Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

viernes, 1 de mayo de 2015

Operaciones con archivos CSV

Un archivo **CSV** (de Valores Separados por Comas) es un tipo de documento que representa los datos de forma parecida a una tabla, es decir, organizando la información en filas y columnas.

En un archivo CSV los datos de las diferentes columnas son separados, habitualmente, por un signo de puntuación (una coma, un punto y coma, etc.) u otro carácter que actúe como separador. Sin embargo, las diferentes filas suelen separarse por un salto de línea. Además, muchas veces los datos pueden ir precedidos por un encabezado con los nombres de campos o identificadores de columnas:

```
nombre,provincia,consumo
Alejandro,Sevilla,330
Carmen,Sevilla,320
Eugenia,Granada,280
...
```

Un archivo **CSV** almacena la información en un formato accesible que facilita el intercambio entre aplicaciones. Es frecuente en muchos programas que utilizan datos de contactos contar con una opción para exportar la información en dicho formato. También, las aplicaciones ofimáticas de hojas de cálculos (*Excel*, *Calc*, etc.) incorporan opciones para guardar la información en CSV.

Asimismo, estos programas ofrecen la posibilidad de elegir el carácter delimitador de campos y, opcionalmente, el carácter que se utilizará como delimitador de textos o datos. Los delimitadores de textos o de datos, a veces, son necesarios para indicar dónde comienza y termina un dato, sobre todo si dentro de éste pueden aparecer en un momento dado caracteres iguales al usado como separador de campo.

En el ejemplo siguiente el delimitador de textos que se utiliza es la doble comilla. Su uso estaría justificado porque en el dato domicilio aparece el carácter coma y éste coincide justamente con el carácter empleado como delimitador de campos. Si la información la tuviera que leer un programa, gracias a la doble comilla, se podría analizar correctamente donde empiezan y terminan los distintos datos:

```
"Domicilio","Municipio"
"Cultura, 24","La Rinconada"
"Antonio Machado, 3","Sevilla"
```

Afortunadamente, el lenguaje Python tiene un módulo denominado **csv** que permite leer y escribir archivos en estos formatos; y provee, también, varias funciones y clases para verificar la integridad de la información, analizar o crear dialectos, comprobar si los datos tienen un encabezamiento, etc.

Los **dialectos** son los distintos formatos que se aplican a los archivos CSV. El dialecto ***excel*** es el que se utiliza de forma predeterminada y establece los campos separados por comas, generalmente. El dialecto ***excel-tab*** utiliza el tabulador como separador de campo y en el dialecto ***unix*** todos los datos se representan entrecomillados.

Buscar

Buscar

Python para impacientes

Python
IPython
EasyGUI
Tkinter
JupyterLab
Numpy

Anexos

Guía urgente de MySQL
Guía rápida de SQLite3

Entradas + populares

Dar color a las salidas en la consola
En Python para dar color a las salidas en la consola (o en la terminal de texto) existen varias posibilidades. Hay un método basado ...

Instalación de Python, paso a paso
Instalación de Python 3.6 A finales de 2016 se produjo el lanzamiento de Python 3.6 . El propósito de esta entrada es mostrar, pas...

Añadir, consultar, modificar y suprimir elementos en Numpy
Acceder a los elementos de un array. [], [,], ... Acceder a un elemento de un array. Para acceder a un elemento se utiliz...

Variables de control en Tkinter
Variables de control Las variables de control son objetos especiales que se asocian a los widgets para almacenar sus valore...

Cálculo con arrays Numpy
Numpy ofrece todo lo necesario para obtener un buen rendimiento cuando se trata de hacer cálculos con arrays. Por como está concebid...

Operaciones con fechas y horas. Calendarios
Los módulos datetime y calendar amplían las posibilidades del módulo time que provee funciones para manipular expresiones de ti...

Tkinter: interfaces gráficas en Python
Introducción Con Python hay muchas posibilidades para programar una interfaz gráfica de usuario (GUI) pero Tkinter es fácil d...

Convertir, copiar, ordenar, unir y dividir arrays Numpy
Esta entrada trata sobre algunos métodos que se utilizan en Numpy para convertir listas en arrays y viceversa; para copiar arrays d...

Threading: programación con hilos (I)
En programación, la técnica que permite que una aplicación ejecute simultáneamente varias operaciones en el mismo espacio de proceso se...

El módulo random
El módulo random de la librería estándar de Python incluye un conjunto de funciones

Por otro lado, ampliar la potencia del módulo **csv** es posible si conocemos algunas técnicas, por ejemplo, para filtrar la información y/o mostrarla ordenada; y mediante el uso de otras funciones.

A continuación, se muestran ejemplos que utilizan este módulo de la biblioteca estándar que hará las delicias de muchos. Para probar los ejemplos es necesario crear en un editor de textos los siguientes archivos CSV:

datos.csv

```
nombre,provincia,consumo
Alejandro,Sevilla,330
Carmen,Sevilla,320
Eugenia,Granada,280
Antonio,Sevilla,340
Ana,Granada,290
Marta,Granada,230
Luis,Huelva,310
Manuel,Huelva,340
Francisco,Granada,320
```

salidat.csv

```
campo1,campo2
aaa,111
bbb,222
ccc,333
```

También, hay que importar el módulo **csv**, protagonista de esta entrada, y el módulo **operator** que se utiliza en algún ejemplo:

```
import csv, operator
```

Leer archivo CSV con reader()

```
# Leer el archivo 'datos.csv' con reader() y
# mostrar todos los registros, uno a uno:

with open('datos.csv') as csvarchivo:
    entrada = csv.reader(csvarchivo)
    for reg in entrada:
        print(reg)  # Cada línea se muestra como una lista de campos
```

Leer archivo CSV con reader() y realizar algunas operaciones básicas

```
# Leer el archivo 'datos.csv' con reader() y
# realizar algunas operaciones básicas:

csvarchivo = open('datos.csv')  # Abrir archivo csv
entrada = csv.reader(csvarchivo)  # Leer todos los registros
reg = next(entrada)  # Leer registro (lista)
print(reg)  # Mostrar registro
nombre, provincia, consumo = next(entrada)  # Leer campos
print(nombre, provincia, consumo)  # Mostrar campos
del nombre, provincia, consumo, entrada  # Borrar objetos
csvarchivo.close()  # Cerrar archivo
del csvarchivo  # Borrar objeto
```

Ordenar datos con sort(), sorted() e itemgetter()

La función **sort()** se utiliza para ordenar una lista en orden numérico, alfabético, etc.:

```
lista = ["d", "a", "e", "b", "c"]
lista.sort()
print(lista)  # ['a', 'b', 'c', 'd', 'e']
```

Por otro lado, disponemos de la función **sorted()** que permite utilizar otros criterios de ordenación, ampliando las posibilidades de sort():

sorted(*objeto_a_ordenar, key=función, reverse=True|False*)

La función **sorted()** devuelve un objeto con los elementos ordenados. Si se utiliza el argumento **key** se aplicará a cada elemento la función indicada. Para obtener los elementos ordenados con

que permiten obtener de distintos modos números a...

Archivo

mayo 2015 (3)

python.org

pypi.org

Sitios

- ActivePython
- Anaconda
- Bpython
- Django
- Flask
- Ipython
- IronPython
- Matplotlib
- MicroPython
- Numpy
- Pandas
- Pillow
- PortablePython
- PyBrain
- PyCharm
- PyDev
- PyGame
- Pypi
- PyPy
- Pyramid
- Python.org
- PyTorch
- SciPy.org
- Spyder
- Tensorflow
- TurboGears

orden inverso hay que incluir el argumento **reverse** con el valor ***True***. Ejemplos:

```
# Crear listas para ordenarlas con la función sorted()

lista1 = ['ABCDEF', 'ABCEFGHIJ', 'ABC', 'ABCD']
lista2 = ['10', '30', '20', '4']

# Ordenar lista1 por la longitud de sus elementos:

lista1 = sorted(lista1, key=len)
print('lista1 ordenada por longitud de cadenas: ', lista1)

# Ordenar lista1 por la longitud de sus elementos en orden inverso:

lista1 = sorted(lista1, key=len, reverse=True)
print('lista1 ordenada por longitud de cadenas inverso:', lista1)

# Ordenar lista2 por el valor numérico de sus elementos:

lista2 = sorted(lista2, key=int, reverse=False)
print('lista2 ordenada por valor numérico:', lista2)
```

Por otra parte, utilizar la función **itemgetter()** del módulo **operator** con la función **sorted()** permite ordenar una lista de tuplas (o de registros) por el índice de uno de sus campos (el índice del primer campo es el 0, del segundo el 1, etc.):

```
# Declarar una lista con cuatro tuplas de dos campos cada una:

lista = [('cccc', 4444), ('d', 1), ('aa', 22), ('bbb', 333)]

# Ordenar lista por el segundo campo de cada tupla (índice 1):

listaord = sorted(lista, key=operator.itemgetter(1), reverse=False)
print('lista ordenada por campo2:', listaord)

# Ordenar lista por primer campo de cada tupla (índice 0),
# orden inverso:

listaord = sorted(lista, key=operator.itemgetter(0), reverse=True)
print('lista ordenada inversa por campo1:', listaord)

# Ordenar alfabéticamente por el primer campo:

listaord = sorted(lista)
print('lista ordenada alfabéticamente:', listaord)
```

Leer archivo CSV con reader() y ordenar salida por un campo

```
# Leer un archivo con reader() y mostrarlo ordenado por tercer
# campo con la función itemgetter() del módulo operator:

csvarchivo = csv.reader(open('datos.csv'))
listaordenada = sorted(csvarchivo,
                       key=operator.itemgetter(2),
                       reverse=False)
print(listaordenada)
```

Leer archivo CSV con DictReader() y filtrar salida

```
# Leer un archivo csv como lista de diccionarios con DictReader() y
# mostrar sólo datos de algunas columnas:

with open('datos.csv') as csvarchivo:
    entrada = csv.DictReader(csvarchivo)
    for reg in entrada:
        print(reg['provincia'], reg['consumo'])
```

Leer archivo CSV con DictReader() y consultar propiedades

```
# Mostrar lista de diccionarios a partir CSV y
# consultar número de líneas (registros), dialecto y campos:
```

```
csvarchivo = open('datos.csv')
entrada = csv.DictReader(csvarchivo)
listadicc = list(entrada)  # Obtener lista de diccionarios
print('Lista:', listadicc)  # Mostrar lista de diccionarios
print('Líneas:', entrada.line_num)  # Obtener número de registros
print('Dialecto:', entrada.dialect)  # Obtener dialecto
print('Campos:', entrada.fieldnames)  # Obtener nombre de campos
del entrada, listadicc
csvarchivo.close()
del csvarchivo
```

Leer archivo CSV con DictReader() y ordenar salida por nombre de campo

```
# Obtener lista ordenada descendente por el campo 'consumo'
# con la función itemgetter() del módulo operator.

csvarchivo = open('datos.csv')
entrada = csv.DictReader(csvarchivo)
listadicc = list(entrada)  # Obtener lista de diccionarios
listadiccord = sorted(listadicc,
                      key=operator.itemgetter('consumo'),
                      reverse=True)
for registro in listadiccord:
    print(registro)

del entrada, listadicc, listadiccord
csvarchivo.close()
del csvarchivo
```

Escribir archivo CSV con writer() y writerow(). Quoting

La opción de **quoting** establece el tipo de entrecomillado que se aplicará a un archivo **CSV**. Existen cuatro posibilidades: ***QUOTE_ALL*** para entrecomillar todos los campos; ***QUOTE_MINIMAL*** para entrecomillar sólo los campos que sean necesarios para mantener la integridad de los datos; ***QUOTE_NONE*** para no entrecomillar nada y ***QUOTE_NONNUMERIC*** para entrecomillar sólo aquellos campos que no sean numéricos.

```
# Leer con DictReader y escribir datos en otro csv si se
# cumple condición.
# En el archivo 'salida.csv' se escribirán todos los datos
# entrecomillaods con dobles comillas y separados entre sí
# con el carácter "|".

with open('datos.csv') as csvarchivo:
    entrada = csv.DictReader(csvarchivo)
    csvsalida = open('salida.csv', 'w', newline='')
    salida = csv.writer(csvsalida, delimiter='|',
                        quotechar='"',
                        quoting=csv.QUOTE_ALL)
    print('Escribiendo archivo "salida.csv"...')
    print('Dialecto:', entrada.dialect, '...')
    for reg in entrada:
        if reg['provincia'] != 'Huelva':
            salida.writerow([reg['nom'],
                             reg['cons']])  # Escribir registro

    print('El proceso de escritura ha terminado.')

del entrada, salida, reg
csvsalida.close()
del csvsalida
```

Escribir archivo CSV con writer() y writerows()

```
# Escribir todas las tuplas de una lista con writerows()

datos = [('aaa', 111), ('bbb', 222), ('ccc', 333)]
csvsalida = open('salidat.csv', 'w', newline='')
salida = csv.writer(csvsalida)
salida.writerow(['campo1', 'campo2'])
salida.writerows(datos)
```

```
del salida
csvsalida.close()
```

Leer archivo CSV con reader(), skipinitialspace y strict

```
# Leer archivo ignorando o no los espacios que existan después
# de los delimitadores de campos

# Si la propiedad skipinitialspace es True se ignorarán
# los espacios
# Si la propiedad strict es True se producirá un error si el
# archivo csv no es válido

# Leer archivos sin ignorar espacios

with open('archivo.csv') as csvarchivo:
    entrada = csv.reader(csvarchivo,
                         skipinitialspace=False,
                         strict=True)
    for reg in entrada:
        print(reg)

# Leer archivos ignorando espacios

with open('archivo.csv') as csvarchivo:
    entrada = csv.reader(csvarchivo,
                         skipinitialspace=True,
                         strict=True)
    for reg in entrada:
        print(reg)
```

Escribir archivo CSV con DictWriter() capturando excepciones

```
try:
    salidacsv = open('campos.csv', 'w')
    campos = ['Campo1', 'Campo2']
    salida = csv.DictWriter(salidacsv, fieldnames=campos)
    salida.writeheader()
    for indice in range(6):
        salida.writerow({ 'Campo1':indice+1,
                          'Campo2':chr(ord('a') + indice)})

    print('Se ha creado el archivo "campos.csv"')

finally:
    salidacsv.close()
```

Leer archivo CSV con reader() de un dialecto determinado

```
# Leer archivo csv con dialecto 'unix'

with open('salida.csv') as csvarchivo:
    entrada = csv.reader(csvarchivo,
                         dialect='unix',
                         delimiter='|')
    for reg in entrada:
        print(reg)
```

Crear dialecto y leer archivo CSV

```
# Crear nuevo dialecto 'personal' y abrir archivo usándolo.

csv.register_dialect('personal',
                     delimiter='|',
                     quotechar='"',
                     quoting=csv.QUOTE_ALL)
with open('salida.csv') as csvarchivo:
    entrada = csv.reader(csvarchivo, dialect='personal')
    for reg in entrada:
        print(reg)
```

Listar dialectos disponibles

```
# Listar dialectos

print(csv.list_dialects())
```

Obtener información de un dialecto

```
# Obtener información del dialecto "personal"

dialecto = csv.get_dialect('personal')
print('delimiter', dialecto.delimiter)
print('skipinitialspace', dialecto.skipinitialspace)
print('doublequote', dialecto.doublequote)
print('quoting', dialecto.quoting)
print('quotechar', dialecto.quotechar)
print('lineterminator', dialecto.lineterminator)
```

Suprimir dialecto

```
# Suprimir dialecto "personal"

csv.unregister_dialect('personal')
print(csv.list_dialects())  # Listar dialectos después
```

Deducir con Sniffer() el dialecto de un archivo csv

```
with open('salida.csv') as csvarchivo:
    dialecto = csv.Sniffer().sniff(csvarchivo.read(48))
    csvarchivo.seek(0)
    print("Dialecto:", dialecto)
    csvarchivo.seek(0)
    entrada = csv.reader(csvarchivo, dialecto)
    for reg in entrada:
        print(reg)
```

Deducir con Sniffer() si un archivo tiene encabezado

```
# El archivo salida.csv no tiene encabezado

csvarchivo1 = open('salida.csv')
cabecera1 = csv.Sniffer().has_header(csvarchivo1.read(48))
print("\ncsvarchivo1 (salida.csv) cabecera:", cabecera1)
csvarchivo1.seek(0)
entrada = csv.reader(csvarchivo1, "unix")
for reg in entrada:
    print(reg)

csvarchivo1.close()

# El archivo salidat.csv sí tiene encabezado

csvarchivo2 = open('salidat.csv')
cabecera2 = csv.Sniffer().has_header(csvarchivo2.read(76))
print("\ncsvarchivo2 (salidat.csv) cabecera:", cabecera2)
csvarchivo2.seek(0)
entrada = csv.reader(csvarchivo2, "excel")
for reg in entrada:
    print(reg)

csvarchivo2.close()
```

Mostrar/Establecer límite de tamaño de campo a analizar

```
# Para establecer un nuevo límite: field_size_limit(NuevoLímite)

print(csv.field_size_limit())
```

Ir al índice del tutorial de Python

Publicado por Pherkad en 9:04

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua

2014-2020 | Alejandro Suárez Lamadrid y Antonio Suárez Jiménez, Andalucía - España
. Tema Sencillo. Con la tecnología de Blogger.