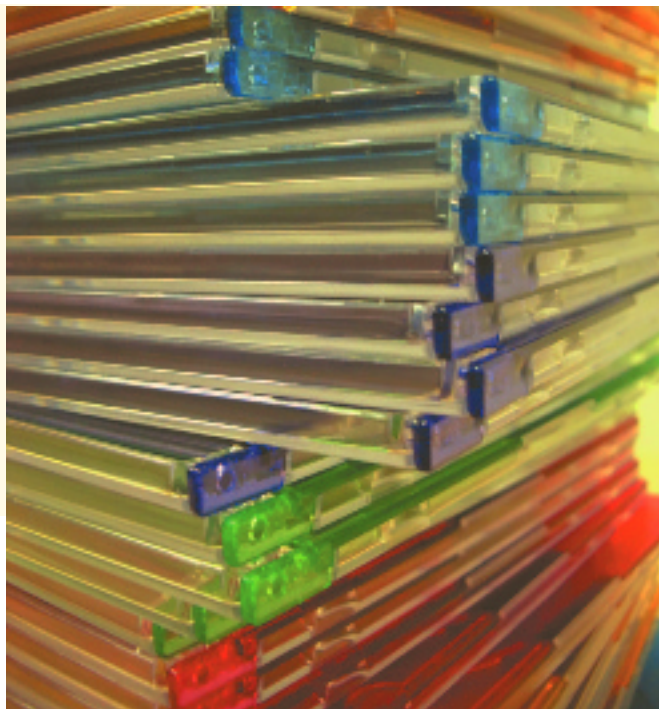


Manejo básico de ficheros

Crea un álbum de imágenes

Siguiendo con nuestro paseo por Python vamos a seguir viendo características básicas, y concretamente en este artículo, el tratamiento de ficheros.

POR JOSÉ MARÍA RUÍZ Y JOSÉ PEDRO ORANTES



En el último artículo vislumbramos algo sobre cómo trabajar con objetos en Python. Fue algo muy simple, pero que ya nos daba la posibilidad de organizar nuestro código en torno a ello. Python hace un uso extensivo de los objetos en sus APIs, y especialmente del control de errores mediante excepciones, lo que nos da la posibilidad de lanzar excepciones cuando algo va mal. Una excepción es un mensaje que podemos capturar cuando se ejecuta cierta función o método y aparece un error de algún tipo. Normalmente controlamos estos errores mediante el valor devuelto por la función (como por ejemplo en C). Esta técnica es engorrosa, pero al igual que todo, tiene sus virtudes y sus desventajas. Sin embargo, Python hace uso de las excepciones en su lugar.

Cuando una función genera una excepción decimos que **eleva una excepción**. Es muy normal tener que controlar las excepciones en las operaciones que realizamos con recursos que pueden no estar disponibles. Por eso las vamos a ver, puesto que en este artículo vamos a trabajar con archivos y conexiones a Internet.

Vamos a crear un objeto que gestione un recurso que puede no estar disponible. En este caso el objeto gestiona una variable (vease Listado 1).

Alguien puede crear un objeto de la clase *obj_variable* y llamar al método *set_variable(23)*, pero ¿cómo puede estar seguro de que la variable *var* tiene el valor 23 después de la llamada? Puede que *var* no fuese 0, porque otra llamada anterior ya podría haberla asignado. Lo único que podríamos hacer es llamar a *reset_variable()* y así asegurarnos de que nuestro valor sea asignado, pero entonces destruiríamos el valor anterior y no sabríamos qué podría pasar.

Por lo tanto, necesitamos un mecanismo de comunicación para darle a conocer al usuario que esa variable ya está asignada. Esto lo podemos hacer con las excepciones.

En el Listado 2 aparece una clase que hereda de la clase *Exception* (veremos los mecanismos de herencia en otro artículo dedicado a los objetos) llamada *Var_Asignada*. Cuando en la clase *obj_variable* intentamos asignar un valor a la variable *var* y ésta no es 0, entonces se dispara, se eleva, la excepción *Var_Asignada*. Si no controlamos la porción de código en la que se encuentra *set_variable()* y aparece una excepción, el programa se detendrá y acabará.

La idea detrás de las excepciones es que es posible tratarlas y evitar males

mayores, pudiendo en ocasiones incluso recuperarnos de ellas. Para ello está la estructura *try-except*. Rodeamos el código que puede disparar excepciones con esta estructura (Vease Listado 3).

A partir de ahora, y hasta que no expliquemos con más profundidad el tema de las excepciones, cuando digamos que una función genera una excepción, significará que ese código deberá estar rodeado con una estructura *try-except*.

Trabajo con Ficheros

Ya que hemos conseguido cierta soltura con los conceptos de objetos en Python, ahora vamos a ver como se manejan los accesos a ficheros en él.

Para acceder a un fichero primero necesitamos crear un objeto *file*. El objeto *file* es parte de la librería base de Python, así que no es necesario importar ninguna librería externa.

```
>>> archivo = file('texto.txt')
```

file abre los ficheros en modo de sólo lectura por definición. Eso significa que si el fichero no existe obtendremos un error. Para verificar si el fichero existe podemos usar la función *exists()* de la librería *os.path*.

```
LI:>>> import os.path >>>
os.path.exists('texto.txt') True >>>
```

```
os.path.exists('algo-peludo-y-feo.txt')
False
```

Por lo tanto, si vamos a abrir un fichero, será importante asegurarse de que ya existe. Si en lugar de leer, lo que queremos es crear el fichero, deberemos invocar al constructor de *file* con los parámetros:

```
>>> archivo = file('texto.txt',
, 'w')
```

Este segundo parámetro opcional nos permite definir el tipo de acceso que vamos a realizar al fichero. Tenemos varias posibilidades: podemos leer (*r*), escribir (*w*), añadir al final del fichero (*a*) y también tener acceso de lectura/escritura (*r+w*). Disponemos también del modificador *b* para indicar acceso binario. Python considera a los ficheros como ficheros de texto por definición. Vamos a ver todas las combinaciones en el Listado 4.

Con cualquiera de estas llamadas tendríamos en *archivo*, si todo ha ido bien, un objeto que gestiona el archivo indicado. Es entonces cuando podemos operar sobre él. Las operaciones más típicas son las de leer desde el archivo y escribir en él. Para ello el objeto *file* dispone de los métodos *read()*, *readline()*, *write()* y *writeline()*. Todos ellos operan con cadenas de caracteres: *readline()* y *writeline()* trabajan con líneas de texto (acabadas en retorno de carro), mientras que *read()* y *write()* lo hacen con cadenas sin restricciones.

Lo que vemos en el Listado 5 son algunas manipulaciones sobre un fichero. Lo

Listado 1

```
01 >>> class obj_variable:
02 ...     __init__(this):
03 ...         var = 0
04 ...
05 ...     set_variable(this,
06 ...                 valor):
07 ...         if (var == 0):
08 ...             var = valor
09 ...
10 ...     reset_variable(this):
11 ...         var = 0
12 >>>
```

primero que tenemos que hacer es crear el fichero, para lo cual lo abrimos en modo de escritura, 'w', que creará el fichero o truncará el existente (lo borrará para crearlo de nuevo, aunque si lo que queremos es añadir al final, usaremos 'a'). Posteriormente escribimos en él una cadena con un retorno de carro en mitad (para hacer nuestras pruebas) y cerramos el fichero. Es importante cerrar los ficheros cuando dejemos de usarlos, pero en este caso la razón para cerrarlo es que vamos a volver a abrirlo en modo de lectura.

Ahora volvemos a abrir el fichero en modo de lectura y leemos 4 bytes que almacenamos en la variable *cadena*. Cuando leemos con *read()* avanzamos en el fichero, esa es la razón de que el posterior *readline()* lea la cadena "mundo\n" en lugar de "Hola mundo". También vemos que se para en el retorno de carro en lugar de continuar. El segundo *readline()* ya nos permite leer la cadena "Adios mundo".

Pero... ¿qué ocurriría si en una de las lecturas nos encontrásemos con el fin de fichero? En el caso de que leyésemos una cadena con el fin de fichero (EOF), al final simplemente nos quedaríamos con la cadena hasta el EOF. En cambio, si sólo leemos el EOF, entonces obtenemos una *null*. Esto es importante para comprobar que hemos acabado con el fichero. Así, un bucle que escriba por pantalla el contenido del fichero, comprobaría en cada vuelta si la cadena que devuelve *readline()* es *null*.

Ahora que ya sabemos crear archivos, tenemos que aprender a borrarlos. Esto se realiza mediante la función *remove()* de la librería *os*. Esta función acepta la ruta de un fichero y lo borra. Si en lugar de un fichero le pasamos un directorio elevará una excepción "OSError".

```
>>> import os
>>> os.remove(texto.txt)
>>>
```

Directorios y Sistema de Ficheros

Con estos pocos métodos tenemos ya a nuestro alcance la manipulación básica de ficheros. Pero vamos a necesitar para nuestro programa la posibilidad de crear directorios. ¿Cómo lo haremos? Pues

mediante la función *mkdir()*, que acepta una cadena y crea un directorio con ese nombre. Si queremos crear un directorio que esté dentro de otros directorios también nuevos, tenemos que usar *makedirs()*. Ambas funciones pertenecen al módulo *os* por lo que para usarlas tendremos que hacer:

```
>>> import os
>>> os.mkdir('uno')
>>> os.makedirs('dos/tres')
```

Para borrar esos directorios usaremos la función *rmdir()* y *removedirs()*. La primera borra un directorio, mientras que la segunda borra una ruta de directorios. Vamos a ver esto con más detenimiento.

```
>>> os.rmdir('uno')
>>> os.removedirs('dos/tres')
```

Listado 2

```
01 >>> class
02 ...     Var_Asignada(Exception):
03 ...         """ Excepción que se
04 ...             dispara al intentar asignar
05 ...             una variable ya
06 ...             asignada en obj_variable"""
07 ...     pass
08 ...
09 >>> class obj_variable:
10 ...     """ Administra una
11 ...         variable """
12 ...     def __init__(self):
13 ...         self.var = 0
14 ...
15 ...     def set_variable(self,
16 ...                     valor):
17 ...         if (self.var ==
18 ...             0):
19 ...             self.var =
20 ...                 valor
21 ...         else:
22 ...             raise
23 ...                 Var_Asignada
24 ...
25 ...     def
26 ...         reset_variable(self):
27 ...             self.var = 0
28 ...
29 ...
30 >>>
31 >>> a = obj_variable()
32 >>> a.set_variable(12)
33 >>> a.set_variable(34)
```

Listado 3

```

01 >>> try:
02 ...     set_variable(12)
03 ...     set_variable(34)
04 ... except:
05 ...     print "ERROR: Se ha
06 ...     intentado asignar"
07 ...     print "un valor a una
08 ...     variable ya asignada"
09 ...
10 >>>

```

`rmdir()` borrará el directorio “uno”, que no contiene ningún otro objeto en su interior (ni directorios, ni ficheros). En caso de tenerlo, la llamada devolvería un error. La función `removedirs()` comenzaría a borrar desde el directorio que está más a la derecha de la ruta (“tres”) hacia el que está más a la izquierda (“dos”). Pero imaginemos que dentro de “dos” también hay un directorio “cuatro”. Entonces se borraría el directorio “tres”, y cuando la función fuese a borrar el directorio “dos” se encontraría con que no puede porque existe dentro de él un directorio llamado “cuatro” y pararía. En cuanto a `removedirs()`, este comando aplica `rmdir()` de derecha a izquierda en la ruta.

Imaginemos ahora que necesitamos cambiar el directorio en el que estamos trabajando. En el momento de arrancar el programa ese directorio es el que alberga el programa o bien es el directorio desde el que se ejecutó, pero no siempre es en el que queremos que el programa trabaje. Cualquier referencia a un fichero será tomando como base el directorio de trabajo, en caso contrario tendríamos que especificar la ruta de trabajo. Para poder cambiar el directorio de trabajo, el módulo `os` tiene la función `chdir()`. Si invocamos dentro de nuestro programa:

```
>>> os.chdir('/tmp')
```

Desde ese momento cualquier referencia a un fichero será direccionada a “/tmp”.

Ahora podemos:

- Abrir, cerrar, modificar un fichero.

Listado 5

```

01 >>> archivo =
02 ...     file('/tmp/texto.txt','w')
03 >>> archivo.write("Hola
04 ...     mundo\nAdios mundo")
05 >>> archivo.close()
06 >>>
07 >>> archivo =
08 ...     file('/tmp/texto.txt','r')
09 >>> cadena = archivo.read(4)
10 >>> print cadena
11 'Hola'
12 >>> archivo =
13 ...     file('/tmp/texto.txt','r')
14 >>> cadena = archivo.readline()
15 >>> print cadena
16 'Adios mundo'
17 >>> archivo.close()

```

- Crear, eliminar un directorio.
- Cambiar el directorio de trabajo.

Python y la web

Python posee gran cantidad de librerías para trabajar con recursos de Internet. De hecho Zope [1], un servidor de aplicaciones con gran éxito, está creado en Python y hace uso de todas sus características. Mailman [2] o Bittorrent [3] son también buenos ejemplos.

Debido a su flexibilidad, Python es usado como lenguaje de implementación para multitud de aplicaciones de red así como aplicaciones distribuidas. Por eso, no es de extrañar que Python suela ser el lenguaje en el que se implementan muchas de las más novedosas tecnologías de red.

En este apartado vamos a comenzar con lo básico. Queremos traer un recurso de la red a nuestra máquina, y para ello emplearemos una URL [4] del estilo `http://www.algunaweb.algo/imagen.jpg`. Pero primero necesitamos crear una conexión con el servidor.

Vamos a emplear para ello la librería `httplib` que viene con Python. Esta librería nos permite establecer una conexión con un servidor http y mandarle comandos http. Los comandos http son simples, y de todos ellos, de momento sólo nos interesa uno, el comando `GET`.

Listado 4

```

01 >>> archivo =
02 ...     file('texto.txt','r')
03 >>> archivo =
04 ...     file('texto.txt','w')
05 >>> archivo =
06 ...     file('texto.txt','a')
07 >>> archivo =
08 ...     file('texto.txt','r+w')
09 >>> archivo =
10 ...     file('texto.txt','r+b')
11 >>> archivo =
12 ...     file('texto.txt','rb')

```

Cuando accedemos a un servidor http, por ejemplo para ver una página web, lo que hacemos es pedirle objetos. Esto se hace mediante el comando `GET <objeto>`. Por ejemplo, si queremos la página `index.html` de la web `http://www.python.org`, primero conectamos con el servidor http de `http://www.python.org` y después, una vez conectados, le enviamos al servidor http el comando `GET index.html`. En ese momento el servidor nos devuelve por el mismo canal el contenido del archivo `index.html`.

Dicho así parece muy fácil, pero es una tarea que en un lenguaje de más bajo nivel requeriría gran cantidad de librerías y control de errores. En Python vamos a tener muchas menos dificultades. Siguiendo con la explicación vamos a ver en el Listado 6 como haríamos precisamente lo que hemos dicho.

Lo primero es importar la librería `httplib`, creamos entonces una conexión con el host en cuestión y pedimos el archivo `index.html`. Esa conexión genera una respuesta. La respuesta está formada por varias partes, entre ellas un código numérico (como el famoso 404), un texto que describe el error y una conexión al archivo que pedimos. En el caso de una conexión correcta recibiremos un 200, un OK y una conexión con el fichero. De esa conexión leemos con `read()` el contenido y lo almacenamos en una variable que llamamos `dato`. Entonces podremos cerrar la conexión como si de un fichero se tratara.

En ese momento ya tenemos la información que queríamos en `dato` y el canal cerrado. ¿No es muy difícil, no?

Listado 6: El "Recogedor" de Imágenes

```

001 #!/usr/local/bin/python                == len(self.lista))                respuesta.read()
002 #                                     037                                072                conexion.close()
    ---NOTA-----                        038                def longitud(self):                073
    -----                                039                return                                074                if( respuesta.status
003 # El fichero que debe ser                len(self.lista)                                075                != '200'):
    pasado como argumento                040                                076                nomb_fichero =
004 # debe consistir en un                041 def crea_directorio(cadena):                componentes[len(componentes)
    listado con una url por                042                componentes =                -1]
005 # linea.                                cadena.split('.')                                076                nomb_fichero =
006 #                                     043                                nomb_fichero[:-1]
    -----                                044                                077                archivo =
    -----                                if(os.path.exists(componentes                file(nomb_fichero , 'w')
007 class Lista_URLs:                                [0])):                                078
008     """Recibe un fichero y                045                print "Error: el                archivo.write(datos)
    carga                                directorio ya existe"                                079                archivo.close()
009     sus cadenas en una lista.                046                sys.exit()                                080                else:
    Provee de métodos                047                else:                                081                print "Fallo
010     para obtener de nuevo las                048                                os.makedirs(componentes[0])                082
    cadenas desde la                                049                                os.chdir(componentes[0])
011     lista."""                                050                print 'Creando
012     def                                directorio ' + componentes[0]
013     __init__(self,nombre):                051                                083 def genera_index(lista):
014         self.lista= []                052 def descarga_urls(lista):                084         print 'Generando índice
015         self.contador = 0                053         lista.rebobina()                index.html'
016         self.archivo =                054                                085         archivo =
    file(nombre)                                055         while( not lista.fin() ):                file('index.html','w')
017         self.cadena =                056         url =                                086
    self.archivo.readline()                lista.siguiete()                                087         archivo.write('<html>\n')
018                                057                                088         archivo.write('<head>\n')
019         while(self.cadena !=                058                componentes =                089         archivo.write('<title>
'\n'):                                url.split('/')                                Imagenes </title>\n')
020                                059                servidor =                090
    self.lista.append(self.cadena                componentes[2]                                archivo.write('</head>\n')
    )                                060                                091         archivo.write('<body>\n')
021         self.cadena =                061         ruta_imagen = '/'                092
    self.archivo.readline()                062         for i in range( 3,                archivo.write('<h1>Imagenes</
022         self.archivo.close()                len(componentes)):                h1>\n')
023                                063                 ruta_imagen =                093         archivo.write('<ul>\n')
024         def rebobina(self):                ruta_imagen + '/' +                094
025         self.contador = 0                componentes[i]                                lista.rebobina()
026                                064                                095         url = lista.siguiete()
027         def siguiente(self):                065                print 'Descargando                096         componentes =
028         if ( self.contador >=                imagen: ' + url[:-1]                url.split('/')
len(self.lista)):                                066                conexion =                097         imagen =
029         return ''                                httpLib.HTTPConnection(servid                componentes[len(componentes)
030         else:                                or)                                - 1]
031         self.valor =                067                                100
    self.lista[self.contador]                068                                101         while( url != ''):
032         self.contador =                conexion.request("GET", url)                102
    self.contador + 1                                069                respuesta =                archivo.write('<li><img
033         return self.valor                070                conexion.getresponse()                src="" + imagen
034                                071                datos =                +'></img></li>\n')
035         def fin(self):                                103                url =
036         return (self.contador                104                lista.siguiete()
                componentes =
                url.split('/')

```

Listado 6: El “Recogedor” de Imágenes (continuación)

```

105     imagen =
106     componentes[1:len(componentes)
107     - 1]
108     archivo.write('</u1>\n')
109     archivo.write('</body>\n')
110     archivo.write('</html>\n')
111     archivo.close()
112     #-----
113 # Main
114
115 if __name__ == '__main__':
116     import httpLib
117     import os
118     import os.path
119     import sys
120
121     if len(sys.argv) == 2:
122         lista =
123         Lista_URLs(sys.argv[1])
124         crea_directorio(sys.argv[1])
125         descarga_urls(lista)
126         genera_index(lista)
127
128     elif len(sys.argv) == 0:
129         print 'La sintaxis
130         del programa es:\n'
131         print sys.argv[0] + '
132         archivo\n'
133         print 'El archivo
134         debe contener una URL por
135         línea'
136     else:
137         print "ERROR: la
138         sintaxis es " + sys.argv[0] +
139         " <fichero>"

```

Parámetros

Estamos acostumbrados a poder pasar parámetros a los programas; en UNIX es algo común. Pero... ¿cómo podemos obtener los parámetros de ejecución en Python? De nuevo tenemos que recurrir a una librería: la librería **sys**. **sys** nos proporciona el acceso a los argumentos a través de su variable *argv*. Esta variable es una lista, por lo que podemos obtener los argumentos accediendo a las posiciones de la misma. La posición 0 contiene el nombre del programa que estamos ejecutando, y a partir de la posición 1 encontraremos los parámetros pasados. Al ser una lista podemos controlar la cantidad de los parámetros llamando a *len()*.

Programa

Ahora es el momento de poner todo lo aprendido en práctica con un programa que puede ser útil.

En este caso vamos a examinar un programa que realizará las siguientes tareas:

- El programa aceptará un parámetro de entrada que le indicará el nombre de un fichero.
- El programa abrirá ese fichero y lo leerá línea por línea. Cada línea del fichero será la dirección URL de una imagen.
- Cada URL será introducida dentro de una lista para su posterior uso.
- Una vez que hayamos acabado de leer el fichero, lo cerraremos y entraremos en la segunda

- parte del programa.
- Crearemos un directorio con el nombre del archivo que nos hayan dado.
- Cambiaremos el directorio de trabajo a ese directorio.
- Descargaremos cada una de las URLs dentro del directorio.
- Generaremos un archivo *index.html* que muestre las imágenes.

¿Mucho trabajo? Para eso están los programas. Evidentemente no realizaremos todas las comprobaciones que serían necesarias, ya que en tal caso el programa se alargaría demasiado, por lo que se deja al lector la opción de mejoras. Pensemos ahora en el diseño del programa. Tenemos varias partes:

Primero comprobaremos y almacenaremos la opción con el nombre del archivo. A continuación leeremos las URLs. El siguiente paso es crear un directorio y cambiar el directorio de trabajo. Después descargaremos las URLs. Y, por fin, hemos de generar el archivo HTML.

Vamos a seguir estos puntos para crear las funciones. Las URLs vamos a almacenarlas en una lista. ¿Deberíamos usar objetos? Esta es una de las cosas maravillosas que ofrece Python: NO estamos obligados a usar objetos. Y no digo que los objetos sean malos, sino que en ocasiones pueden llegar a ser engorrosos. Por ejemplo, podríamos crear un objeto *Lista_URLs* que aceptase como parámetro en su constructor el nombre de un fichero y que después nos permitiese ir consiguiendo las URLs una

detrás de otra. También podemos hacerlo mismo usando una función que cargue las URLs en una variable global. Aquí vamos a hacerlo con un objeto. Es en este momento cuando se deja al lector que explore la posibilidad de sustituir el objeto por una variable global y las funciones de lista. Algunas notas finales, el programa solo funciona con URLs de imágenes, y el archivo que contenga las URLs debe acabar con un fila vacía (un enter después de la última imagen).

Este programa (que se puede ver en el Listado 6) es muy simple y esta ampliamente comentado, pero de nuevo retamos a los lectores a mejorarlo y a introducirle control de excepciones. Suerte. ■

RECURSOS

- [1] Zope: <http://www.zope.org>
- [2] Mailman: <http://www.gnu.org/software/mailman/>
- [3] BitTorrent: <http://bittorrent.com/>
- [4] Universal Resource Language

LOS AUTORES

José María Ruiz actualmente está realizando el Proyecto Fin de Carrera de Ingeniería Técnica en Informática de Sistemas. Lleva 7 años usando y desarrollando software libre y, desde hace dos, se está especializando en FreeBSD. Pedro Orantes está cursando 3º de Ingeniería Técnica en Informática de Sistemas.