 'last_name': 'Spam'})
    writer.writerow({'first_name': 'Wonderful', 'last_name': 'Spam'})
```

class csv.**Dialect**

diferentes aplicativos produzem dados CSV sutilmente diferentes. As instâncias de `Dialect` definem como as instâncias `reader` e `writer` se comportam.

Todos os nomes de `Dialect` disponíveis são retornados por `list_dialects()`, e eles podem ser registrados com classes `reader` e `writer` específicas através de suas funções inicializadoras (`__init__`) como esta:

```
import csv

with open('students.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile, dialect='unix')
                                 ^^^^^^^^^^^^^^
```

class `csv.`**`excel`**

A classe `excel` define as propriedades usuais de um arquivo CSV gerado pelo Excel. Ele é registrado com o nome do dialeto `'excel'`.

class `csv.`**`excel_tab`**

A classe `excel_tab` define as propriedades usuais de um arquivo delimitado por TAB gerado pelo Excel. Ela é registrado com o nome do dialeto `'excel-tab'`.

class `csv.`**`unix_dialect`**

A classe `unix_dialect` define as propriedades usuais de um arquivo CSV gerado em sistemas UNIX, ou seja, usando `'\n'` como terminador de linha e citando todos os campos. É registrado com o nome do dialeto `'unix'`.

Novo na versão 3.2.

class `csv.`**`Sniffer`**

A classe `Sniffer` é usada para deduzir o formato de um arquivo CSV.

A classe `Sniffer` fornece dois métodos:

`sniff`(*`sample`*, *`delimiters=None`*)

Analisa a *sample* fornecida e retorna uma subclasse `Dialect`, refletindo os parâmetros encontrados. Se o parâmetro opcional *delimiters* for fornecido, ele será interpretado como uma string contendo possíveis caracteres válidos de delimitador.

`has_header`(*`sample`*)

Analisa o texto de amostra (presume-se que esteja no formato CSV) e retorne `True` se a primeira linha parecer uma série de cabeçalhos de coluna. Inspecionando cada coluna, um dos dois critérios principais será considerado para estimar se a amostra contém um cabeçalho:

- da segunda até a enésima linha contém valores numéricos
- da segunda até a enésima linha contém strings em que pelo menos o comprimento de um valor difere daquele do cabeçalho putativo dessa coluna.

Vinte linhas após a primeira linha são amostradas; se mais da metade das colunas + linhas atenderem aos critérios, `True` será retornado.

Nota: Este método é uma heurística aproximada e pode produzir falsos positivos e negativos.

Um exemplo para uso de `Sniffer`:

```
    csvfile.seek(0)
    reader = csv.reader(csvfile, dialect)
    # ... process CSV file contents here ...
```

O módulo `csv` define as seguintes constantes:

`csv.`**QUOTE_ALL**
Instrui objetos `writer` a colocar aspas em todos os campos.

`csv.`**QUOTE_MINIMAL**
Instrui objetos `writer` a colocar aspas apenas nos campos que contêm caracteres especiais como *delimiters*, *quotechar* ou qualquer um dos caracteres em *lineterminator*.

`csv.`**QUOTE_NONNUMERIC**
Instrui objetos `writer` a colocar aspas em todos os campos não numéricos.

Instrui o leitor a converter todos os campos não citados no tipo *float*.

`csv.`**QUOTE_NONE**
Instrui objetos `writer` a nunca colocar aspas nos campos. Quando o *delimiter* atual ocorre nos dados de saída, é precedido pelo caractere *escapechar* atual. Se *escapeechar* não estiver definido, o escritor levantará `Error` se algum caractere que exija escape for encontrado.

Instrui `reader` a não executar nenhum processamento especial de caracteres de aspas.

O módulo `csv` define a seguinte exceção:

exception `csv.`**Error**
Levantada por qualquer uma das funções quando um erro é detectado.

Dialetos e parâmetros de formatação

Para facilitar a especificação do formato dos registros de entrada e saída, parâmetros específicos de formatação são agrupados em dialetos. Um dialeto é uma subclasse da classe `Dialect` com um conjunto de métodos específicos e um único método `validate()`. Ao criar objetos `reader` ou `writer`, o programador pode especificar uma string ou uma subclasse da classe `Dialect` como parâmetro de dialeto. Além do parâmetro *dialect*, ou em vez do parâmetro *dialect*, o programador também pode especificar parâmetros de formatação individuais, com os mesmos nomes dos atributos definidos abaixo para a classe `Dialect`.

Os dialetos possuem suporte aos seguintes atributos:

`Dialect.`**delimiter**
Uma string de um caractere usada para separar campos. O padrão é `','`.

`Dialect.`**doublequote**
Controla como as instâncias de *quotechar* que aparecem dentro de um campo devem estar entre aspas. Quando `True`, o caractere é dobrado. Quando `False`, o *escapechar* é usado como um prefixo para o *quotechar*. O padrão é `True`.

Na saída, se *doublequote* for `False` e não *escapeechar* for definido, `Error` é levantada se um *quotechar* é encontrado em um campo.

`Dialect.`**escapechar**

do caractere seguinte. O padrão é `None`, que desativa o escape.

`Dialect.`**`lineterminator`**
A string usada para terminar as linhas produzidas pelo `writer`. O padrão é `'\r\n'`.

> **Nota:** A `reader` é codificada para reconhecer `'\r'` ou `'\n'` como fim de linha e ignora *lineterminator*. Esse comportamento pode mudar no futuro.

`Dialect.`**`quotechar`**
Uma string de um caractere usada para citar campos contendo caracteres especiais, como *delimiter* ou *quotechar*, ou que contêm caracteres de nova linha. O padrão é `'"'`.

`Dialect.`**`quoting`**
Controla quando as aspas devem ser geradas pelo escritor e reconhecidas pelo leitor. Ele pode assumir qualquer uma das constantes `QUOTE_*` (consulte a seção Conteúdo do módulo) e o padrão é `QUOTE_MINIMAL`.

`Dialect.`**`skipinitialspace`**
Quando `True`, os espaços em branco imediatamente após o *delimiter* são ignorados. O padrão é `False`.

`Dialect.`**`strict`**
Quando `True`, levanta a exceção `Error` em uma entrada CSV ruim. O padrão é `False`.

Objetos Reader

Os objetos Reader (instâncias `DictReader` e objetos retornados pela função `reader()`) têm os seguintes métodos públicos:

`csvreader.`**`__next__`**`()`
Retorna a próxima linha do objeto iterável do leitor como uma lista (se o objeto foi retornado de `leitor()`) ou um dict (se for uma instância de `DictReader`), analisado de acordo com a `Dialect` atual. Normalmente, você deve chamar isso de `next(reader)`.

Os objetos Reader possuem os seguintes atributos públicos:

`csvreader.`**`dialect`**
Uma descrição somente leitura do dialeto em uso pelo analisador sintático.

`csvreader.`**`line_num`**
O número de linhas lidas no iterador de origem. Não é o mesmo que o número de registros retornados, pois os registros podem abranger várias linhas.

Os objetos DictReader têm o seguinte atributo público:

`csvreader.`**`fieldnames`**
Se não for passado como parâmetro ao criar o objeto, esse atributo será inicializado no primeiro acesso ou quando o primeiro registro for lido no arquivo.

Objetos Writer

dicionário mapeando nomes de campos para strings ou números (passando-os por `str()` primeiro) para `DictWriter`. Observe que números complexos são escritos cercados por parênteses. Isso pode causar alguns problemas para outros programas que leem arquivos CSV (supondo que eles suportem números complexos).

`csvwriter.`**`writerow`**(*row*)

Escreve o parâmetro *row* no objeto arquivo do escritor, formatado de acordo com a `Dialect` atual. Retorna o valor de retorno da chamada ao método *write* do objeto arquivo subjacente.

Alterado na versão 3.5: Adicionado suporte a iteráveis arbitrários.

`csvwriter.`**`writerows`**(*rows*)

Escreve todos os elementos em *rows* (um iterável de objetos *row*, conforme descrito acima) no objeto arquivo do escritor, formatado de acordo com o dialeto atual.

Os objetos Writer têm o seguinte atributo público:

`csvwriter.`**`dialect`**

Uma descrição somente leitura do dialeto em uso pelo escritor.

Os objetos DictWriter têm o seguinte método público:

`DictWriter.`**`writeheader`**()

Escreve uma linha com os nomes dos campos (conforme especificado no construtor) no objeto arquivo do escritor, formatado de acordo com o dialeto atual. Retorna o valor de retorno da chamada `csvwriter.writerow()` usada internamente.

Novo na versão 3.2.

Alterado na versão 3.8: `writeheader()` agora também retorna o valor retornado pelo método `csvwriter.writerow()` que ele usa internamente.

Exemplos

O exemplo mais simples de leitura de um arquivo CSV:

```
import csv
with open('some.csv', newline='') as f:
    reader = csv.reader(f)
    for row in reader:
        print(row)
```

Lendo um arquivo com um formato alternativo:

```
import csv
with open('passwd', newline='') as f:
    reader = csv.reader(f, delimiter=':', quoting=csv.QUOTE_NONE)
    for row in reader:
        print(row)
```

O exemplo de escrita possível mais simples possível é:

```
import csv
with open('some.csv', 'w', newline='') as f:
```

Como `open()` é usado para abrir um arquivo CSV para leitura, o arquivo será decodificado por padrão em unicode usando a codificação padrão do sistema (consulte `locale.getpreferredencoding()`). Para decodificar um arquivo usando uma codificação diferente, use o argumento `encoding` do open:

```
import csv
with open('some.csv', newline='', encoding='utf-8') as f:
    reader = csv.reader(f)
    for row in reader:
        print(row)
```

O mesmo se aplica à escrita em algo diferente da codificação padrão do sistema: especifique o argumento de codificação ao abrir o arquivo de saída.

Registrando um novo dialeto:

```
import csv
csv.register_dialect('unixpwd', delimiter=':', quoting=csv.QUOTE_NONE)
with open('passwd', newline='') as f:
    reader = csv.reader(f, 'unixpwd')
```

Um uso um pouco mais avançado do leitor — capturando e relatando erros:

```
import csv, sys
filename = 'some.csv'
with open(filename, newline='') as f:
    reader = csv.reader(f)
    try:
        for row in reader:
            print(row)
    except csv.Error as e:
        sys.exit('file {}, line {}: {}'.format(filename, reader.line_num, e))
```

E embora o módulo não tenha suporte diretamente à análise sintática de strings, isso pode ser feito facilmente:

```
import csv
for row in csv.reader(['one,two,three']):
    print(row)
```

Notas de rodapé

1(1,2) Se `newline=''` não for especificado, as novas linhas incorporadas nos campos entre aspas não serão interpretadas corretamente, e nas plataformas que usam fim de linha `\r\n` na escrita, um `\r` extra será adicionado. Sempre deve ser seguro especificar `newline=''`, já que o módulo csv faz seu próprio tratamento de nova linha (universal).