Organización de Archivos

Antes de continuar trabajando con Archivos les recomiendo, si es que no lo hicieron todavía, desactivar la opción de Ocultar extensiones de archivos en Windows.
Inicio ▸ Panel de Control ▸ Apariencia y Personalización ▸ Opciones del explorador de Archivos
En la pestaña Ver, dentro de la lista de opciones Avanzadas destildar Ocultar las extensiones para tipos de archivos conocidos.

El modulo shutil

Este módulo tiene funciones que nos permiten copiar, mover, renombrar, y eliminar archivos con nuestros programas de Python.

Copiando Archivos y Carpetas

Llamando a la función shutil.copy(origen, destino) se copiará el archivo en el origen a la dirección destino que designemos en el argumento. Si el destino tiene un nombre de archivo este se usará para el nombre del nuevo archivo. Esta función también retorna una cadena con la dirección del archivo que creado.

```
>>> import shutil, os
>>> from pathlib import Path
>>> d = Path.home()
>>> shutil.copy(d / 'spam.txt', d / 'copia_spam')
>>> shutil.copy(d / 'huevos.txt', d / 'una_carpeta/huevos2.txt')
```

Por otro lado, con shutil.copytree() podemos copiar todo el contenido de una carpeta, incluyendo las subcarpetas que se encuentren dentro. Llamamos shutil.copytree(origen, destino) de la misma manera que a shutil.copy(), definiendo un directorio origen y un directorio de destino para los archivos que se crearan.

```
>>> import shutil, os
>>> from pathlib import Path
>>> d = Path.home()
>>> shutil.copytree(d / 'spam', d / 'spam_backup')
```

Moviendo y Renombrando Carpetas y Archivos

Llamando a la función shutil.move(origen, destino) podemos mover un archivo o carpeta a otra dirección del disco. Si el destino apunta a una carpeta el archivo retendrá su nombre original.

```
>>> import shutil
>>> shutil.move('H:\\cerdo.txt', 'H:\\huevos\\nuevo_cerdo.txt')
>>> shutil.move('H:\\cerdo.txt', 'H:\\huevos')
```

En el caso de que la carpeta destino no exista, cuando se copie un archivo se le cambiará el nombre al de esa carpeta inexistente. Por este motive hay que ser cuidadosos al utilizar la función move().

Borrando archivos y carpetas de manera Permanente

Para borrar archivos y carpetas tenemos las siguientes funciones a nuestra disposición:

- Llamando a os.unlink(dirección) se borrara el archivo en esa dirección
- Llamando a os.rmdir(dirección) se borrará la carpeta en esa dirección. La carpeta debe estar vacía.
- Llamando a shutil.rmtree(dirección) se borrará la carpeta y todos los archivos y carpetas que contenga.

Cuando trabajamos con alguna de estas funciones siempre conviene primero comentarlas mientras estamos probando el programa. De esa manera podemos prevenir eliminar archivos importantes por error. Por ejemplo en el siguiente programa se quiere borrar todos los archivos con extensión .txt pero hay un error de tipeo y borrará los archivos .rxt en su lugar.

```
import os
from pathlib import Path
for filename in Path.home().glob('*.rxt'):
  os.unlink(filename)
```

Es decir si hubiéramos tenido archivos importantes del tipo .rxt los habríamos eliminado por error. Para evitar esto prueben sus programas con esas funciones comentadas primero e imprimiendo el nombre del archivo que intentaban eliminar. De esta manera:

```
import os
from pathlib import Path
for filename in Path.home().glob('*.rxt'):
  #os.unlink(filename)
  print(filename)
```

Recorriendo un directorio

Supongamos queremos renombrar todos los archivos y subcarpetas en una carpeta. Es decir, queremos recorrer un directorio, modificando cada uno de los elementos. Existe una función en Python para estos casos, os.walk().

```
import os

for folderName, subfolders, filenames in os.walk('H:\\deliciosos'):
  print('CARPETA ACTUAL ' + folderName)
  for subfolder in subfolders:
    print('SUBCARPETAS DE ' + folderName + ': ' + subfolder)
  for filename in filenames:
    print('ARCHIVOS DENTRO ' + folderName + ': '+ filename)
  print('')
```

La función os.walk() recibe una cadena con la dirección del directorio que queramos recorrer, y devolverá una serie de elementos con 3 valores:

- Una cadena con el nombre de la carpeta que se está recorriendo.
- Una cadena con las subcarpetas.
- Un cadena con los archivos en la carpeta

Comprimiendo archivos con el módulo zipfile

Comprimir archivos reduce su tamaño y hace que sea más fácil transferirlos a través de internet. Los archivos ZIP son carpetas que contienen estos archivos y/o carpetas comprimidas.
Para poder crear y abrir (o extraer) archivos ZIP con Python necesitamos importar primero el módulo zipfile.

Leyendo archivos ZIP

Para leer un archivo ZIP primero hay que crear un objeto ZipFile. Esto objetos son similares a los objetos File con los que venimos trabajando y creando utilizando la función open(). Para crear un objeto ZipFile hay que llamar a la función zipfile.ZipFile() pasándole como argumento una cadena con la dirección del archivo .ZIP que queremos abrir. Ejemplo:

```
>>> import zipfile, os
>>> from pathlib import Path
>>> d = Path.home()
>>> ejemploZip = zipfile.ZipFile(d / 'ejemplo.zip')
>>> ejemploZip.namelist()
>>> spamInfo = ejemploZip.getinfo('ejemplo.mp4')
>>> spamInfo.file_size
>>> spamInfo.compress_size
>>> ejemploZip.close()
```

Los objetos ZipFile tienen un método llamado **namelist()** que devuelve una lista de cadenas con todos los archivos y carpetas dentro del archivo. Estas cadenas se las podemos pasar a otro método llamado **getinfo()** para obtener un **objeto ZipInfo** de ese archivo en particular. Los objetos ZipInfo tienen un par de atributos con los que podemos conseguir su tamaño original y su tamaño comprimido, que son **file_size** y **compress_size**.
Un **objeto ZipFile** representa al archivo ZIP en su totalidad mientras que un ZipInfo solo contiene información sobre uno de los archivos dentro.

Extrayendo archivos de un archivo ZIP

El método extractall() de los objetos ZipFile extrae todos los archivos y carpetas de un archivo ZIP dentro del CWD. Ejemplo:

```
>>> import zipfile, os
>>> from pathlib import Path
>>> d = Path.home()
>>> ejemploZip = zipfile.ZipFile(d / 'ejemplo.zip')
>>> ejemploZip.extractall()
>>> ejemploZip.close()
```

También se le puede pasar una dirección a extractall() para descomprimir los archivos en otra ubicación. Si esta carpeta no existe, se creará.

El otro método se llama extract() y extraerá un solo archivo del archivo ZIP. Ejemplo:

```
>>> ejemploZip.extract('ejemplo.mp4')
>>> ejemploZip.extract('ejemplo.mp4', 'H:\\deliciosos\\waffles\\belgas')
>>> ejemploZip.close()
```

La cadena que le pasamos a extract() debe coincidir con alguna de las cadenas que devuelve el método namelist(). Opcionalmente le podemos pasar un segundo argumento para especificar en qué carpeta extraer el archivo en lugar del CWD. Al igual que antes, si la carpeta no existe se creará. Por último, extract devuelve la dirección absoluta del archivo extraído.

Creando y añadiendo a archivos ZIP

Para crear nuestro propio archivo ZIP, primero hay que abrir un objeto ZipFile en modo escritura, pasándole el argumento 'w'. (similar a los modos de la función open())
Cuando le pasamos una dirección al método write() de un objeto ZipFile, Python comprimirá el archivo en esa dirección y lo agregará al archivo ZIP. El primer argumento del método write() es una cadena con el nombre del archivo a agregar. El segundo argumento es el tipo de compresión a utilizar, es decir que algoritmo de compresión utilizará la computadora para comprimir el archivo. Por ahora utilizaremos el tipo zipfile.ZIP_DEFLATED, que funciona para la mayoría de tipos de archivos. Ejemplo:

```
>>> import zipfile
>>> nuevoZip = zipfile.ZipFile('new.zip', 'w')
>>> nuevoZip.write('spam.txt', compress_type=zipfile.ZIP_DEFLATED)
>>> nuevoZip.close()
```

Al igual que con el modo de escritura 'w' de open(), en este caso también se sobrescribirá el contenido del archivo ZIP. Si queremos agregar archivos tendremos que utilizar el modo agregar o append con el argumento 'a'.