# Ficheiros de Valores Separados por Vírgulas (CSV)

Programação II

Universidade de Lisboa
Faculdade de Ciências
Departamento de Informática
Licenciatura em Tecnologias da Informação

Vasco Thudichum Vasconcelos

Abril 2017

## Introdução

*Valores separados por vírgulas* (em inglês, CSV ou *comma-separated values*) é um formato popular para troca de dados em ficheiros, suportado por folhas de cálculo e por bases de dados. Todos as folhas de cálculo (por exemplo, Microsoft Excel, LibreOffice Calc, Apple Numbers) são capazes de abrir este tipo de ficheiros e de guardar dados neste formato.

Um ficheiro CSV é um ficheiro de texto comum, onde os dados estão organizados por linhas e os valores em cada linha estão separados por um dado caracter separador. Estes ficheiros têm geralmente a extensão `.csv`. O separador por omissão é a vírgula (de onde advém o nome *comma-separated values*).

Na língua portuguesa, a parte inteira de um número é separada da parte decimal por uma vírgula, o que gera confusão se pretendermos guardar números em ficheiros CSV separados por vírgulas. Por exemplo os números `3,14` e `2,71` guardados num ficheiro CSV ficariam com o formato `3,14,2,71` e seriam interpretados como 4 números (`3`, `14`, `2` e `71`). Deste modo utilizamos como separador o ponto e vírgula, eliminando deste modo a ambiguidade: `3,14;2,71`.

Se guardarmos uma folha de cálculo como esta:

| | | |
|---|---|---|
| **Ana** | Silva | 1998 |
| **Pedro** | Castro | 2000 |
| **Eva** | Sousa | 1999 |
| **Nuno** | Martins | 1999 |

no formato CSV, obtemos um ficheiro com quatro linhas:

```
Ana;Silva;1998
Pedro;Castro;2000
Eva;Sousa;1999
Nuno;Martins;1999
```

Para manipular os dados constantes num ficheiro CSV, precisamos de o ler para memória e de o guardar numa estrutura de dados. As estruturas de dados mais comuns para este efeito são as *listas de listas* e as *listas de dicionários*. O resto deste documento está organizado em quarto partes: leitura e escrita de listas de listas, leitura e escrita de listas de dicionários.

## Leitura de ficheiros CSV para listas de listas

A linguagem Python conta com um módulo para manipulação de ficheiros CSV, chamado `csv`. Para informação sobre este módulo consultar *CSV File Reading and Writing*.

Quando pretendemos ler um ficheiro para listas, o padrão típico é o seguinte:

```
import csv
with open('ficheiro.csv', 'rU') as ficheiro_csv:
  leitor = csv.reader(ficheiro_csv, delimiter=';')
```

No código acima `leitor` é um *iterador*, Iterator, e pode ser usado num ciclo `for`, como por exemplo:

```
for linha in leitor:
  # fazer algo com linha
```

A razão para usar `'rU'` está relacionada com a função `open()`, e não propriamente com CSV. O parâmetro `'U'` (de Universal), aceita mudanças de linha com a convenção Unix (`'\n'`) e com a convenção Windows (`'\r\n'`) . Usamos o parâmetro opcional `delimiter=';'` para dizer que pretendemos usar o ponto e vírgula como separadador.

Eis um pequeno exercício de aquecimento: ler o conteúdo de um ficheiro CSV e escrevê-lo no ecrã.

```
def imprimir_csv(nome_ficheiro):
  """
  Requires: nome_ficheiro é uma string e representa um
    ficheiro CSV usando ponto e vírgula como separador.
  Ensures: Imprime o conteúdo do ficheiro no ecrã.
  """
  with open(nome_ficheiro, 'rU') as ficheiro_csv:
    leitor = csv.reader(ficheiro_csv, delimiter = ';')
    for linha in leitor:
      print linha
```

Se correrrmos a função `imprimir_csv` sobre o ficheiro CSV acima descrito obtemos o seguinte resultado:

```
>>> imprimir_csv('pessoas.csv')
['Ana', 'Silva', '1998']
['Pedro', 'Castro', '2000']
['Eva', 'Sousa', '1999']
['Nuno', 'Martins', '1999']
```

De notar que todos elementos constantes nas listas são strings. Em particular, se pretendermos efectuar cálculos aritméticos com as idades temos primeiro de converter as strings em números.

Em vez de ler para o ecrã, podemos ler para uma estrutura de dados que se possa mais tarde manipular. Dado que cada linha do ficheiro é lida para uma lista, a estrutura de dados em que estamos interessados é uma *lista de listas*. Eis uma função que dado o nome de um ficheiro CSV devolve uma lista de listas com o seu conteúdo. De notar a utilização da função **`list`** para converter o iterador `leitor` numa lista.

```
def ler_csv(nome_ficheiro):
  """
  Requires: nome_ficheiro é uma string e representa um
    ficheiro de valores separados por ponto e vírgula.
  Ensures: Devolve uma lista de listas de strings com toda a
    informação constante no ficheiro. Cada elemento da lista
    contém uma linha do ficheiro CSV. Cada string corresponde a
    um valor no ficheiro CSV.
  """
  with open(nome_ficheiro, 'rU') as ficheiro_csv:
    leitor = csv.reader(ficheiro_csv, delimiter = ';')
    return list(leitor)
```

Em vez de convertermos o `leitor` diretamente numa lista ou de utilizarmos um ciclo **`for`** para iterar sobre as linhas de ficheiro CSV, podemos chamar o método `leitor.next()` para obter os sucessivos elementos no iterador. A exceção `StopIteration` é levantada quando tentamos fazer um **`next`**`()` e não há mais elementos no iterador, isto é, o ficheiro CSV chegou ao fim.

Se pretendermos ler, não todo o ficheiro CSV, mas apenas as primeiras linhas podemos usar a seguinte função.

```
def ler_primeiros_csv(nome_ficheiro, n):
    """
    Requires: nome_ficheiro é uma string e representa um
    ficheiro de valores separados por ponto-e-vírgula. O
    ficheiro deverá conter pelo menos n linhas.
    n é um número não negativo.
    Ensures: Devolve uma lista de listas de strings com as
    primeiras n linhas do ficheiro. Cada elemento da lista
    contém uma linha do ficheiro CSV. Cada string corresponde a
    um valor no ficheiro CSV.
```

```
    """
    resultado = []
    with open(nome_ficheiro, 'rU') as ficheiro_csv:
        leitor = csv.reader(ficheiro_csv, delimiter = ';')
        for _ in xrange (n):
            resultado.append(leitor.next())
    return resultado
```

Se pretendermos apenas as duas primeiras linhas do ficheiro `'pessoas.csv'` podemos escrever:

```
>>> ler_primeiros_csv('pessoas.csv', 2)
[['Ana', 'Silva', '1998'], ['Pedro', 'Castro', '2000']]
```

## Escrita de ficheiros CSV a partir de listas de listas

O padrão típico para escrever num ficheiro CSV é o seguinte.

```
import csv
with open('ficheiro.csv', 'wb') as ficheiro_csv:
    escritor = csv.writer(ficheiro_csv, delimiter=';')
    escritor.writerow([0, 1, 2, 3, 4, 5])
    escritor.writerow([0, 2, 4, 6, 8, 10])
```

Em Windows, a opção `'b'` abre o ficheiro em modo binário. A linguagem Pyton em Windows distingue entre ficheiros de texto e ficheiros binários: os caracteres de fim de linha são diferentes em ambos os casos. Em Unix, a opção `'b'` não produz efeito algum, e portanto usamos sempre o modo `wb` de modo a ficarmos independentes da plataforma.

A função abaixo escreve uma lista de listas num ficheiro em formato CSV.

```
def escrever_csv(nome_ficheiro, lista_de_listas):
  """
  Requires: nome_ficheiro é uma string e descreve o nome do
    ficheiro CSV onde escrever a lista.
  Ensures: Escreve a lista_de_listas no ficheiro, uma lista
    por linha, valores separados por ponto e vírgula
  """
  with open(nome_ficheiro, 'wb') as ficheiro_csv:
     escritor = csv.writer(ficheiro_csv, delimiter = ';')
     for linha in lista_de_listas:
       escritor.writerow(linha)
```

Estamos agora em condições de escrever uma aplicação típica de manipulação de ficheiros CSV:

1. Ler um ficheiro CSV para memória;
2. Manipular estes dados para produzir uma outra lista de listas;
3. Escrever o novos dados num ficheiro CSV.

Imaginemos que temos um ficheiro CSV de números apenas, qualquer coisa como:

```
-1;-3;7;5
0;8
-3;-4;-2
```

e que pretendemos escrever num novo ficheiro os valores mínimos e máximos de cada linha constante no ficheiro original:

```
-3;7
0;8
-4;-2
```

Além disso, vamos admitir que não há linhas vazias (para simplificar o cálculo do mínimo e do máximo).

Dadas as funções de escrita e leitura de ficheiros CSV a nossa tarefa fica simplificada. Para ler utilizamos:

```
linhas = ler_csv(nome_ficheiro_entrada)
```

Mas `linhas` é uma lista de listas de strings. Há que converter cada elemento para inteiro. Como pretendemos manter a estrutura dos dados usamos a função **`map`**; como estamos em presença de uma *lista de listas*, usamos um **`map`** dentro de um **`map`**:

```
linhas_numeros = map(lambda l: map(int, l), linhas)
```

Para calcular os mínimos e os máximos usamos as funções **`min`** e **`max`** que, dada uma lista de números devolvem o menor e o maior número na lista, respetivamente. Como pretendemos aplicar a função **`max`** a todas as linhas usamos um **`map`**. Finalmente, porque queremos um valor por linha, temos de colocar o resultado do **`min`** e do **`max`** dentro de uma lista. Eis o resultado:

```
linhas_maximos = map(lambda l: [min(l), max(l)],
    linhas_numeros)
```

Juntando todos os pedaços obtemos a função seguinte.

```
def ler_escrever_min_max_csv (nome_ficheiro_entrada,
    nome_ficheiro_saida):
  """
  Requires: nome_ficheiro_entrada é uma string e representa um
    ficheiro de valores separados por ponto e vírgula; os
    valores neste ficheiro são números; nome_ficheiro_saida é
    uma string e descreve o nome do ficheiro CSV onde escrever
    o resultado.
  Ensures: Em cada linha do ficheiro de saída escreve o mínimo
    e o máximo dos valores constantes numa linha do ficheiro de
    entrada. Assim, o ficheiro de saída terá tantas linhas
    quantas o ficheiro de entrada.
  """
```

```
  linhas = ler_csv(nome_ficheiro_entrada)
  linhas_numeros = map(lambda l: map(int, l), linhas)
  linhas_maximos = map(lambda l: [min(l), max(l)],
    linhas_numeros)
  escrever_csv(nome_ficheiro_saida, linhas_maximos)
```

## Leitura de ficheiros CSV para listas de dicionários

Guardar uma linha de um ficheiro CSV numa lista pode não ser a melhor solução. Dada a lista `['Ana', 'Silva', '1998']`, nem sempre será óbvio o significado do número `'1998'`. Em vez de guardarmos a informação de uma linha numa lista podemos usar dicionários cujas chaves indicam o significado de cada campo: `{'Nome proprio':'Ana', 'Apelido':'Silva', 'Ano nascimento':'1998'}`.

Se pretendermos ler cada linha de um ficheiro CSV para um dicionário, usamos a função `DictReader`. A função recebe o nome de um ficheiro CSV e alguns parâmetros opcionais. Entre eles, `fieldnames` tem especial interesse: descreve uma lista com as chaves do dicionário a criar. Eis o padrão de utilização típico:

```
import csv
with open('ficheiro.csv', 'rU') as ficheiro_csv:
 leitor = csv.DictReader (ficheiro_csv, fieldnames=['Nome
    proprio', 'Apelido', 'Ano nascimento'], delimiter=';')
 for d in leitor
   # d é um dicionário com chaves 'Nome proprio', 'Apelido',
    'Ano nascimento'
```

Começamos por ler um ficheiro CSV para uma lista de dicionários (em vez de lista de listas, como fizémos na secção anterior). A função `ler_csv_sem_cabecalho` espera o nome de um ficheiro CSV e uma lista com os nomes das chaves dos dicionários a criar. Devolve uma lista de dicionários.

```
def ler_csv_sem_cabecalho(nome_ficheiro, cabecalho):
  """
  Requires: nome_ficheiro representa um ficheiro CSV.
    cabecalho é uma lista contendo os chaves dos dicionários a
    construir.
  Ensures: Devolve uma lista de dicionarios com o conteúdo do
    ficheiro.
  """
  with open(nome_ficheiro, 'rU') as ficheiro_csv:
    leitor = csv.DictReader(ficheiro_csv, fieldnames =
    cabecalho, delimiter = ';')
    return list(leitor)
```

Por exemplo:

```
>>> ler_csv_sem_cabecalho('pessoas.csv',
```

```
      ['Nome proprio', 'Apelido', 'Ano nascimento'])
[{'Ano nascimento': '1998', 'Nome proprio': 'Ana',
  'Apelido': 'Silva'},
 {'Ano nascimento': '2000', 'Nome proprio': 'Pedro',
  'Apelido': 'Castro'},
 {'Ano nascimento': '1999', 'Nome proprio': 'Eva',
  'Apelido': 'Sousa'},
 {'Ano nascimento': '1999', 'Nome proprio': 'Nuno',
  'Apelido': 'Martins'}]
```

Acontece que muitas tabelas vêm equipadas com um *cabeçalho*. Por cabeçalho entendemos a primeira linha do ficheiro, cujos campos descrevem o conteúdo das colunas que aparecem por baixo. O ficheiro com informação sobre pessoas que temos vindo a utilizar pode estar também equipado com um cabeçalho.

| Nome proprio | Apelido | Ano nascimento |
|---|---|---|
| **Ana** | Silva | 1998 |
| **Pedro** | Castro | 2000 |
| **Eva** | Sousa | 1999 |
| **Nuno** | Martins | 1999 |

Neste caso podemos retirar a informação sobre as colunas diretamente da primeira linha do ficheiro. A leitura de um destes ficheiros processa-se como anteriormente, com a diferença que não utilizamos o parâmetro `fieldnames` aquando da criação do leitor:

```
leitor = csv.DictReader (ficheiro_csv, delimiter = ';')
```

Chegados a este ponto poderíamos escrever uma outra função para ler ficheiros com cabeçalhos. Ficaríamos então com duas funções: ler com e ler sem cabeçalho. Acontece que as funções são iguais a menos do parâmetro opcional `fieldnames` aquando da criação do leitor. Em vez de repetir código, tornamos opcional o parâmetro `cabecalho` na função acima.

A função de leitura, que desta vez vamos chamar `ler_csv_dicionario`, espera dois parâmetros: o nome do ficheiro e a lista (opcional) o nome das chaves dos dicionários a criar.

```
def ler_csv_dicionario (nome_ficheiro, cabecalhos = None):
  """
  Requires: nome_ficheiro representa um ficheiro CSV;
    cabecalhos é uma lista.
  Ensures: Devolve uma lista de dicionarios com o conteúdo do
    ficheiro; as chaves do dicionário são lidas da primeira
    linha do ficheiro se cabecalhos == None. Caso contrário são
    tiradas da lista de cabecalhos.
  """
```

```
  with open(nome_ficheiro, 'rU') as ficheiro_csv:
    leitor = csv.DictReader(ficheiro_csv, cabecalhos,
    delimiter = ';')
    return list(leitor)
```

## Escrita de ficheiros CSV a partir de listas de dicionários

Quem sabe ler também sabe escrever (às vezes). Para escrever precisamos do nome do ficheiro e da lista de dicionários. Também precisamos de uma lista de strings, representando a ordem pela qual queremos que os elementos do dicionário apareçam no ficheiro.

Porque é que não extraímos esta informação dos próprios dicionários? Acontece que as entradas num dicionário Python não são ordenadas. Por exemplo:

```
>>> {'a':1, 'b':False} == {'b':False, 'a':1}
True
```

Assim sendo, não existe informação suficiente numa lista de dicionários para inferir a ordem pela qual os elementos de um dicionário devem ser escritos no ficheiro. Temos portanto de passar o cabeçalho como parâmetro, uma vez que, ao contrário da classe `DictReader`, o parâmetro dos nomes das chaves do `DictWriter` não é opcional.

```
def escrever_csv_dicionario(nome_ficheiro,
    lista_de_dicionarios, cabecalhos):
  """
  Requires: nome_ficheiro é uma string e descreve o nome de um
    ficheiro de escrita. lista_de_dicionarios é uma lista.
    cabecalho é uma lista contendo os chaves dos dicionários.
  Ensures: Escreve o cabeçalho na primeira linha do ficheiro e
    a lista_de_dicionarios nas linhas seguintes; valores
    separados por ponto e vírgula.
  """
  with open(nome_ficheiro, 'w') as ficheiro_csv:
     escritor = csv.DictWriter(ficheiro_csv, delimiter = ';',
    fieldnames = cabecalhos)
     escritor.writeheader()
     for linha in lista_de_dicionarios:
       escritor.writerow(linha)
```

Acabamos esta secção com uma pequena aplicação típica: ler, manipular, escrever. Começamos por ler um ficheiro CSV para uma lista de dicionários:

```
tabela = ler_csv_dicionario(nome_ficheiro_entrada)
```

Chegados a este ponto temos de juntar mais um campo à tabela: `'Idade'`. Para isso temos de converter o campo `'Ano de nascimento'` num número

inteiro: utilizamos a função `int`. Para obter o ano corrente usamos o pacote `datetime` que fornece a função `datetime.datetime.now().year`. O modo mais simples de obter o novo dicionário é alterar o dicionário original. Por exemplo:

```
d['Idade'] = datetime.datetime.now().year - int(d['Ano
    nascimento'])
```

E precisamos de fazer isto para todos os dicionários na lista. Quando utilizámos listas de listas recorrremos à função de ordem superior `map` e a uma expressão `lambda`. Acontece que não podemos utilizar instruções (como a acima) dentro de um `map`. Usamos então um ciclo `for`. O código completo fica assim:

```
import datetime
def juntar_idade (nome_ficheiro_entrada, nome_ficheiro_saida):
  """
  Requires: nome_ficheiro_entrada é uma string e descreve um
    ficheiro CSV com cabeçalho (1a linha). Um dos campos do
    cabeçalho é 'Ano nascimento'; nome_ficheiro_saida é uma
    string e descreve o nome de um ficheiro de escrita.
  Ensures: escreve no ficheiro de saída o conteúdo do ficheiro
    de entrada mas com uma coluna adicional contendo a 'Idade'.
  """
  tabela = ler_csv_dicionario(nome_ficheiro_entrada)
  for d in tabela:
    d['Idade'] = datetime.datetime.now().year - int(d['Ano
    nascimento'])
  escrever_csv_dicionario(nome_ficheiro_saida, tabela, ['Nome
    proprio', 'Apelido', 'Ano nascimento', 'Idade'])
```