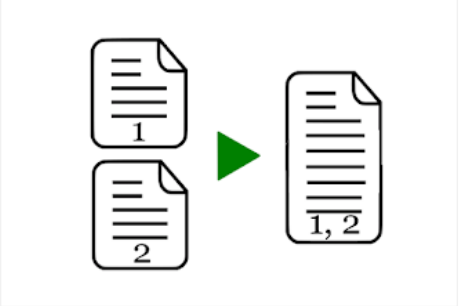
Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

jueves, 9 de junio de 2016

Fileinput: procesar múltiples archivos fácilmente



El módulo **fileinput** consta de una clase y varias funciones que permiten procesar datos procedentes de varios archivos, incluidos como elementos de una lista o como argumentos de un *script*. Esta característica es idónea para leer y procesar todas las líneas de un conjunto de archivos como si se tratara de uno solo.

También, el módulo permite procesar múltiples líneas de datos introducidas en la entrada estándar; reescribir archivos modificando el contenido original y obtener los datos directamente de archivos comprimidos.

A continuación, varios ejemplos para mostrar algunas aplicaciones del módulo. Los ejemplos utilizan tres archivos de texto con el siguiente contenido:

cine1.txt

```
Título;Dirección;País
El clan;Pablo Trapero;Argentina
Papeles en el Viento;Juan Taratuto;Argentina
El club;Pablo Larraín;Chile
El botón de nácar;Patricio Guzmán;Chile
```

cine2.txt

```
Título;Dirección;País
Gente de bien;Franco Lolli;Colombia
Los hongos;Oscar Ruiz Navia;Colombia
Manos sucias;Josef Kubota Wladyka;Colombia
Amama;Asier Altuna;España
Un día perfecto;Fernando León de Aranoa;España
El desconocido;Dani de la Torre;España
```

cine3.txt

```
Título;Dirección;País
600 millas;Gabriel Ripstein;México
Archivo 253;Abe Rosenberg;México
El cumple de la abuela;Javier Colinas;México
Lo que lleva el río;Mario Crespo;Venezuela
```

Procesar lista de archivos con fileinput.input()

En el primer ejemplo con la función **fileinput.input()** se recorren los tres archivos de una lista y se muestran todas sus líneas en la salida estándar (normalmente, la pantalla). También, se antepone el número de línea leída obtenido con la función **fileinput.lineno()**.

fileinput.input(files=None, mode='r'**)**

Por defecto, el modo de apertura de los archivos es **'r'**, que permite la lectura de los archivos de

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...

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

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...

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...

Tkinter: Tipos de ventanas
Ventanas de aplicación y de diálogos En la entrada anterior tratamos los distintos gestores de geometría que se utilizan para di...

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

tipo texto. Sólo en ese caso el argumento **mode** puede omitirse. En otro caso, para que sea posible la lectura de los archivos binarios, es necesario establecer **mode** con el valor '**rb**'.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#

import fileinput
import sys, glob

cines = ["cine1.txt", "cine2.txt", "cine3.txt"]
for linea in fileinput.input(cines):
    ln = str(fileinput.lineno())
    sys.stdout.write(ln + " -> " + linea)

'''
Salida:
1 -> Título;Dirección;País
2 -> El clan;Pablo Trapero;Argentina
3 -> Papeles en el Viento;Juan Taratuto;Argentina
4 -> El club;Pablo Larraín;Chile
5 -> El botón de nácar;Patricio Guzmán;Chile
6 -> Título;Dirección;País
7 -> Gente de bien;Franco Lolli;Colombia
8 -> Los hongos;Oscar Ruiz Navia;Colombia
9 -> Manos sucias;Josef Kubota Wladyka;Colombia
10 -> Amama;Asier Altuna;España
11 -> Un día perfecto;Fernando León de Aranoa;España
12 -> El desconocido;Dani de la Torre;España
13 -> Título;Dirección;País
14 -> 600 millas;Gabriel Ripstein;México
15 -> Archivo 253;Abe Rosenberg;México
16 -> El cumple de la abuela;Javier Colinas;México
17 -> Lo que lleva el río;Mario Crespo;Venezuela
'''
```

Los archivos anteriores tienen en su primera línea una cabecera que describe el contenido de los mismos: ***Título;Dirección;País***. Si sólo interesan los datos y se quiere omitir esta línea:

```
for linea in fileinput.input(cines):
    if not fileinput.isfirstline():
        sys.stdout.write(linea)

'''
Salida:
El clan;Pablo Trapero;Argentina
Papeles en el Viento;Juan Taratuto;Argentina
El club;Pablo Larraín;Chile
El botón de nácar;Patricio Guzmán;Chile
Gente de bien;Franco Lolli;Colombia
Los hongos;Oscar Ruiz Navia;Colombia
Manos sucias;Josef Kubota Wladyka;Colombia
Amama;Asier Altuna;España
Un día perfecto;Fernando León de Aranoa;España
El desconocido;Dani de la Torre;España
600 millas;Gabriel Ripstein;México
Archivo 253;Abe Rosenberg;México
El cumple de la abuela;Javier Colinas;México
Lo que lleva el río;Mario Crespo;Venezuela
'''
```

Procesar con funciones que ayudan a controlar el flujo

Cuando se utilice la función **fileinput.input()**, que está basada en la clase **fileinput.FileInput()**, se podrán utilizar varias funciones que devuelven información que se puede usar para controlar el proceso. También, el módulo incluye una función para pasar, en cualquier momento, de un archivo al siguiente y, otra, para forzar la finalización del proceso:

- **fileinput.filename()**: Devuelve nombre del archivo que se está leyendo en un momento dado. Antes de leer la primera línea devuelve **None**.
- **fileinput.fileno()**: Devuelve número (descriptor) de archivo. Antes de leer la primera línea y con el proceso entre dos archivos devuelve **-1**.
- **fileinput.lineno()**: Devuelve número de línea leída del total de líneas de todos los archivos. Antes de leer la primera línea devuelve **0** y después de la última línea, devuelve el número de dicha línea.
- **fileinput.filelineno()**: Devuelve el número de línea leída del archivo actual. Antes de leer la primera línea devuelve **0** y después de la última línea leída de un archivo,

que permiten obtener de distintos modos números a...

Archivo

junio 2016 (2)

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

devuelve el número de dicha línea.

- **fileinput.isfirstline()**: Devuelve **True** si la línea leída es la primera y **False** si no lo es.
- **fileinput.isstdin()**: Devuelve **True** si la última línea se ha leído desde la entrada estándar (**sys.stdin**), de lo contrario devuelve **False**.
- **fileinput.nextfile()**: Cierra el archivo actual y salta al siguiente archivo si fuera posible. No se puede utilizar para omitir el primer archivo.
- **fileinput.close()**: Finaliza el proceso

En el siguiente ejemplo se utilizan algunas funciones para obtener información de los archivos y para controlar el proceso. El resultado muestra la información obtenida y la segunda línea de cada archivo (primera línea de datos), excepto del tercer archivo:

```
for linea in fileinput.input(cines):
    if not fileinput.isfirstline():
        ar = fileinput.filename()
        an = str(fileinput.fileno())
        al = str(fileinput.filelineno())
        nl = str(fileinput.lineno())
        linea = linea.rstrip()
        print(ar, an, al, nl, linea)
        fileinput.nextfile()
    if fileinput.filename()== 'cine3.txt':
        fileinput.close()

'''
Salida:
cine1.txt 3 2 2 El clan;Pablo Trapero;Argentina
cine2.txt 3 2 4 Gente de bien;Franco Lolli;Colombia
'''
```

Procesar lista de archivos obtenida con glob.glob()

Cuando los nombres de archivos siguen un patrón es posible utilizar el módulo glob para obtener del directorio la lista de archivos, como en el siguiente ejemplo. Además, en este caso se utiliza la función **split()** para dividir cada línea en tres partes equivalentes a sus datos para poder filtrar la información (en este ejemplos se muestran las líneas de las películas de "**Colombia**"):

```
archivos = glob.glob("cine*.txt")
archivos.sort()
for linea in fileinput.input(archivos):
    if not fileinput.isfirstline():
        linea = linea.rstrip()
        pais = linea.split(";")
        if pais[2] == "Colombia":
            print(linea)

'''
Salida:
Gente de bien;Franco Lolli;Colombia
Los hongos;Oscar Ruiz Navia;Colombia
Manos sucias;Josef Kubota Wladyka;Colombia
'''
```

Procesar archivos incluidos como argumentos de un script

Asimismo, los nombres de los archivos a procesar se pueden incluir como argumentos de un script Python.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#

# script: peliculas.py

import fileinput, sys

cines = sys.argv[1:]
print("Archivos:", cines)
for linea in fileinput.input(cines):
    if not fileinput.isfirstline():
        sys.stdout.write(linea)

'''
Salida:
Archivos: ['cine1.txt', 'cine2.txt', 'cine3.txt']
```

```
El clan;Pablo Trapero;Argentina
Papeles en el Viento;Juan Taratuto;Argentina
El club;Pablo Larraín;Chile
El botón de nácar;Patricio Guzmán;Chile
Gente de bien;Franco Lolli;Colombia
Los hongos;Oscar Ruiz Navia;Colombia
Manos sucias;Josef Kubota Wladyka;Colombia
Amama;Asier Altuna;España
Un día perfecto;Fernando León de Aranoa;España
El desconocido;Dani de la Torre;España
600 millas;Gabriel Ripstein;México
Archivo 253;Abe Rosenberg;México
El cumple de la abuela;Javier Colinas;México
Lo que lleva el río;Mario Crespo;Venezuela
'''
```

Para ejecutar el script "**peliculas.py**" desde la línea de comandos:

$ python peliculas.py *cine1.txt cine2.txt cine3.txt*

Procesar múltiples líneas de la entrada estándar (stdin)

Como se comentó al comienzo la función **fileinput.input()** puede procesar múltiples líneas de datos introducidas en la entrada estándar.

Con el siguiente ejemplo se pueden introducir varias líneas utilizando la entrada estándar (teclado, normalmente). Cada vez que se complete una línea presionar la tecla [**Return**] para avanzar a la siguiente. Para finalizar la entrada presionar [**Ctrl+D**]. Después, de todas las líneas introducidas sólo se mostrarán las tres primeras. Cuando se alcance la número tres terminará el proceso.

```
print("Introducir varias líneas de texto (Finalizar - CTRL+D):")
for linea in fileinput.input():
    print(linea.rstrip())
    if fileinput.lineno() == 3:
        fileinput.close()
```

Procesar lista de archivos con la sentencia with

La función **fileinput.input()** también se puede utilizar con la sentencia **with**:

```
with fileinput.input(["cine1.txt", "cine2.txt"]) as archivo:
    for linea in archivo:
        if not fileinput.isfirstline():
            sys.stdout.write(linea)

'''
Salida:
El clan;Pablo Trapero;Argentina
Papeles en el Viento;Juan Taratuto;Argentina
El club;Pablo Larraín;Chile
El botón de nácar;Patricio Guzmán;Chile
Gente de bien;Franco Lolli;Colombia
Los hongos;Oscar Ruiz Navia;Colombia
Manos sucias;Josef Kubota Wladyka;Colombia
Amama;Asier Altuna;España
Un día perfecto;Fernando León de Aranoa;España
El desconocido;Dani de la Torre;España
'''
```

Procesar reescribiendo lista de archivos en la salida estándar

El argumento **inplace** (*in situ*) de la función **fileinput.input()** con el valor **True** permite reescribir los archivos de entrada utilizando la salida estándar. El contenido previo de cada archivo se puede salvar asignando la extensión de los archivos de respaldo al argumento **backup**.

fileinput.input(files=None, inplace=False, backup='', mode='r'**)**

En el ejemplo que sigue se van se utilizan dos archivos nuevos de texto que contienen datos de varias salas de cine, tanto el nombre de la sala como el aforo que tiene. En el proceso se utiliza la función **fileinput.lineno()** para obtener el número de línea. Luego, se utiliza **print()** para anteponer dicho número a cada línea; que en lugar de ser mostrada en la salida estándar (pantalla) será redirigida al archivo original modificando su contenido, que previamente es salvado en otro archivo con la extensión **.bck**:

sala1.txt

Cinema Paradise (300)
Cines Nervión (400)

sala2.txt

Cine Apolo (260)
Corona Center (180)
Cine Bécquer (300)

```
for linea in fileinput.input(["sala1.txt", "sala2.txt"],
                             inplace=True, backup='.bck'):
    nl = str(fileinput.lineno())
    linea = nl + ": " + linea.rstrip('\n')
    print(linea)

'''
Salida sala1.txt:
1: Cinema Paradise (300)
2: Cines Nervión (400)

Salida sala2.txt:
3: Cine Apolo (260)
4: Corona Center (180)
5: Cine Bécquer (300)

Los contenidos originales se salvan en los archivos
sala1.txt.bck y sala2.txt.bck
'''
```

La clase fileinput.FileInput()

La clase **fileinput.FileInput()** permite crear objetos **FileInput** y sus métodos tienen los mismos nombres y funcionalidades que las funciones utilizadas con **fileinput.input()**. Además, cuenta con el método **readline()** que devuelve la siguiente línea leída en la entrada y el método **__getitem__()**.

class fileinput.FileInput(files=None, inplace=False, backup='', mode='r', openhook=None**)**

En relación al argumento **openhook** (también disponible en **fileinput.input()**) comentar que permite controlar la forma de abrir los archivos. Para ello, se puede asignar una función a medida con dos argumentos (el nombre del archivo y su modo de apertura) que devuelva un objeto de tipo fichero abierto. Por otra parte, el módulo incorpora de fábrica la función **fileinput.hook_encoded()** para abrir los archivos utilizando una codificación determinada y la función **fileinput.hook_compressed()** para abrir archivos comprimidos.

Para finalizar, varios ejemplos que utilizan objetos de la clase **fileinput.FileInput()**.

Procesar lista de archivos con la clase FileInput()

En el siguiente ejemplo se crea un objeto **FileInput** a partir de una lista de archivos y se muestra el contenido numerando las líneas:

```
cines = ["cine1.txt", "cine2.txt", "cine3.txt"]
contador = 0
objfi = fileinput.FileInput(cines)
for linea in objfi:
    if not objfi.isfirstline():
        contador += 1
        sys.stdout.write(str(contador) + ": " + linea)
```

Procesar lista de archivos con la clase FileInput() y readline()

Crea un objeto **FileInput** y después utiliza el método **readline()** para leer las líneas de los archivos de entrada:

```
objfi=fileinput.FileInput(["cine1.txt", "cine2.txt"])
while True:
    linea=objfi.readline()
    if not linea: break
    if not objfi.isfirstline():
        sys.stdout.write(linea)
```

Procesar archivos con clase FileInput() con una codificación específica

Crea un objeto **FileInput** abriendo los archivos con la codificación "**utf-8**" mediante la función **fileinput.hook_encoded**

```
print()
objfi = fileinput.FileInput(["sala1.txt"],
                            openhook=fileinput.hook_encoded("utf-8"))
for linea in objfi:
    sys.stdout.write(linea)
```

Procesar archivo comprimido con clase FileInput()

Crea un objeto **FileInput** obteniendo los archivos "sala1.txt" y "sala2.txt" del archivo comprimido "**salacines.gz**", mediante la función **fileinput.hook_compressed**. Las líneas de los archivos son devueltas en binario pero son convertidas a cadenas de texto con la función **decode()**:

```
objfi = fileinput.FileInput(["salacines.gz"],
                            openhook=fileinput.hook_compressed)
for linea in objfi:
    sys.stdout.write(linea.decode("utf-8"))
```

Para crear un archivo comprimido **.gz** en **GNU/Linux** con los dos archivos que contienen los datos de las salas, ejecutar desde la consola los siguientes comandos:

$ gzip -c sala1.txt **>** salacines.gz
$ gzip -c sala2.txt **>>** salacines.gz

Relacionado:

- Operaciones con archivos
- Tipos de cadenas: Unicode, Byte y Bytearray

Ir al índice del tutorial de Python

Publicado por Pherkad en 12:26

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua