Interfaces múltiples en Python

Marcos Sánchez Provencio

rapto@arrakis.es

Este artículo es una breve introducción a las posibilidades dinámicas de Python. Generaremos rápidamente módulos intercambiables de interfaz para un sencillo servidor de datos. En concreto, veremos una interfaz interactiva (con tkinter) y un servidor de aplicaciones HTTP. El código funciona sin modificación alguna en cualquier plataforma en la que funcione Python, ya que se han utilizado sólo las bibliotecas estándar de Python.

1. Sobre Python

Python es un lenguaje de programación interpretado, orientado a objetos y de sintaxis sencilla. Se ha implementado sobre muchos sistemas operativos (incluyendo Linux, Windows, VMS, AmigaOS...) y para los entornos más variados (incluyendo .NET de Microsoft y Java, de Sun).

Los usuarios de Python destacan su legibilidad (incluyendo el código ajeno) y su versatilidad (en cuanto a funciones y a escalabilidad). Es poco común encontrar programadores que hayan pasado del Python (dentro de su rango de aplicación, claro) a otro lenguaje. Las aplicaciones de Python más conocidas son:

- Zope, un servidor de aplicaciones de alto nivel
- Mailman, el gestor de listas de correo de GNU

2. Arquitectura de la aplicación

La arquitectura de la aplicación, a pesar de hacer todas las concesiones a la brevedad, intentará emular la de un servidor de datos a medida. Nuestra base de datos utilizará el servicio más simple de datos persistentes de Python, **dbm**.

La primera capa, precisamente, nos independizará del servicio de datos subyacente a nuestra aplicación, ofreciendo una interfaz orientada a objetos especializada en el tema de interés de nuestra aplicación. Como ejemplo sencillo, hemos tomado una agenda de direcciones de correo electrónico. Dispone de funciones para añadir direcciones y buscarlas posteriormente a partir de la clave exacta (búsqueda rápida) o a partir de cualquier parte de la clave (búsqueda lenta).

La segunda capa (versión tkinter) implementa una interfaz de programa interactivo clásico. En general, las interfaces de este tipo son mucho más rápidas que las interfaces web, aunque exigen al cliente tener instalado cierto software. La ventaja de Python en este caso es que está incluido en todas las distribuciones de Linux, y que la instalación en Windows se hace de una vez por todas, sin exigir instalaciones posteriores en ningún caso. Es posible generar ejecutables autocontenidos (que incluyen su propio intérprete) si se desea asegurar que la ejecución es correcta (sería parecido a una compilación estática).

La segunda capa (versión HTTP) implementa un servidor de aplicaciones mínimo, pero completo. Lo único que hace es esperar peticiones en el puerto 8000. Si alguna de las peticiones es al documento `/busca`, con el parámetro CGI param, se ejecuta el método homónimo sobre el objeto agenda (que abre la BD una sola vez, lo que le permite responder con más rapidez que un CGI) y se devuelven los resultados con un formato HTML presentable.

3. La capa de datos

A continuación se presenta el listado íntegro del módulo que implementa la capa de datos.

```
#agendadb.py
import anydbm        #Gestión de datos persistentes
import string        #Gestión de cadenas de texto
import pickle        #Conversión de objetos a cadena (serialización)

class agenda:
    def __init__(self,fich='agenda.db'):
        self.db=anydbm.open(fich,'c')    #Abrimos la BD al instanciar el objeto

    def encuentra(self, clave):            #Búsqueda rápida
        clave=string.upper(str(clave))
```

```
        try:
            #Recuperamos el objeto a partir de la representación interna
            return pickle.loads(self.db[clave])
        except:
            #Si no se encuentra, se devuelve None (el nulo de Python)
            return None

    def busca(self, clave):        #Búsqueda lenta, por contenido
        ret=[]
        clave=string.upper(str(clave))
        for k in self.db.keys():
            if string.find(k,clave)>-1:
                #Añadimos el elemento al resultado
                ret.append( (k,pickle.loads(self.db[k])) )
        return ret

    def nuevo(self,clave,contenido):    #Nuevo elemento de la BD
        nombre=string.upper(str(clave))         #Las claves independientes de mays/minus
        picContenido=pickle.dumps(contenido)     #Convertimos lo que venga a cadena
        self.db[nombre]=picContenido             #Lo guardamos en la BD con su clave

if __name__==’__main__’:        #No es un módulo, hacemos una prueba unitaria
    ag=agenda()
    if not ag.db:                 #Agenda vacía, metemos contenido inicial
        ag.nuevo(’ErnestoBKE’, (’Ernesto Molina’,’emolina@grupoburke.com’))
        ag.nuevo(’Ernesto’, (’Ernesto Molina’,’rotoxl@jazzfree.com’))
        ag.nuevo(’MarcosBKE’, (’Marcos Sánchez’,’msanchez@grupoburke.com’))
        ag.nuevo(’Marcos’, (’Marcos Sánchez’,’rapto@arrakis.es’))

    print ag.encuentra(’Ernesto’)     #Recuperamos unos cuantos datos
    print ag.busca(’BKE’)
```

3.1. Comentarios al listado

No se ha incluido en este módulo nada de código de interfaz, salvo lo trivial de print, que presenta en la consola texto sin formato.

El módulo pickle es responsable de la serialización, es decir, convertir a cadena (de manera reversible) cualquier objeto. pickle es muy flexible y normalmente funciona sin tener que preocuparse de cómo lo hace.

De momento, es posible pasar a la base de datos cualquier objeto y cualquier clave. Es costumbre en Python dejar cuantas más características posibles abiertas, para no restringir la utilidad de los módulos para un uso posterior.

Python dispone de interfaces de datos muy variadas, desde la interfaz minimalista que vemos aquí, hasta el acceso a servidores de datos multidimensionales. Cuenta con bibliotecas nativas para los servidores más comunes y con puentes con ODBC/JDBC para cubrir el resto de los servidores.

Si es la primera vez que ves un listado en Python, habrás observado que los niveles de anidamiento se marcan simplemente por el margen izquierdo.

No es necesario declarar las variables, aunque es ilegal utilizarlas sin asignarles anteriormente un valor. Los tipos de las variables en Python se determinan en el instante en que se definen asignándoles su valor.

Se utiliza el latiguillo if __name__==’__main__’: para detectar que el módulo está actuando como programa autónomo. Se acostumbra a poner funciones de prueba del propio módulo en este caso (si el módulo no tiene una utilidad por sí solo).

El argumento self de los métodos indica el propio objeto, a la manera de this (Java) o Me (Visual Basic). En Python este argumento es explícito y no se puede omitir en ningún caso.

4. La capa de interfaz gráfica (tkinter)

A continuación se presenta el listado íntegro del módulo que implementa la capa de datos.

Figura 1. La interfaz tkinter


```
from Tkinter import *

class agendatk:
    def __init__(self,tipo_bd):
        #Nuestra capa de BD es una clase que nos pasan en el constructor
        self.db=tipo_bd()
        #Contruimos el interfaz gráfico
        self.r=Tk()
        self.t=Entry(self.r)
        self.t.pack()
        self.res=Label(self.r,text='Sin resultados')
        self.res.pack()
        Button(self.r,text='Buscar',command=self.busca).pack()
```

```
        Button(self.r,text='Encontrar',command=self.encuentra).pack()

    def run(self):
        self.r.mainloop()

    def busca(self):
        texto=self.t.get()
        ret=self.db.busca(texto) or []
        for elem in ret:
            texto = texto + '\n%s = %s - %s ' % (elem[0] , elem[1][0], elem[1][1])
        self.res.configure(text=texto)

    def encuentra(self):
        texto=self.t.get()
        ret=self.db.encuentra(texto)
        self.res.configure(text='%s - %s' % ret)

if __name__=='__main__':
    import agendadb
    agtk=agendatk(agendadb.agenda)
    agtk.run()
```

4.1. Comentarios al listado

En Python es posible pasar casi cualquier cosa como parámetro: Una variable simple, un objeto cualquiera, atributos de un método, funciones, métodos, módulos, clases... Además, el modo de hacerlo es el más intuitivo posible. En este caso estamos pasando una clase de acceso a datos al constructor de nuestra interfaz y las funciones de respuesta a los sucesos.

No hay nada privado ni protegido en Python. Todos los atributos de una clase y sus métodos son públicos. Esto fomenta la cooperatividad entre colaboradores y elimina horas de trabajo empleadas en decidir qué es publico o privado, y en permitir acceso a valores privados.

La interfaz tkinter está fuertemente basada en tcl/tk, hasta el punto de que lo utiliza internamente. Por supuesto, esto provoca un descenso en el rendimiento, pero es inapreciable en la mayoría de las aplicaciones, como en ésta. Si se conoce tcl/tk, todo el conocimiento de construcción de interfaces es inmediatamente aplicable a tkinter. Tan sólo hay que hacer la traducción inmediata a Python. En Python se pueden realizar interfaces gráficas con wxWindows, la más pujante, con swing y awt en el caso de Jython y otras menos conocidas.

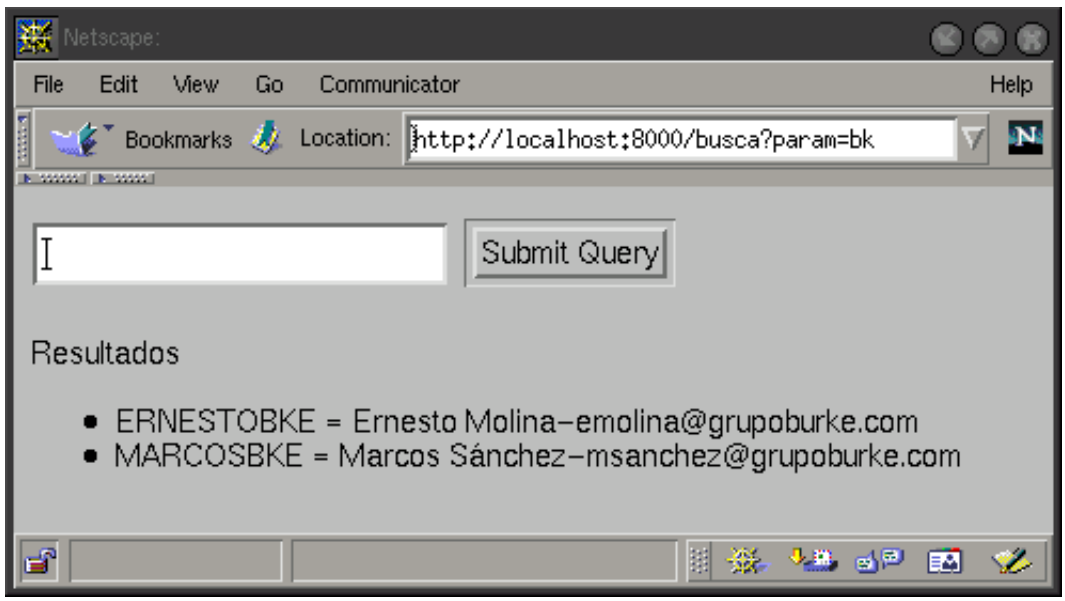
Vemos que se importa el módulo agendadb (el módulo de acceso a datos creado antes) sólo si es ése el que vamos a utilizar. En el caso del módulo realizando sus funciones de auto-prueba (ejecutándose de manera autónoma), utilizamos este acceso a datos por la sencilla razón de que es el único que tenemos hasta ahora. En otras palabras, no hace

falta que el módulo agendadb esté disponible si disponemos de otro módulo de acceso a datos que sea conforme a la interfaz (no definida explícitamente, sino por su uso) de agendadb.

5. La capa de interfaz HTTP

A continuación se presenta el listado íntegro del módulo que implementa el servidor de aplicaciones.

Figura 2. La interfaz web




```
from BaseHTTPServer import BaseHTTPRequestHandler, HTTPServer,test
import cgi
import string


class AgendaHTTP(BaseHTTPRequestHandler):
def do_GET(self):
self.do_HEAD()
if self.response==404:
self.wfile.write('404 El método no existe' )
return
if self.metodo=='/busca':
try:
ret=db.busca(self.dicParams['param'][0])
```

```
except:
ret=[]
retHTML=”
for elem in ret:
retHTML=retHTML+’<li>%s = %s-%s’ % (elem[0],elem[1][0],elem[1][1])
self.wfile.write(”’<html><body>
<form><input name="param"> <input type="submit"></form>
<p>Resultados</p>
<ul>%s</ul>
</body></html>”’ % retHTML)
def do_HEAD(self):
if ’?’ in self.path:
self.metodo, params=string.split(self.path,’?’)
else:
self.metodo, params= self.path,”
self.dicParams=cgi.parse_qs(params)
if self.metodo in [’/busca’]:
self.response=200
else:
self.response=404
self.send_response(self.response)
self.send_header("Content-type", ’text/html’)
self.end_headers()
def run(db,host=’localhost’,port=8000):
    httpd = HTTPServer( (host, port) , AgendaHTTP)
    print "Serving HTTP on port", port, "..."
    httpd.serve_forever()

if __name__==’__main__’:
    import sys
    if sys.argv==[’sql’]:
        import agendaSQL
        import PoPy
        db=agendaSQL.agenda(PoPy,’dbname=template1’)
    else:
        import agendadb
        db=agendadb.agenda()
    run(db)
```

5.1. Comentarios al listado

El operador **%** inserta las variables indicadas en la cadena de plantilla. Las cadenas pueden abarcar varias líneas si se encierran entre comillas triples. Esto resulta tremendamente útil para copiar y pegar código HTML o SQL, presentar pantallas de ayuda de consola, etc.

Python dispone de módulos muy potentes, listos para usar, en la distribución estándar. Es especialmente potente en todo tipo de aplicaciones relacionadas con internet, XML, HTTP, CGI, etc.

El código presentado es compatible con la versión 1.5.2 (considerada un poco antigua, pero aún frecuente en entornos de producción). Python 2 permite una sintaxis más elegante en algunos aspectos.

El atributo **db**, al que se hace referencia dentro del método do_GET, no se asigna hasta el final del todo. En Python no hay ningún problema en una variable inexistente en la definición de las funciones, con tal de que la variable exista en el momento de ejecutar el código.

Zope es como el servidor que acabamos de hacer, en versión mastodóntica. Zope incluye controles de seguridad, mecanismos de herencia y adquisición complejos, base de datos de objetos propia, gestión por la web, un sistema de plantillas, servidores virtuales, gestión de caches y equilibrado de carga entre servidores. Salvo esto, es prácticamente lo mismo... :-)

Más cercano resultaría Webware. Webware no se aleja de la programación en Python tanto como Zope. Da unos servicios más limitados y mucho más ligeros. Resumiendo burdamente, si lo que deseas hacer lo puedes hacer en Zope, es posible que tardes un tiempo sorprendentemente corto. En el momento en que algo no se adapte a Zope, es mejor abandonar y echar mano de Webware.

6. Dinamismo en Python

La característica que quiero destacar en este documento es la gran ventaja de Python sobre los lenguajes de programación no dinámicos. Al no tener que declarar las variables, no las estamos limitando a la utilización en la que estaba pensando el programador. Como ejemplo, tenemos el caso del clásico juego de la vida. Se da el caso de que alguien (siento no dar referencias más concretas, pero todo esto apareció en usenet hace un par de años) hizo un mini-juego de la vida, con sus leyes de supervivencia, etc. Como respuesta al mensaje de usenet, otro programador envió una generalización a tres dimensiones del juego original, algo en lo que el primer autor ni siquiera había pensado. Esto fue posible porque cada vez que el autor decía, por ejemplo, distancia(a,b) no hacía ninguna suposición sobre los objetos a y b. Él pensaba que hablaba de puntos 2D, pero no le fue necesario hacerlo explícito en ningún momento, por lo que las reglas que construía eran generales. Desde luego, si el efecto es buscado, se le da más potencia a este hecho.

Otro ejemplo sería una función para sumar todos los elementos de una lista. Estamos pensando en valores numéricos, ¿no? Bien, la misma función es capaz de concatenar todas las cadenas de una lista, operar sobre números complejos, reales, enteros, o una mezcla de todos ellos. En realidad, sólo estamos exigiendo que los objetos (en Python, hasta los enteros son objetos) que le pasemos sea capaz de sumarse, y para ello basta con que tengan un método **__add__**.

Un ejemplo típico en Python es crear una clase que imita el comportamiento de los ficheros (al menos parcialmente). Hay muchas funciones en Python que esperan recibir como parámetro un objeto que cumpla la interfaz de los ficheros. Casos concretos: Funciones de depuración que vuelcan información a un fichero, codificadores, etc. Otro caso típico es el de las secuencias. Podremos hacer un bucle **for** sobre cualquier objeto que defina un método **__getitem__**. A esto se le llama emular la interfaz de las secuencias.

En nuestro caso concreto, veamos nuestra interfaz tk. ¿Qué le exigimos al parámetro tipo_db que pasamos al constructor de la clase principal? Debe ser invocable sin parámetros. Nosotros estábamos pensando en una clase, claro. Pues bien, no es necesario que tipo_db sea una clase. Podría ser cualquier función tal que, al llamarla sin parámetros, devolviera un objeto con los métodos **busca** y **encuentra**. Éstos últimos deben ser métodos que acepten un parámetro (en este caso, debe ser una cadena, pues utilizamos el parámetro en funciones que toman necesariamente una cadena como parámetro) y devuelvan, respectivamente, una secuencia o un elemento simple de la forma (clave, (nombre, dirección)).

En este caso, hemos intercambiado la interfaz de usuario, aunque sería posible cambiar el servidor de datos. Resultaría muy práctico especializar el servidor para utilizar internamente SQL, y hacer módulos para que dicho SQL se entendiera con ODBC en MS Windows, con JDBC en el caso de Jython sobre JVM o con módulos de acceso a datos db-api estándares de Python. Si se mantiene un buen diseño, seremos capaces de reutilizar todo nuestro código tanto como sea lógicamente posible. Comparado con C++, es como si todas nuestras clases fueran plantillas (templates), sin ningún esfuerzo adicional.

Esta eliminación de la burocracia tiene otras ventajas. Por ejemplo, es posible eliminar un método que ha quedado obsoleto de una clase. Sólo se quejarán los usuarios que utilicen dicho método. Es posible tambien redistribuir módulos aislados con funcionalidad añadida sin hacer nada sobre los clientes ya instalados, pues simplemente, no utilizarán las nuevas funciones, pero las antiguas seguirán funcionando tranquilamente. Nada de DLLs incompatibles en Python (sí, estoy pensando en las interfaces ActiveX). Como curiosidad, debo indicar que es muy sencillo en Python forzar la carga de unos módulos concretos e incluso tener una versión de Python completa instalada para cada programa concreto, si tenemos paranoia acerca de los componentes compartidos.

Otro aspecto, que no hemos utilizado aquí, del dinamismo de Python en ejecución es el acceso a parámetros y métodos por su nombre. Este nombre puede ser una variable, lo que nos permitiría, en el ejemplo del servidor de aplicaciones, tomar el nombre del documento solicitado por el navegador e invocar al método de dicho nombre. Sólo

habría que capturar los errores y añadir una capa de seguridad para tener un servidor de aplicaciones dinámico.

7. Listados adicionales

7.1. Acceso a BD relacional por PoPy

```
import string
class agenda:
    def __init__(self,modDB,info):
        self.db=modDB.connect(info)
        self.db.autoCommit(1)
    def volcado(self):
        ret=[]
        for k in self.db.keys():
            ret.append( (k,self.db[k]) )
        return ret

    def encuentra(self, clave):
        clave=string.upper(str(clave))
        cur=self.db.cursor()
        cur.execute("'select clave,nombre,direccion
                            from agenda
                            where clave='%s' "' % clave)
        k=cur.fetchall()
        cur.close()
        if k:
            k=k[0]
            return (k[0],(k[1],k[2]))
        else:
            return None

    def busca(self, clave):
        clave='%'+string.upper(str(clave))+'%'
        cur=self.db.cursor()
        cur.execute("'select clave,nombre,direccion
                            from agenda
                            where clave like '%s' "' % clave)
        ret=[]
        for k in cur.fetchall():
            ret.append( (k[0],(k[1],k[2])) )
        return ret
    def nuevo(self,clave,contenido):
        clave=string.upper(str(clave))
        nombre,direccion=contenido
        cur=self.db.cursor()
        cur.execute("'insert into agenda(clave,nombre,direccion)
                            values('%s','%s','%s')"' % (clave,nombre,direccion) )
        cur.close()


if __name__=='__main__':
    import PoPy

    import pprint
    ag=agenda(PoPy,'dbname=template1')
    try:
        cur=ag.db.cursor()
        cur.execute('select * from agenda')
        cur.close()
    except:
```

```
    print 'creando tabla agenda'
    cur=ag.db.cursor()
    cur.execute("""create table agenda(
                               clave varchar(255) not null,
                               nombre varchar(255),
                               direccion varchar(255) )""")
    print 'creando clave'
    cur=ag.db.cursor()
    cur.execute('create index agenda_pk on agenda(clave)')
    print 'insertando datos de prueba'
    ag.nuevo('ErnestoBKE', ('SQL Ernesto Molina','emolina@grupoburke.com'))
    ag.nuevo('Ernesto', ('SQL Ernesto Molina','rotoxl@jazzfree.com'))
    ag.nuevo('MarcosBKE', ('SQL Marcos Sánchez','msanchez@grupoburke.com'))
    ag.nuevo('Marcos', ('SQL Marcos Sánchez','rapto@arrakis.es'))
print ag.encuentra('Ernesto')
pprint.pprint (ag.busca('BKE'))
ag.db.close()
```

Bibliografía

Web de Python, disponible en http://www.python.org .

Documentación de Python en castellano, disponible en http://pyspanishdoc.sourceforge.net .

Comparativa de Python con varios lenguajes de programación, Esta comparativa es subjetiva, aunque no completamente tendenciosa. Es un problema real para hacer una comparativa poco apasionada el hecho de que hay pocos usuarios de Python descontentos con él. La dirección es http://www.python.org/doc/Comparisons.html .

Webware, un servidor de aplicaciones ligero en Python, Webware está ganando posiciones como alternativa a otros servidores de web más pesados (tipo Tomcat) o complejos (tipo Zope). Sus componentes son tan ligeros como el código fuente de ejemplo de este artículo. http://webware.sourceforge.net .

Zope, un servidor de aplicaciones complejo en Python, Zope es un servidor de arquitectura compleja y muy potente. Está más orientado a la gestión del contenido que a la programación clásica. Tiene mucho interés como solución de desarrollo llave en mano (no como webware, que exige un conocimiento considerable de Python). Zope dispone de muchos módulos de extensión (llamados productos) que lo hacen interesante por sí solos. En concreto, hay un producto llamado Squishdot

que permite montar una web a-la-Squishdot en menos de cinco minutos (y no es una forma de hablar, quiero decir 300 segundos). http://www.zope.org .

Vaults of Parnassus, catálogo de Python (en inglés), Los cofres del Parnaso recopilan todo lo existente para Python: Aplicaciones, bibliotecas de programador, documentación, etc. http://www.vex.net/parnassus .

Inmersión en Python (parcialmente traducido), Trata temas concretos: Introspección, objetos, tratamiento de X/HTML, pruebas unitarias y de regresión. http://diveintopython.org/index.html http://diveintopython.org/es/index.html (traducción parcial) .