martes, 26 de abril de 2016

Tempfile: archivos y directorios temporales

El módulo tempfile

A veces, los programas necesitan crear archivos para almacenar datos que son usados en procesos intermedios, que con posterioridad cuando ya no son necesarios, son suprimidos. Python provee una serie de funciones que facilitan las operaciones con archivos pero, concretamente, para este tipo de archivos temporales cuenta con el módulo **tempfile** que ofrece algunas ventajas interesantes al programador, en este ámbito.

Entre las ventajas del módulo se encuentran las siguientes:

- Permite crear archivos y directorios temporales posibilitando el acceso a los directorios temporales que ofrecen los distintos sistemas operativos.
- Crea automáticamente y con seguridad los nombres de los archivos y directorios temporales pudiéndose establecer un prefijo y/o un sufijo para definir dichos nombres, para favorecer con ello su localización y tratamiento.
- Proporciona funciones que suprimen automáticamente los archivos y/o directorios temporales cuando ya no son necesarios. Estas funciones son **TemporaryFile**(), **NamedTemporaryFile**(), **TemporaryDirectory**() y **SpooledTemporaryFile**(). Además, para contar con mayor control sobre el borrado de los archivos y directorios temporales ofrece, alternativamente, las funciones **mkstemp**() y **mkdtemp**(), que requieren un borrado manual.

TemporaryFile(). Crea archivo temporal (sin nombre)

La función **TemporaryFile**() crea un archivo temporal de forma segura y devuelve un objeto (sin nombre) para ser utilizado como espacio de almacenamiento temporal y será suprimido tan pronto como sea cerrado.

*tempfile.*TemporaryFile*(mode='w+b', buffering=None, encoding=None, newline=None, suffix=None, prefix=None, dir=None)*

Por defecto, la apertura del archivo temporal es "**w+b**" (lectura/escritura en modo binario) para un procesamiento consistente entre distintos sistemas operativos, aunque también es frecuente el modo "**w+t**" (lectura/escritura para textos).

Los argumentos **suffix** y **prefix**, si se usan, se corresponden con la cadena de sufijo y/o de prefijo que se empleará en la construcción de los nombres de los archivos temporales. El argumento **dir** se utiliza para especificar un directorio para ubicar los archivos y en caso de que no se especifique ninguno se utilizará el directorio temporal del sistema, que suele ser el recomendado.

En cuanto a los argumentos **buffering**, **encoding** y **newline**, tienen el mismo uso que con la función open(). Son para establecer la política de almacenamiento en el búfer, especificar un nombre de codificación de caracteres y fijar cómo será el salto de líneas en los archivos creados en modo texto.

El siguiente ejemplo muestra cómo crear un archivo temporal con la función **TemporaryFile**(), en modo binario:

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...

Threading: programación con hilos (I)
En programación, la técnica que permite que una aplicación ejecute

```
import tempfile

# Crea un archivo temporal en modo binario
temporal1 = tempfile.TemporaryFile()

# Muestra información del objeto creado
print(temporal1)  # _io .bufferedrandom="" name="4"
print(type(temporal1))  # class io.bufferedrandom=""
print(temporal1.name)  # identificador del archivo, 4

# Escribe en el fichero temporal
temporal1.write(b'archivos temporales\ncon Python')

# Sitúa el puntero al comienzo del archivo
temporal1.seek(0)

# Lee archivo temporal desde su comienzo
lectura1 = temporal1.read()

# Muestra la información leída
print(lectura1)

# Cierra y elimina el archivo temporal
temporal1.close()
```

En el ejemplo que sigue se muestra cómo crear un archivo temporal en modo texto, con la posibilidad de capturar los errores que pudieran producirse al acceder al sistema de archivos:

```
import tempfile

# Crea un archivo temporal en modo texto
temporal2 = tempfile.TemporaryFile(mode='w+t')
print(temporal2.name)  # identificador del archivo, 4

# Captura posibles errores de acceso al sistema de archvios
try:
    # Escribe tres líneas en el archivo temporal
    temporal2.writelines(['Linea 1\n', 'Linea 2\n', 'Linea 3\n'])

    # Sitúa el puntero al comienzo del archivo
    temporal2.seek(0)

    # Lee y muestra todas las líneas del archivo temporal
    for linea in temporal2:
        print(linea.rstrip())

finally:
    # Cierra y elimina el archivo temporal
    temporal2.close()
```

NamedTemporaryFile(). Crea archivo temporal (con nombre)

La función **NamedTemporaryFile**() crea un archivo temporal al que asignará un nombre construido automáticamente, teniendo en cuenta en el caso de que se hayan utilizado, los argumentos **suffix** y **prefix**.

*tempfile.*NamedTemporaryFile*(mode='w+b', buffering=None, encoding=None, newline=None, suffix=None, prefix=None, dir=None, delete=True)*

Hay situaciones en las que es conveniente utilizar esta función que da nombre a los archivos temporales, como por ejemplo, cuando la información de estos archivos es utilizada por varios procesos diferentes. En ese caso la mejor referencia a utilizar será el propio nombre del archivo temporal para poder abrirlo, que puede obtenerse mediante el atributo **name**. Los archivos igualmente serán suprimidos cuando sean cerrados, excepto si al argumento **delete** se le asigna el valor **False**.

En cuanto a lo demás, esta función actúa exactamente igual que **TemporaryFile**().

Ejemplo de uso de la función **NamedTemporaryFile()**:

```
import os, tempfile

# Crea archivo temporal
temporal3 = tempfile.NamedTemporaryFile()

# Muestra nombre y ruta del archivo temporal creado
```

simultáneamente varias operaciones en el mismo espacio de proceso se...

Archivo

abril 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

```
print(temporal3, temporal3.name)

# Escribe en el archivo temporal
temporal3.write(b'Temporal')

# Comprueba si existe el archivo temporal
if os.path.exists(temporal3.name):
    print("El archivo temporal existe")

# Cierra y suprime el archivo temporal
temporal3.close()

# Comprueba si se ha borrado el archivo temporal
if not os.path.exists(temporal3.name):
    print("El archivo temporal ya no existe")
```

SpooledTemporaryFile(). Crea archivo estableciendo tamaño máximo del búfer de memoria

La función **SpooledTemporaryFile**() se utiliza para crear archivos temporales con un matiz que lo diferencia de la función **TemporaryFile**(). Incorpora el argumento **max_size** para establecer el tamaño máximo de memoria que se usará para almacenar el archivo temporal.

Cuando el límite de **max_size** sea superado, o bien, cuando sea llamado el metodo **fileno**() los datos pasarán a escribirse directamente al disco.

*tempfile.*SpooledTemporaryFile*(max_size=0, mode='w+b', buffering=None, encoding=None, newline=None, suffix=None, prefix=None, dir=None)*

El siguiente ejemplo crea archivo temporal limitando la memoria a 1024 bytes:

```
# Crea archivo temporal fijando búfer en 1024 bytes
temporal4 = tempfile.SpooledTemporaryFile(max_size=1024)

# Escribe en el fichero temporal
for ciclo in range(0,20):
    temporal4.write((b"*" * 50) + b"\n")

# Sitúa el puntero al comienzo del archivo temporal
temporal4.seek(0)

# Lee el archivo temporal
lectura4 = temporal4.read()

# Convierte la cadena leida de byte a string con codificación utf-8
print(lectura4.decode("utf-8"))

# Muesta idenfiticador del archivo cuando supere el límite de
# max_size. En caso contrario, el valor devuelto será None porque
# el archivo sólo se alojará en la memoria. Para probar otros resultados
# modificar el valor de max_size, por ejemplo, reduciéndolo
print(temporal4.name)

# Muestra la longitud de la cadena leída
print(len(lectura4))

# Cierra y elimina el archivo temporal
temporal4.close()
```

TemporaryDirectory(). Crea directorio temporal

La función **TemporaryDirectory**() crea un directorio temporal cuyo nombre puede ser recuperado accediendo al valor del atributo **name**. El objeto resultante se puede utilizar como un gestor de contexto que termina suprimiendo el directorio y todo su contenido.

En el siguiente ejemplo se crea un directorio temporal utilizando la declaración "**with ... as**" como gestor de contexto:

```
# Crea directorio temporal con gestor de contexto
with tempfile.TemporaryDirectory() as dirtemporal6:
    print('Directorio temporal', dirtemporal6)
# Al finalizar, tanto directorio como contenido se borran
```

A continuación, un ejemplo en el que se une la creación de un directorio temporal con un archivo temporal y que finaliza con la supresión de ambos:

```
# Crea directorio temporal
tempdir7 = tempfile.TemporaryDirectory()

# Crea archivo temporal en el directorio anterior
temporal7 = tempfile.NamedTemporaryFile(dir=tempdir7.name)

# Muestra nombres de directorio y archivo temporales
print('Directorio temporal:', tempdir7.name)
print('Archivo temporal...:', temporal7.name)

# Escribe en el archivo
for ciclo in range(0,50000):
    temporal7.write(b"=" * 20)

# Borra archivo temporal
del temporal7

# Borra directorio temporal
del tempdir7
```

mkstemp(). Crea archivo temporal sin borrado desatendido

La función **mkstemp**() crea con seguridad un archivo temporal en la que la operación de borrado queda a elección del usuario.

tempfile.mkstemp*(suffix=None, prefix=None, dir=None, text=False)*

Devuelve una tupla que contiene un identificador (como el que devuelve la función **os.open**()) y la ruta de acceso absoluta al archivo temporal.

A partir de Python 3.5 los argumentos **suffix**, **prefix** y **dir** pueden expresarse como **bytes**.

En el siguiente ejemplo se crea un archivo temporal que es borrado "manualmente" con el método **os.remove**().

```
import os, tempfile

id8, archivotemp8 = tempfile.mkstemp(prefix="TempDirApp", text=True)

print("idenficador:", id8)
print("archivo:", archivotemp8)
os.remove(archivotemp8)
if not os.path.exists(archivotemp8):
    print("El archivo {} ha sido borrado\n".format(archivotemp8))
```

mkdtemp(). Crea directorio temporal sin borrado desatendido

La función **mkdtemp**() crea con seguridad un directorio temporal en la que la operación de borrado no es desatendida. El usuario, cuando lo desee, tendrá que realizarla "manualmente".

tempfile.mkdtemp*(suffix=None, prefix=None, dir=None)*

La función **mkdtemp**() devuelve la ruta absoluta del directorio temporal

A partir de Python 3.5 los argumentos **suffix**, **prefix** y **dir** pueden expresarse como **bytes**.

A continuación, un ejemplo de uso de la función **mkdtemp**():

```
import os, tempfile

# Crea directorio temporal con el prefijo 'Aplic'
tempdir9 = tempfile.mkdtemp(prefix='Aplic')

# Muestra directorio creado
print(tempdir9)

# Borra directorio tempora si existe
if os.path.exists(tempdir9):
    os.rmdir(tempdir9)
    print('Directorio suprimido')
```

gettempdir()/gettempdirb(). Obtiene directorio temporal

Las funciones **gettempdir**() y **gettempdirb**() devuelven la ruta del directorio temporal del sistema (/tmp, c:\temp, etc.). La primera función la devuelve en formato **str** y la segunda expresada como **bytes**.

```
print("Dir temporal del sistema", tempfile.gettempdir())
print("Dir temporal (en bytes)", tempfile.gettempdirb())  # Python +3.5
```

gettempprefix()/gettempprefixb(). Obtiene prefijos de nombres

Las funciones **gettempprefix**() y **gettempprefixb**() devuelven los prefijos que se aplican a los nombres de los archivos temporales, en el momento de su creación. La primera función la devuelve como **str** y la segunda como **bytes**.

```
print("Prefijo", tempfile.gettempprefix())
print("Prefijo (en bytes)", tempfile.gettempprefixb())  # Python +3.5
```

La variable tempdir

La variable global **tempdir** almacena el directorio temporal predeterminado del sistema, que es el mismo que devuelven las funciones **gettempdir**() y **gettempdirb**(). Este directorio puede cambiarse asignándose una nueva ruta a dicha variable, aunque no se recomienda.

En el supuesto de que se modifique la ruta, será la que se utilice en todas las funciones de este modulo como predeterminada para el argumento **dir**.

```
tempfile.tempdir = '/home/usuario/temp'
print("Directorio temporal", tempfile.gettempdir())
print("Directorio temporal", tempfile.tempdir)
```

Ir al índice del tutorial de Python

Publicado por Pherkad en 17:14

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua