

Introducción a Python en Linux

HOLA PYTHON



Franz Piluegl, www.fotoia.de

Presentamos a Python, un lenguaje de script fácil de aprender, y te ayudamos a iniciarte en la creación de tus propios programas.

POR AELEEN FRISCH

A lo largo de mi carrera como administrador de sistemas, he tenido que escribir scripts en diversos lenguajes y entornos (comenzando con DCL en VMS). Cuando empecé a utilizar los sistemas basados en UNIX, primero estuve trabajando con C shell y luego con Perl, este último lo utilicé durante muchos años. Más recientemente, empecé a utilizar Python. Aunque la elección de un lenguaje de script es algo personal, y se pueden escribir scripts efectivos en diversos lenguajes, Python ofrece bastantes ventajas a los administradores de sistemas. Para mí, las más importantes son:

- Python es rápido de aprender. Los administradores con algo de experiencia en la programación de scripts escribirán programas en Python en minutos u horas en vez de en días o semanas.

- Python es más parecido a los lenguajes de programación tradicionales que a los otros lenguajes de script. Hace que se piense y se diseñen herramientas utilizando un marco de trabajo orientado a objetos.
- El sencillo modo interactivo de Python hace que sea fácil explorar nuevas características y depurar los programas.
- La escritura de herramientas en Python a menudo proporciona gratuitamente una funcionalidad multiplataforma.

La comunidad de Python generalmente es abierta y amistosa, considerando el código disponible como uno de los valores principales.

En este artículo presentaré el lenguaje Python al lector mostrándole diversos ejemplos de código y pequeños scripts, llenos de anotaciones y

comentarios, permitiendo que se aprenda acerca del lenguaje en su propio contexto.

Tabla 1: Tuplas de Iterales y Listas

<code>i=[1,2,3,4,5]</code>	Lista de enteros
<code>l=[1,2,'león',4,5]</code>	Las listas de elementos pueden ser de diferentes tipos
<code>t=(15,'tigre',2.11)</code>	Tupla que contiene tres elementos
<code>i[2]=i[-3]=3</code>	Segundo elemento y tercer elemento desde el final
<code>i[2:4]=[3,4,5]</code>	Porción de lista: cualquier índice puede omitirse (por ejemplo, <code>i[:2]=i[0:2]</code>)

Primeros Pasos

Aquí tenemos el primer programa en Python, que es excesivamente simple:

```
#!/usr/bin/python
print 'Hola Mundo!'
```

Tras la designación inicial de la shell, la única instrucción del programa imprime la cadena de caracteres especificada a la salida estándar. En Python, las cosas simples suelen ser a menudo sencillas, como es el caso de la sentencia *print*.

Python posee además un modo interactivo, al que se puede llegar ejecutando el comando *python*:

```
$ python
>> print 'Hola Mundo!'
Hola Mundo!
```

Sólo con un poco de experiencia se puede aprender bastante sobre las variables y los tipos de datos en Python, como se muestra en el Listado 1. El Listado 2 es un sencillo script que ilustra diversas características importantes de este lenguaje. Se requiere un servidor POP para el estado de buzón de correo de un usuario.

El script en el Listado 2 contiene varios puntos importantes. Primero, la sentencia *import* indica los módulos que se van a utilizar en el script (los módulos proporcionan capacidades adicionales sobre las construcciones básicas de Python). Aquí he utilizado

el módulo relacionado con el protocolo POP3 y el módulo del sistema necesario para el acceso a los argumentos del script. Éstos se almacenan en una lista (array) denominada *sys.argv[]*. Los elementos individuales se seleccionan con la notación familiar basada en los corchetes. La numeración comienza en cero; *sys.argv[0]* almacena el primer elemento de la línea de comandos, el nombre del script.

En este script, la variable *who* se utiliza para almacenar el primer argumento, que es la cuenta del usuario cuyo buzón de correo se desea utilizar. El script también define dos variables que almacenarán cadenas de caracteres.

En la última sección del script es donde se realiza en realidad el trabajo. Primero se abre una conexión a un servidor de correo especificado por medio de la función *POP3_SSL()* en el módulo *poplib* (téngase en cuenta la notación: *poplib.POP3_SSL()*). El manejador de la conexión (que identifica la estructura de datos) se almacena en la variable *m*.

El resto del script utiliza varios métodos (funciones) que están definidos para las conexiones POP3 en el módulo *poplib*. Los métodos *user()* y *pass()* transmiten los datos de autenticación al servidor, el método *stat()* obtiene el número de mensajes presentes en el buzón de correo del usuario (además del espacio utilizado), y el método *quit* cierra la conexión.

La sintaxis para la llamada de los métodos utiliza el formato estándar de la programación orientada a objetos, en el que el nombre del método va a continuación del nombre del objeto al que se aplica, separado por un punto, y seguido por los argumentos entre paréntesis, por ejemplo, *m.user(who)*.

El método *stat()* devuelve una tupla de dos valores que contiene datos sobre el buzón de correo: (*#messages*, *#bytes*). El script selecciona el primer elemento para imprimir con el sufijo *[0]*. La sentencia *print* es algo más complicada aquí; incluye una cadena de formato y una lista de parámetros entre paréntesis a imprimir (otra tupla), separados por signos de porcentaje. Aquí, *%d* y *%s* especifican un entero y una cadena de caracteres respectivamente. Luego veremos más ejemplos de código de formatos.

Incluso este simple script muestra lo potente que puede llegar a ser Python, de modo que una consulta a un servidor sólo supone el uso de unas cuantas sentencias. Hay multitud de módulos para Python, incluso para la mayoría de los servidores y servicios de red.

Listado 1: Trabajando con Python

```
01 >> 3/5          16 7
02 0                17 >> a+c
03 # El resultado de una  18 # Operación no permitida
   operación con
04 # enteros también es un entero
05 >> 3.0/6        19 # (mensaje corto) ...
06 # Forzar una operación 20 TypeError: cannot concatenate
07 # de punto flotante    'str' and 'int' objects
08 0.5
09 >> a='lion'; b='ess'; c=3 21 >> a+str(c)
10 # Definir 3 variables  22 # ... pero Python proporciona
11 >> a+b            23 # funciones de conversión
12 # Concatenación de Cadenas
13 'lioness'         24 'lion3'
14 >> c+4           25 >> ^D
15 # Suma
```

Listado 2: Obtención del estado del buzón de correo

```
01 #!/usr/bin/python
02 import poplib,sys
03 # Habilita los módulos de
   Python
04
05 who=sys.argv[1]
06 # Almacena el argumento del
07 # primer script
08 pw="xxxxxxx"
09 mailsrv="mail.ahania.com"
10
11 m=poplib.POP3_SSL(mailsrv)
12 # Abrir conexiones seguras
13 # al servidor POP3
14 m.user(who)
15 # Autenticación
16 m.pass_(pw)
17 print "There are %d messages
   waiting for %s." %
   (m.stat()[0],who)
18 m.quit
```

Las tuplas y las listas son dos ejemplos de colecciones: variables que almacenan múltiples valores. También son conocidas como secuencias de objetos, ya que sus elementos están ordenados.

La Tabla 1 muestra algunos ejemplos de tuplas y listas. El último ilustra cómo se obtiene una porción de lista; también se pueden tomar porciones de tuplas y de cadenas de caracteres.

Estructuras de Control en Python

El siguiente ejemplo explora las sentencias de Python relacionadas con los bucles y la ejecución condicional. En el script del Listado 3, que convierte temperaturas entre diferentes escalas, vemos el procesamiento de los argumentos y algunos métodos para manipulación de cadenas de caracteres.

El Listado 3 presenta varias estructu-

ras de control importantes: *try/except*, *if/else* y *for/in*. La construcción *try/except* intenta la ejecución de las instrucciones que se encuentran especificadas por *try* y ejecuta las que se encuentran en *except* en el caso de que fallaran algunas de las que se encuentran en *try*. Hay que destacar que estas sentencias van seguidas por el carácter dos puntos, y que el bloque de instrucciones siguiente se indica por medio del sangrado. En general, Python se basa en el sangrado del código para identificar los bloques estructurales (en vez de, digamos, las llaves como en C). Estas dos características son compartidas por la construcción *if/else* más adelante en este mismo script.

El bucle *for* es similar al de los otros lenguajes de programación y de script. Tiene la forma general siguiente: *for variable(s) in lista:*, y las variables especificadas son asignadas sucesiva-

mente al comienzo de cada iteración del bucle. En el Listado 3, la lista se almacena en la variable *olist*, que contiene una lista de tuplas. En el bucle *for* se especifican dos variables que asignan a los elementos primero y segundo de cada tupla en la sucesión.

La función *getopt()* se utiliza para analizar los argumentos de la línea de comandos. Toma la lista de argumentos y una cadena con las letras válidas de los argumentos (y otra información). Aquí se pasan todos los argumentos al script excepto el último (que se almacenó previamente en la variable *temp*), siendo los argumentos válidos *-c*, *-k*, *-f* y sus correspondientes en mayúsculas. La función comprueba los argumentos especificados para validar la presencia de los valores requeridos, generando los errores apropiados. Una vez que *getopt()* haya completado su proceso, el script continúa procesando los argumentos.

En este caso, el script realiza una comparación de cadenas. Python soporta varios operadores de comparación: *=* (igualdad), *!=* (desigualdad), *<* y *>* (menor que y mayor que, respectivamente, que pueden ir seguidos por *=*). Se pueden construir operaciones lógicas complejas con los paréntesis y las palabras reservadas *and*, *or* y *not*.

Operaciones con Cadenas de Caracteres

El Listado 3 presenta el método *lower()* (minúsculas) para las cadenas de caracteres, el método correspondiente *upper()* para la conversión a mayúsculas, y el método *title()* para la capitalización de los títulos. El Listado 4, que descarga un fichero desde un sitio web y procesa su contenido, presenta la búsqueda en cadenas de caracteres y la extracción de subcadenas. La primera parte del fichero inicializa una conexión http a un servidor de Internet y obtiene un fichero que contiene los datos de conversión entre divisas, a continuación almacena los valores en variables.

La segunda parte del script extrae el tipo de conversión correspondiente al dólar americano y el valor monetario que se desea convertir a partir del primer argumento del script. Para cada moneda, los datos en *rates* se buscan a

Listado 3: Conversión de Temperaturas

```
01 #!/usr/bin/python          Opción de convertir a
02 import sys #Si importa los  minúsculas y comparar
    módulos como en
03 from getopt import * #la línea  20   is_c=1
    siguiente, los métodos se  21   if opt.lower()=='-k':
    pueden invocar           22     print_k=1
04 #sin ser precedidas por el  23
    nombre del módulo
05 is_c=0                    24 t=eval(temp) #Covertir cadena
06 print_k=0                 a número y realizar el cálculo
07 temp=sys.argv[-1] #Encuentra  25 if is_c==0:
    el argumento del script que  26   # De Fahrenheit a Celsius
    contiene la temperatura     27   ftemp=t
08 try: #Comprueba una operación,  28   ctemp=(t-32.0)*.55
    pero...                    29 else:
09   t=float(temp)           30   # De Celsius a Fahrenheit
10 except: #...comprueba        31   ctemp=t
    cualquier error y sale     32   ftemp=(t*1.8)+32.0
11   print "%s is not a        33   ktemp=ctemp+273.0
    temperature." % temp
12   sys.exit(1)
13
14 # analizar argumentos del    34
    script #getopt devuelve una  35 if print_k: #Print Kelvin only
    lista de (-x, val)         if requested
15 # tuplas en olist
16 olist,alist =             36   print "%.1f Fahrenheit is
    getopt(sys.argv[1:-1],"cCfFkK"  %.1f Celsius is %.1f K."
    )
17 for opt,a in olist: # Bucle: a  37   %(ftemp,ctemp,ktemp)
    los elementos se le asignan
    variables
18 # para cada tupla en olist  38   print "%.1f Fahrenheit is
19   if opt.lower()=='-c': #    %.1f Celsius." %(ftemp,ctemp)
```

partir de la cadena de caracteres especificada de acuerdo al método *index()*, el cual devuelve la localización del carácter de la primera ocurrencia. Para extraer la línea relevante de *rates*, que comienza en la localización devuelta por *index()* y sigue hasta el siguiente salto de línea y retorno de carro, se usa un *slice*. Aquí se muestra el uso del segundo argumento opcional del método *index()*, que indica la localización dentro de la cadena de caracteres a partir de la que tiene que empezar a realizar la búsqueda (el valor por defecto para el comienzo es 0).

Este es un ejemplo de los datos que deberían extraerse para una moneda determinada (en este caso, el yen):

```
intl1line=Yen,0.0086864,
0.008691,0.008730,0.008731,
0.008465,0.008513
intl1line.split(*,*)[-1]=
0.008513
```

El método *split()* divide una cadena de caracteres en una lista de elementos delimitados en la cadena original por un carácter separador especificado. En este caso se utiliza *split* con el carácter separador coma para dividir la cadena y el sufijo *[-1]* para obtener el elemento final dentro de la línea deseada (tasa de conversión más reciente). El script finaliza calculando la tasa de conversión entre la moneda extranjera especificada y el dólar US (en base a los datos Canadienses disponibles en Internet), e imprimiendo el resultado. La última sentencia condicional se asegura de que la información mostrada se hace de la forma más comprensible posible. Los códigos del formato en esta sentencia de impresión incluyen los números en punto flotante (*f*) junto con los valores de anchura de los campos de salida de seis caracteres, con dos lugares a la derecha del punto decimal.

Otros métodos útiles para el tratamiento de las cadenas de caracteres que se deben conocer son *replace()*, que sustituye una subcadena por otra nueva, *find()*, un método más potente para buscar subcadenas, y *strip()*, junto con sus variantes *lstrip()* y *rstrip()*, que eliminan los caracteres en blanco precedentes o restantes.

Hasta aquí se han considerado un

Listado 4: Descargando y Procesando

```
01 #!/usr/bin/python
02 import urllib
03 # Módulo para operaciones HTTP
04 url =
    "/en/markets/csv/exchange_eng.
    csv"
05 c =
    urllib.HTTPConnection("www.ba
    nkofcanada.ca")
06 c.request("GET", url)
07 r = c.getresponse()
08 rates = r.read()
09 c.close()
10
11 intl =
    rates.index(sys.argv[1])
12 us = rates.index("U.S.
    Dollar")
13 intl1line =
    rates[intl:rates.index("\r\n",
    intl)]
14 intlrate =
    intl1line.split(",")[-1]
15 us1line =
    rates[us:rates.index("\r\n",
    us)]
16 usd = us1line.split(",")[-1]
17 rate=
    float(intlrate)/float(usd)
18 if rate >= .25:
19     print "%4.2f US Dollars =
    1%s" % (rate,sys.argv[1])
20 else:
21     print "1 US Dollar =
    %6.2f%s" %
    (1.0/rate,sys.argv[1])
```

par de tipos de operaciones de red, así como de manipulación de cadenas de caracteres y de operaciones aritméticas. En el siguiente script, veremos el sistema de ficheros.

Python ofrece un sencillo mecanismo para recorrer el árbol de directorios y operar potencialmente con alguno o con todos los ficheros hallados en él: el método *walk()* del módulo *os*.

Caminando por el Sistema de Ficheros

El ejemplo del Listado 5 es uno de los favoritos para ilustrar el funcionamiento del método *walk()* dentro del módulo *os*, ya que actúa de una forma bastante sencilla: cambiando las extensiones de los ficheros. Este Listado 5 también muestra la práctica estándar en Python para implementar funcionalidades como métodos a la hora de ayudar en la reutilización del código. Hay que tener en cuenta que todo lo que se necesita sintácticamente es la palabra reservada inicial *def*, los dos puntos que siguen a la secuencia de llamada y el bloque de sentencias sangrado que componen el código que implementa la funcionalidad del método.

Tras eliminar cualquier punto inicial de la antigua y de la nueva extensión especificada, se llama al método *os.walk()*. Éste devuelve una lista de tuplas de tres elementos, conteniendo

un subdirectorio y la lista de los nombres de los ficheros que aparecen en cada uno de los directorios que encuentre. Dos bucles *for* iteran sobre cada directorio y sobre los ficheros que alberga. El método *split()* se utiliza para dividir el nombre del fichero en límites regulares, y el índice *[-1]* de nuevo se utiliza para obtener el último elemento, la extensión del fichero (pueden probarse otros comandos similares en el modo interactivo para ver cómo funcionan). Si coincide la extensión, entonces se construyen el nuevo nombre de fichero (*new_f*) y su ruta, y se pasan al método *os.rename()*.

El código en la sección final del script se introduce para que el fichero que incluye el método pueda ejecutarse directamente, así como desde otros scripts de Python; el código comprueba el nombre actual del método para indicar si es la rutina principal. Si es así, verifica si se le ha pasado el número suficiente de argumentos antes de invocar al método (en un entorno productivo deberían realizarse más comprobaciones por seguridad).

Acumulación de Datos con Diccionarios

Otra de las tareas que a menudo realizan los administradores consiste en realizar búsquedas en los ficheros de registro, y Python es una buena elec-

ción para llevarla a cabo. Estas operaciones a menudo de benefician de los diccionarios de Python: arrays indexados (normalmente) por palabras claves en vez de por enteros. En otros lenguajes estas estructuras se denominan arrays asociativos. Aquí se presenta un ejemplo de diccionario junto con dos asignaciones de elementos del mismo:

```
temps={"boil":212,
"freeze":32,
"scald":120, "cold":10}
temps["hot"]=110
t=temps.get("tepid",66)
# Devuelve el valor solicitado
o si no está definido el valor
por defecto especificado
```

La segunda sentencia añade un elemento al diccionario con la clave *hot* (y valor 110). La última obtiene el valor de la clave *tepid*, si ese elemento existe; en otro caso, devuelve el valor 66.

El script del Listado 6 produce un informe de tipos de error y los cuenta en el fichero de registro */var/log/messages*. El fichero define una función general que abre y lee el fichero de registro, dividiendo cada línea en campos a partir de los espacios en blanco, y compara un campo específico (índice *which*) con la cadena almacenada en *match* en busca de líneas relevantes. Cuando se encuentra alguna línea relevante, el dato del campo designado en *collect* se convierte en una clave del diccionario; el valor correspondiente es la cuenta para el número de ocurrencias de esa clave dentro del fichero. El valor por defecto del método *get()* del diccionario se utiliza para inicializar un contador para las claves nuevas a 1.

La segunda parte del script llama a la función genérica que se ha creado en busca de mensajes de la utilidad *sudo* dentro de */var/log/messages*, registrando el nombre del usuario presente en cada entrada. Este código también ilustra el uso de una variable para la cadena de formato dentro de la sentencia *print*. Los códigos de formato que contiene incluyen lo que parece ser un valor negativo para el ancho de un campo; de hecho, el signo menor indica que la salida debe justificarse a la izquierda dentro del

campo. Aquí mostramos otro ejemplo de salida de este comando:

```
sudo Command Usage by User
User Times
aefrisch 5
chavez 191
root 100
```

GUI para Scripts con Tkinter

La creación de programas y herramientas con una interfaz gráfica es también una tarea sencilla y rápida con Python. Se encuentran disponibles varios módulos gráficos diferentes. Probablemente Tkinter sea el más utilizado. El Listado 7 es un script de ejemplo que muestra algunas de las características más básicas de Tkinter.

Este Listado 7 importa el módulo necesario, utilizando de nuevo la sintaxis que permite omitir los prefijos del nombre del módulo en la invocación de los métodos. A continuación se crea la ventana principal, donde residirán el resto de los elementos gráficos y se establece el título de la misma. Luego se añaden a la ventana una etiqueta de texto y un botón.

Cada elemento tiene que llamar a su método *pack()* antes de visualizarse. El script finaliza entrando en el bucle principal de eventos, donde se queda a la espera de la interacción del usuario, al que responderá de acuerdo con ésta.

En la sentencia donde se crea el botón se especifica la ventana en la que éste residirá, el texto que tiene que mostrar, el color de este texto y el comando a ejecutar cuando se pulse el botón. Esto último se denomina *callback*, y es normalmente una llamada a una función interna o a un método definido por el programador (en este caso se realiza una llamada a una función interna). La Figura 1 muestra el aspecto de esta ventana (su apariencia variará dependiendo del gestor de ventanas que se esté utilizando).

Presionando el botón *Quit* se invoca al método *quit* de la ventana, que sale del bucle de eventos y la destruye.

Ahora se va a examinar una aplicación GUI más compleja para manipular los servicios del sistema, con el objeto de reiniciar los servidores que se estén ejecutando. El programa genera una casilla de verificación de

Listado 5: Cambiando Extensiones

```
01 #!/usr/bin/python
02 import os
03
04 def mass_mv(dir, old, new):
05     # Define un método
06     # (subrutina) con tres
07     # argumentos
08     if old[0] == ".":
09         # Si existe elimina
10         # el punto inicial de las
11         # extensiones antiguas y
12         # nuevas
13         old = old[1:]
14         l=len(old)
15         # Guarda la longitud de la
16         # extensión anterior
17         if new[0] == '.': new
18         =new[1:]
19         # # os.walk
20         # devuelve una lista de:
21         # (ruta, nombre de
22         # directorios,
23         # nombre de ficheros)
24         for path, ignore, files in
25             os.walk(dir):
26             for f in files:
27                 # Repasa cada fichero en
28                 # el
29                 # subdirectorio
30                 if f.split(".")[1] ==old:
31                     new_f = f[0:-1]+new
32                     os.rename(path+"/"+f,path+"/"+
33                             new_f)
34             return
35     # Fin del método mass_mv
36
37 if __name__=='__main__':
38     # Verdadero cuando el script
39     # se
40     # ejecuta directamente
41     import sys
42     if len(sys.argv) < 4:
43         print "Too few args:
44         start-dir old-ext new-ext"
45         sys.exit(1)
46     mass_mv(sys.argv[1],
47             sys.argv[2], sys.argv[3])
```

forma dinámica por cada servidor especificado en un fichero de configuración. Algunos ejemplos de entrada podrían ser los siguientes:

```
Label:Type:Name-or-Command
Cron:i:cron
yslog:h:syslogd
xinetd:h:xinetd
Mail:s:/etc/init.d/ ↗
postfix reload
Printing:i:cups
```

Los campos tienen el texto para las casillas de verificación, el tipo de método de señalización y el nombre del servicio (o el comando completo).

El script del Listado 8 comienza con las sentencias *import* seguidas por la definición de un método, que se examinará más adelante. A continuación se crea la ventana principal, se establece su título y se le añade un texto.

Cada línea del fichero de configuración se divide en los tres campos que la forman utilizando el carácter dos puntos como delimitador. Cada uno de ellos es almacenado

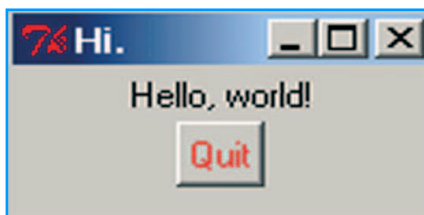


Figura 1: Tkinter permite crear una GUI en Python en unos pasos..

en la lista *dconfig* por el método de la lista *append()*; el último carácter del tercer campo se descarta (carácter de salto de línea). A la lista *states* se le añade una variable de tipo entero y se inicializa para usarse con los controles Tkinter. Por último, se crea una casilla de verificación para cada entrada con la etiqueta que indica el fichero de configuración (el primer campo) como el texto del botón.

La casilla de verificación resultante es controlada por una variable, el elemento correspondiente de *states*, cuyo valor puede establecerse, consultarse o ambas cosas. Esta sección del script termina creando dos botones: uno para actuar sobre el servidor seleccionado y otro para que la aplicación finalice. El argumento del

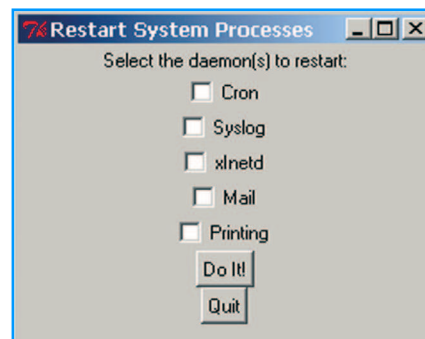


Figura 2: Una GUI en Python práctica.

comando para el primer botón se merece una atención especial, ya que hace uso de la denominada función *lambda*, la cual hace que el método especificado sea pasado al control en vez de ser ejecutado inmediatamente (se puede probar a crear un control con una función callback con y sin esta construcción para comprender su funcionamiento). La Figura 2 muestra un ejemplo de cómo es la ventana resultante.

El Listado 9 es la función callback (que en realidad debería aparecer al comienzo del fichero del script). Examina el estado del array. Para los elementos que son iguales a 1, que

Listado 6: Informe de tipos de error

```
01 #!/usr/bin/python
02
03 def count_em(logfile_pathname,which,match,collect):
04     counts = { }
05     # Inicializar diccionario
06     lf = open(logfile_pathname,"r")
07     contents = lf.readlines()
08     # Se coloca el fichero completo
09     # en contents
10     for line in contents:
11         fields = line.split()
12         # Lo divide por los espacios en blanco
13         if fields[which] ==match :
14             what=fields[collect]
15             counts[what] = counts.get(what, 0) + 1
16     return counts
17
18 if __name__=='__main__':
19     sudo_users = count_em("/var/log/messages",4,"sudo:",5)
20     format = "%-8s %-5s"
21     # Define el formato de salida
22     # de la cadena para varios usos
23     print "sudo Command Usage by User"
24     print format % ("User", "Times")
25     users = sudo_users.keys()
26     # Extrae las claves del diccionario
27     # en una lista...
28     users.sort()
29     # ...ordena la lista
30     # de usuarios...
31     for u in users:
32         # ...e imprime los datos
33         # de forma ordenada
34         print format % (u, sudo_users[u])
```

Listado 7: GUI en Python con Tkinter

```
01 #!/usr/bin/python
02 from Tkinter import *
03 window = Tk()
04 # Crear una ventana
05 window.title("Hi.")
06 # Asignar a la ventana un título
07 w = Label(window,text="Hola, mundo!")
08 w.pack()
09 # Finalizar y construir la
10 # etiqueta
11 q = Button(window,text="Quit",fg="red",command=window.quit)
12 q.pack()
13 # Finalizar y construir el
14 # botón
15
16 window.mainloop()
17 # Bucle de evento: esperar que
18 # el usuario haga algo
```

corresponden a los elementos cuyos cuadros están marcados, se les genera el comando apropiado en base al tipo del segundo campo: los comandos del tipo *i* se arrancan con la palabra reservada *restart* con un script de */etc/init.d*, cuyo nombre es el tercer campo del fichero de configuración; a los comandos del tipo *h* se les envía una señal HUP por medio del comando *killall* con el nombre del tercer campo como argumento, y cualquier otro tipo utiliza el tercer campo como el comando completo para su ejecución.

El comando construido se ejecuta por el método *os.popen()*, encargado de ejecutar comandos externos de la shell. El último acto del método es reiniciar la casilla de verificación para cada elemento. Hay que tener en cuenta que este script no realiza nin-

gún control de errores, pero ilustra muchas de las características útiles que posee Python.

Explorando Más

Solamente se han explorado unas pocas de las múltiples características básicas de Python, pero el lector ya debería estar preparado para aprender más sobre este lenguaje por su propia cuenta. Dos de sus características más útiles, que no hemos examinado en este artículo, son las clases y las extensiones que se escriben en otros lenguajes de programación. Python hace que sea sencillo tanto la programación orientada a objetos como el uso de métodos y programas escritos en otros lenguajes de programación como C, C++ y Fortran. Véase el cuadro Recursos para más información. ■

Listado 8: Añadiendo Opciones a la GUI

```

01 #!/usr/bin/python
02 from Tkinter import *
03 import os
04
05 def do_it(s,d):
06     # Código listado y descrito
07     # a continuación
08
09     w=Tk()
10     # Crear una ventana
    principal (root)
11     w.title('Restart System
    Processes')
12     txt=Label(w,text='Select the
    daemon(s) to restart:')
13     txt.pack()
14
15     # El siguiente trozo del
16     # script está relacionado
    con el
17     # procesado del fichero de
18     # configuración y añade
    casillas de
19     # verificación a la ventana
    para
20     # varias entradas:
21
22     states=[]
23     # Inicializar las listas y
24     # la variable i
25     dconfig=[]
26     i=0
27
28     dfile=open('/usr/local/sbin/da
    emons.conf','r')
29     devils=dfile.readlines()
30     # Obtiene la lista de
    procesos del
31     # fichero de
    configuración...
32     for d in devils:
33         # ...almacena los datos...
34         (dlabel,dtype,dcmd)=d.split(":")
35         dconfig.append(dlabel)
36         dconfig.append(dtype)
37         dconfig.append(dcmd[0:-1])
38         states.append(IntVar())
39         # ...y crea el botón
40         c=Checkbox(w,text=dlabel,va
    riable=states[i],width=6)
41         c.pack()
42         i+=1
43     q=Button(w,text='Do
    It!',command=lambda:do_it(sta
    tes,dconfig))
44     q.pack()
45
46     q=Button(w,text='Quit',command=w.
    quit)
47     q.pack()
48     w.mainloop()

```

Listado 9: Función Callback

```

01 def do_it(s,d):
02     # Función Callback
03     # para un botón
04     for i in range(len(s)):
05         # Examina el estado de la
    lista: ...
06         if s[i].get() == 1:
07             # Si el cuadro está
    marcado,...
08             start=i*3
09             name=d[start]
10             # desempaqueta el campo
11             # correspondiente de la
12             # lista ...
13             type=d[start+1]
14             cmd=d[start+2]
15             if type=="i":
16                 # configura el
    comando
17                 # apropiado...
18
19             the_cmd="/etc/init.d/"+cmd+"
    restart"
20             elif type=="h":
21                 the_cmd="killall
    -HUP"+cmd
22             else:
23                 the_cmd=cmd
24                 os.popen(the_cmd)
25                 # lo ejecuta...
26                 s[i].set(0)
27                 # ...y resetea
    # la casilla

```

RECURSOS

- [1] Resumen de sintaxis de este artículo: http://www.aeleen.com/python_summary.htm
- [2] Sitio web oficial de Python: <http://www.python.org>
- [3] Python Cookbook (ejemplos): <http://aspn.activestate.com/ASPN/Cookbook/Python>
- [4] "Instant Python" por Magnus L. Hetland: <http://www.hetland.org/python/instant-python.php>
- [5] "Python Short Course" por Richard P. Muller: http://www.wag.caltech.edu/home/rpm/python_course