Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

sábado, 26 de septiembre de 2015

Explorando directorios con listdir, walk y scandir

Las funciones **listdir()**, **walk()** y **scandir()** pertenecen al módulo **os** de la librería estándar de Python y se utilizan para acceder al sistema de ficheros y explorar sus directorios.

Obtener las entradas de un directorio: listdir()

La función **os.listdir()** devuelve una lista que contiene los nombres de las entradas (archivos y directorios) del directorio indicado (**path**). La lista no sigue ningún tipo de orden y no se incluyen las entradas '.' y '..'.

```
os.listdir(path='.')
```

El argumento **path** puede ser del tipo **str** o **bytes**.

En el siguiente ejemplo se leen los archivos (excepto ejecutables) que se encuentren en el directorio del script. De cada archivo se obtiene su tamaño, la fecha de último acceso y la fecha de la última modificación. Al final, se muestra el número archivos encontrados y el tamaño total que ocupan en kbytes.

```
import os
from datetime import datetime

ruta_app = os.getcwd()  # obtiene ruta del script
contenido = os.listdir(ruta_app)  # obtiene lista con archivos/dir
total = 0
archivos = 0
formato = '%d-%m-%y %H:%M:%S'  # establece formato de fecha-hora
linea = '-' * 40

for elemento in contenido:
    archivo = ruta_app + os.sep + elemento
    if not os.access(archivo, os.X_OK) and os.path.isfile(archivo):
        archivos += 1
        estado = os.stat(archivo)  # obtiene estado del archivo
        tamaño = estado.st_size  # obtiene de estado el tamaño

        # Obtiene del estado fechas de último acceso/modificación
        # Como los valores de las fechas-horas vienen expresados
        # en segundos se convierten a tipo datetime.

        ult_acceso = datetime.fromtimestamp(estado.st_atime)
        modificado = datetime.fromtimestamp(estado.st_mtime)

        # Se aplica el formato establecido de fecha y hora

        ult_acceso = ult_acceso.strftime(formato)
        modificado = modificado.strftime(formato)
```

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...

El módulo random
El módulo random de la librería estándar de Python incluye un conjunto de funciones que permiten obtener de distintos modos números a...

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

```
        # Se acumulan tamaños y se muestra info de cada archivo

        total += tamaño
        print(linea)
        print('archivo      :', elemento)
        print('modificado   :', modificado)
        print('último acceso:', ult_acceso)
        print('tamaño (Kb)  :', round(tamaño/1024, 1))

print(linea)
print('Núm. archivos:', archivos)
print('Total (kb)   :', round(total/1024, 1))
```

Con **if not os.access(archivo, os.X_OK)** se verifica que el archivo no es ejecutable. El resto de modos que se pueden utilizar con **os.access()** son:

- **os.F_OK** para comprobar que es posible acceder a un archivo,
- **os.R_OK** para saber si el archivo se puede leer y
- **os.W_OK** para conocer si además se permite la escritura.

Con la función **os.path.isfile(archivo)** se comprueba si el elemento es un archivo.

Por cierto, aunque hemos utilizado **os.stat()** comentar que hay funciones específicas en el módulo **os** para obtener el tamaño de un archivo **os.path.getsize()** y las fechas de último acceso **os.path.getatime()** y de última modificación **os.path.getmtime()**.

Para finalizar, los métodos de la clase **stat** son los siguientes (dependiendo del sistema algunos no estarán disponibles):

- **st_size**: tamaño en bytes.
- **st_mode**: tipo de archivo y bits de permisos.
- **st_ino**: número de inodo.
- **st_dev**: identificador del dispositivo.
- **st_uid**: identificador del usuario propietario.
- **st_gid**: identificador del grupo propietario.
- **st_atime**: fecha-hora del último acceso (en segundos).
- **st_mtime**: fecha-hora de la última modificación (en segundos).
- **st_ctime**: fecha-hora ultimo cambio (unix) o creación (win).
- **st_atime_ns**, **st_mtime_ns** y **st_ctime_ns** (idem. expresado en nanoseg).
- **st_blocks**: número de bloques de 512 bytes asignados.
- **st_blksize**: tamaño de bloque preferido por sistema.
- **st_rdev**: tipo de dispositivo si un dispositivo inode.
- **st_flags**: banderas definidas por usuario.
- **st_gen**: Número fichero generado.
- **st_birthtime**: tiempo de creación del archivo.
- **st_rsize**: tamaño real del archivo.
- **st_creator**: creador del archivo.
- **st_type**: tipo de archivo.
- **st_file_attributes**: atributos.

Obtener recursivamente las entradas de un directorio: walk()

La función **os.walk()** se utiliza para leer recursivamente los archivos y directorios del directorio indicado (**top**) excluyendo las entradas '.' y '..'.

```
os.walk(top, topdown=True,onerror=None,followlinks=False)
```

> Hasta Python 3.4 **os.walk()** llama a la función **os.listdir()** para cumplir con su propósito. A partir de Python 3.5 las llamadas se realizan a **os.scandir()**, ganándose velocidad en el acceso por reducir, a su vez, las llamadas a **os.stat()**.

En cada lectura la función **os.walk()** devuelve una tupla con la siguiente información del directorio leído:

- **root**: directorio leído
- **dirs**: lista de directorios existentes en el directorio leído
- **files**: lista de archivos existentes en el directorio leído

El argumento **topdown** con el valor **True** indica que la lectura se hace comenzando por el directorio de menor profundidad; y con **False** en el directorio de mayor profundidad.

El argumento opcional **onerror** con el valor por defecto **None** se utiliza para ignorar los errores

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

Archivo

septiembre 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

de **os.listdir()**. Para la gestión de estos errores es posible utilizar una función definida por el usuario.

El argumento opcional **followlinks** con el valor por defecto **False** indica que no se resuelvan los enlaces simbólicos que se encuentren en la exploración.

El siguiente ejemplo es similar al anterior pero utiliza la lectura recursiva que permite **os.walk()** para leer todos los archivos no ejecutables contenidos en el directorio inicial y en los subdirectorios existentes, mostrando al final el número de archivos leídos y el tamaño total en kbytes.

```
import os
from datetime import datetime

ruta_app = os.getcwd()
total = 0
num_archivos = 0
formato = '%d-%m-%y %H:%M:%S'
linea = '-' * 60

for ruta, directorios, archivos in os.walk(ruta_app, topdown=True):
    print('\nruta       :', ruta)
    for elemento in archivos:
        num_archivos += 1
        archivo = ruta + os.sep + elemento
        estado = os.stat(archivo)
        tamaño = estado.st_size
        ult_acceso = datetime.fromtimestamp(estado.st_atime)
        modificado = datetime.fromtimestamp(estado.st_mtime)
        ult_acceso = ult_acceso.strftime(formato)
        modificado = modificado.strftime(formato)
        total += tamaño
        print(linea)
        print('archivo      :', elemento)
        print('modificado   :', modificado)
        print('último acceso:', ult_acceso)
        print('tamaño (Kb)  :', round(tamaño/1024, 1))

print(linea)
print('Núm. archivos:', num_archivos)
print('Total (kb)   :', round(total/1024, 1))
```

Obtener iterador con las entradas de un directorio: scandir()

Para concluir, la función **os.scandir()** devuelve un iterador basado en la clase **DirEntry** que contiene información relacionada con las entradas (archivos y directorios) del directorio indicado (**path**). La información no sigue ningún tipo de orden predeterminado y no se incluyen, si existen, las entradas '.' y '..'.

```
os.scandir(path='.')
```

La clase **DirEntry** cuenta con métodos que permiten acceder a información relativa a cada entrada:

- **name**: nombre del archivo o directorio leído.
- **path**: ruta completa del archivo o directorio leído.
- **inode()**: devuelve número de inodo de la entrada.
- **is_dir(*, follow_symlinks=True)**: devuelve **True** si es directorio
- **is_file(*, follow_symlinks=True)**: devuelve **True** si es archivo
- **is_symlink()**: devuelve **True** si es un enlace simbólico.
- **stat(*, follow_symlinks=True)**: devuelve estado de la entrada

La función **os.scandir()** se propone en Python 3.5 (PEP0471) como alternativa a **os.listdir()** al mejorar la velocidad de acceso al sistema de ficheros por realizar menos llamadas a **os.stat()**. Además, dependiendo del tipo de sistema y del tamaño de los archivos la velocidad con **os.walk()** puede ser más rápida de 2 a 20 veces.

El siguiente ejemplo es casi una réplica del primero pero está basado en la función **os.scandir()**.

```
import os
from datetime import datetime

ruta_app = os.getcwd()
contenido = os.listdir(ruta_app)
total = 0
```

```
archivos = 0
formato = '%d-%m-%y %H:%M:%S'
linea = '-' * 40

contenido = os.scandir(ruta_app)
for elemento in contenido:
    if not os.access(elemento.path, os.X_OK) and elemento.is_file():
        archivos += 1
        estado = elemento.stat()
        tamaño = estado.st_size
        ult_acceso = datetime.fromtimestamp(estado.st_atime)
        modificado = datetime.fromtimestamp(estado.st_mtime)
        ult_acceso = ult_acceso.strftime(formato)
        modificado = modificado.strftime(formato)
        total += tamaño
        print(linea)
        print('archivo      :', elemento.name)
        print('permisos     :', estado.st_mode)
        print('modificado   :', modificado)
        print('último acceso:', ult_acceso)
        print('tamaño (Kb)  :', round(tamaño/1024, 1))

print(linea)
print('Núm. archivos:', archivos)
print('Total (kb)   :', round(total/1024, 1))
```

Relacionado: Filtrando archivos y directorios con glob y fnmatch

Ir al índice del tutorial de Python

Publicado por Pherkad en 15:15

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua