# Pensando en Python (III): 3 en raya en la web

DIEGO LZ. DE IPIÑA GZ. DE ARTAZA (profesor del departamento de Ingeniería del Software de la facultad de Ingeniería (ESIDE) de la Universidad de Deusto)

En esta tercera entrega de la serie sobre Python aprenderemos a realizar aplicaciones web que acceden a bases de datos. Ilustraremos cómo Python representa una seria opción para el desarrollo de aplicaciones web mediante la extensión de la aplicación de tres en raya desarrollada en las dos entregas anteriores.

## Introducción

Dada la popularidad de Internet, el desarrollo de aplicaciones con interfaz web, agnósticas de la plataforma donde se ejecute el cliente web (navegador), ha experimentado un crecimiento exponencial. A pesar de la menor sofisticación de las interfaces web respecto a las interfaces gráficas de aplicaciones de sobremesa, su facilidad tanto de uso, dada la familiaridad de los usuarios con el paradigma web, como su simplicidad de desarrollo, son razones suficientes para considerarlas como primera opción en cualquier desarrollo que una empresa acomete hoy en día.

En las aplicaciones web, una sola instancia de la misma ejecutándose en un servidor web (o varias en un cluster de servidores), es compartida por varios (en algunos casos cientos o miles) de usuarios que acceden a ella desde un cliente universal, el navegador de Internet. El principal cometido de una aplicación web es crear páginas HTML (o en cualquier otro lenguaje de marcado) de manera dinámica. Es decir, los datos estáticos predefinidos de una página son combinados con datos generados por la lógica de la aplicación u obtenidos de fuentes de datos en tiempo real, por ejemplo de una base de datos. Durante la ejecución concurrente de las tareas requeridas simultáneamente por los diferentes usuarios de la aplicación web es frecuente que se detecten conflictos de acceso y modificación a datos en el servidor. Además, la aplicación web debe generar las páginas web, ejecutando toda la lógica de negocio subyacente, de la manera más rápida y eficiente posible. Es por tanto necesario, adoptar mecanismos en la parte servidora que agilicen los accesos a datos y soporten transacciones, es decir, conjuntos de operaciones físicas relacionadas sobre datos, que aparecen al cliente como una unidad lógica. Entran en juego aquí los llamados sistemas gestores de bases de datos (SGBD). La versión más comúnmente utilizada de los mismos son los sistemas relaciones de bases de datos. En definitiva, servidores web y bases de datos relacionales conforman un binomio esencial en la creación de toda aplicación web.

A continuación, explicaremos cómo se pueden crear aplicaciones web en Python que acceden a bases de datos relacionales. Como resultado de este proceso transformaremos la aplicación de sobremesa tres en raya desarrollada durante las dos anteriores entregas en una aplicación web.

**Crearemos una aplicación web que accederá a una base de datos**

## Programación web en Python

Al tratarse Python de un lenguaje de código abierto existen multitud de módulos/librerías para realizar cualquier tarea programática que imaginemos. En ocasiones, como es el caso de la programación web, existen varios módulos diferentes que de una manera más básica o sofisticada nos permiten llevar a cabo la misma tarea. Para comprobarlo no tenemos más que visitar el portal de Python, en su sección de Temas/web (http://www.python.org/topics/web/). Allí podemos encontrar un largo listado de estos módulos, detallando sus diferentes funciones. A continuación enumeramos los más destacados, clasificados según la categoría de plataforma de desarrollo de aplicación web a la que pertenecen:

- **Programación CGI Básica:**
  - Módulo CGI de la librería Standard de Python. CGI (Common Gateway Inteface) es un mecanismo estándar para la ejecución de código ejecutable por un servidor web y la obtención de los resultados de tal ejecución.
  - Módulo Cookie para la creación y procesamiento de cookies en aplicaciones web. Una cookie es un mecanismo para mantener estado entre las peticiones HTTP de una sesión web. Una cookie es una cabecera HTTP que permite la identificación unívoca en el servidor web del peticionario de la información.
- **Programación CGI Avanzada.** En CGI, un nuevo proceso es creado por cada petición HTTP recibida y eliminado cuando la petición es resuelta. La eficiencia es pobre. Esta es la razón por la que han

aparecido numerosas tecnologías que permiten una integración superior con el servidor web subyacente, y lo más importante, una mayor eficiencia. Ejemplos claros de estas tecnologías son PHP, Java Servlets y JSPs, y ASPs. En el caso particular de Python, la contribución más interesante a este respecto es mod_python:

- mod_python es un módulo Apache que integra el intérprete Python dentro del servidor, de modo que las aplicaciones pueden ejecutarse de manera más rápida que CGI, retiene datos persistentes entre las peticiones HTTP de una sesión y permite acceder a la parte interna de Apache.

- **Servidores de aplicaciones.** Van más allá de la simple generación de páginas dinámicas y asisten al programador en otras tareas de la programación de la lógica de negocio, tales como la persistencia de datos, la gestión de transacciones o la seguridad:
  - Webware facilita una suite de componentes Python para el desarrollo de aplicaciones web. Provee un servidor de aplicaciones, servlets, Python Server Pages (PSP), mapeo de objetos a bases de datos relacional y organizador de tareas. La arquitectura es muy modular, y permite añadir tus propias extensiones.
  - Zope, el Z Object Publishing Environment, proporciona una ORB (Object Request Broker) HTTP que permite publicar e invocar objetos Python en la web sin necesidad de ningún tipo de código CGI o HTTP específico. Objetos complicados pueden ser publicados con URLs que simulan jerarquías de objetos. Zope proporciona varios componentes que pueden usarse en concierto o separadamente. Algunos componentes incluidos son: un paquete de plantillas HTML, un sistema de persistencia de objetos, una plataforma para la gestión de objetos vía web, e incluso un servidor web sencillo.

En este artículo nos concentraremos en el módulo mod_python, que a juicio del autor representa la manera más sencilla y eficiente de implementar aplicaciones web en Python. Los servidores de aplicaciones como Zope representan una clara alternativa, y aunque también tienen un rendimiento tan elevado o superior a mod_python, su uso no es trivial y requerirían su exposición mucho más detallada.

## Integración Apache/Python: mod_python

Grisha Trubetskoy (http://www.ispol.com/home/grisha/) creó mod_python en 2003 con el propósito de ofrecer un módulo de extensión para el servidor web Apache (www.apache.org) que permitiese la generación de páginas dinámicas con Python de una manera más eficiente que el tradicional módulo CGI. mod_python empotra el intérprete de Python dentro del servidor Apache. Con mod_python puedes escribir aplicaciones web en Python que ejecutan muchas veces más rápido que los CGIs tradicionales, tienen acceso a características avanzadas tales como retener conexiones a bases de datos entre conexiones HTTP y permiten acceder a la parte interna de Apache. Con mod_python, el código Python se ejecuta directamente dentro del servidor Apache, sin necesidad de que éste tenga que lanzar procesos externos o delegar las peticiones a servidores de aplicaciones externos. mod_python se beneficia de las habilidades de Apache para aceptar y procesar peticiones entrantes de una manera escalable a la carga. El resultado es una plataforma que, según Trubetskoy, puede procesar peticiones más rápido que cualquier otra plataforma de desarrollo web en Python. mod_python es:

- Un módulo Apache recargable que empotra el intérprete de Python (libpython) proveyendo la habilidad de ejecutar código Python en el mismo proceso que Apache.
- Un manejador de las fases de procesamiento de las peticiones HTTP en Apache, que permite implementar cualquier fase en Python. También permite implementar filtros y manejadores de conexiones en Python. Un filtro HTTP intercepta de manera dinámica peticiones y respuestas para transformar o usar la información contenida en las mismas.
- Una interfaz a un subconjunto de las APIs de Apache, permitiendo la invocación de funciones internas del servidor desde Python. Esto permite acceder a información interna del servidor o de beneficiarse de facilidades del mismo como logeo.
- Una colección de herramientas para el desarrollo de aplicaciones web. Provee un conjunto de manejadores de peticiones: Publisher (mapea URLs a objetos y funciones dentro de módulos Python), PSP (permite la creación de Python Server Pages) y CGI (permite la programación web conformando con el estándar CGI). Cada uno de estos manejadores ofrece una manera diferente de desarrollar aplicaciones web, así como un conjunto de objetos y funciones de utilidad para el procesamiento de cookies, gestión de sesiones, y otros aspectos comunes en el desarrollo web.

El mayor incoveniente de mod_python es ser específico a Apache, a diferencia de JSPs y PHP que pueden integrarse con diferentes servidores web.

**LISTADO 1** Hay que incluir este código XML en el fichero httpd.conf

```
<Directory "<directorio-instalación-apache>/cgi-bin/python">
    SetHandler mod_python
    PythonHandler mod_python.publisher
    PythonDebug On
</Directory>

<Directory "<directorio-instalación-apache>/cgi-bin/psp">
    AddHandler mod_python .psp
    PythonHandler mod_python.psp
    PythonDebug On
</Directory>
```

**LISTADO 2** Código del fichero holasolop.psp

```
<html>
<%
if form.has_key('nombre'): # el objeto form es descrito en la siguiente sección
    saludo = '¡Hola, %s!' % form['nombre'].capitalize()
else:
    saludo = 'Hola solop!'
# end
%>
  <h1><%= saludo %></h1>
</html>
```

**LISTADO 3** Página de partida (index.html) de la aplicación web Tres en Raya

```
<html>
  <head>
    <meta name="description" content="">
    <meta name="keywords" content="">
    <title>Juego Tres en raya para Solop</title>
  </head>
  <frameset cols="100%" border="0">
    <frame src="/cgi-bin/psp/tresenraya.psp" name="tresenraya">
  </frameset>
</html>
```

**LISTADO 4** Lógica de control de /cgi-bin/psp/tresenraya.psp

```
<%
import tresenrayaweb
# hay que hacer login antes de poder jugar, verificamos si se ha pasado un nombreUsuario
if not form.has_key('nombreUsuario'):
   # solicitamos nombreUsuario en login.psp
   psp.redirect('/cgi-bin/psp/login.psp')
else:
   saludo = '¡Bienvenido %s a Tres en raya!' % form['nombreUsuario'].capitalize()
# end

# se crea un objeto partida y se cachea en sesión
if not session.has_key('partida'):
   session['partida'] = tresenrayaweb.JuegoTresEnRayaWeb()
# de momento no hay ganador
ganador = tresenrayaweb.NO_GANADOR
# si se pasan como parámetros las coordenadas x e y de una casilla, ésta se selecciona
if form.has_key('x') and form.has_key('y'):
   ganador = session['partida'].select_casilla(eval(form['x']), eval(form['y']))
# continua en listado 5 ...
```

**LISTADO 5** Código principal de web /cgi-bin/psp/tresenrayaweb.py

```
import tresenraya

NO_GANADOR = -1
EMPATE = 0
GANA_JUGADOR = 1
PIERDE_JUGADOR = 2

class JuegoTresEnRayaWeb(tresenraya.JuegoTresEnRaya):
   # falta el constructor y algunos métodos, mirar código completo en CD...
   def select_casilla(self, x, y):
      if not self.jugar(x, y):
         return self.ganador
      else:
         return NO_GANADOR

  def jugar(self, x, y):
     if self._JuegoTresEnRaya__casillasMarcadas < 9:
        if self._JuegoTresEnRaya__casillero[x][y] == None:
           self._JuegoTresEnRaya__casillero[x][y]=self._JuegoTresEnRaya__marcaJugador
           self._JuegoTresEnRaya__casillasMarcadas += 1
           self.__marcarCasilla(x, y, 'o.png')
           if self._JuegoTresEnRaya__hayGanador():
              self._JuegoTresEnRaya__registro.registrarVictoria(
                self._JuegoTresEnRaya__nombreUsuario)
              self.ganador = GANA_JUGADOR
              return False
           elif self._JuegoTresEnRaya__casillasMarcadas == 9:
              self._JuegoTresEnRaya__registro.registrarEmpate(
                self._JuegoTresEnRaya__nombreUsuario)
              self.ganador = EMPATE
              return False

           casillaIndex = self._JuegoTresEnRaya__elegirMarcaMaquina()
           self._JuegoTresEnRaya__casillero[casillaIndex[0]][casillaIndex[1]] =
             self._JuegoTresEnRaya__marcaMaquina
           self._JuegoTresEnRaya__casillasMarcadas += 1
           self.__marcarCasilla(casillaIndex[0], casillaIndex[1], 'x.png')
           if self._JuegoTresEnRaya__hayGanador():
              self._JuegoTresEnRaya__registro.registrarPerdida(
                 self._JuegoTresEnRaya__nombreUsuario)
              self.ganador = PIERDE_JUGADOR
              return False
           elif self._JuegoTresEnRaya__casillasMarcadas == 9:
           self._JuegoTresEnRaya__registro.registrarEmpate(
            self._JuegoTresEnRaya__nombreUsuario)
         self.ganador =  EMPATE
         return False
    return True
```

### Instalando Apache y mod_python

Para poder instalar mod_python es requisito imprescindible la previa instalación del intérprete de Python y el servidor web Apache. Detalles sobre cómo instalar Python tanto en Windows como UNIX fueron cubiertos en la primera entrega de esta serie.

Apache ha sido el servidor más popular en Internet desde 1996. En Octubre del 2003, 64% de los servidores web en Internet usaban Apache. Para obtener la última versión de Apache es necesario acceder a su página web: http://httpd.apache.org/. La mayoría de las distribuciones Linux ya traen preinstalado este servidor bien en su versión 1.3 o 2.0 (la última y la que utilizaremos en este artículo). Alternativamente, se recomienda que utilices tu gestor de paquetes Linux favorito para conseguir una copia o bájate el código fuente de http://httpd.apache.org/download.cgi y sigue las instrucciones de compilación incluidas. Si eres un usuario de Windows bájate el fichero de instalación apache_2.xxx-win32-x86-xxx.msi y haz doble clic sobre él (el CD-ROM adjunto incluye la versión de Apache para Windows).

mod_python puede obtenerse de http://httpd.apache.org/modules/python-download.cgi. Los usuarios de Windows deben ejecutar el fichero .exe suministrado que instalará mod_python en su sistema. Los usuarios de UNIX pueden bajarse el código fuente y las instrucciones de compilación del mismo lugar. En la preparación de este artículo hemos usado la versión 3.0 de mod_python (el CD-ROM adjunto incluye la versión de mod_python para Windows).

### Configurando Apache y mod_python

Una vez instalado Apache y mod_python tan sólo resta modificar el principal fichero de configuración de Apache, ubicado en <directorio-instalación-apache>/conf/httpd.conf:

**1.–** Al final del bloque de sentencias LoadModule añadir la siguiente:

```
LoadModule python_module modules/mod_python.so
```

**2.–** Después del bloque XML iniciado por el elemento Directory y correspondiente al directorio htdocs, directorio raíz del que cuelgan, por defecto, los documentos HTML estáticos en Apache, colocar los bloques Directory del listado 1, reemplazando <directorio-instalación-apache> por la ruta de instalación de Apache en tu máquina.

Con las definiciones mostradas en el listado 1 indicamos que debajo del subdirecto-

rio python de cgi-bin, directorio por defecto del que cuelgan los scripts CGI en Apache, colocaremos programas Python que generarán páginas HTML siguiendo el modelo del manejador de peticiones Publisher. Con la segunda directiva Directory indicamos que bajo el directorio psp colocaremos scripts en formato Python Server Pages (PSP).

### Python Server Pages en mod_python

Python Server Pages (PSP) es un mecanismo para incluir sentencias Python en documentos HTML o XML. El servidor interpreta el código Python incluido para producir el documento HTML o XML enviado al cliente. Este mecanismo se ha hecho popular a través de otras herramientas bien conocidas como JSP, PHP o ASP.

Es importante remarcar que mezclar código fuente Python en mitad de un documento HTML es objeto de cierta controversia. Algunos ingenieros del software consideran incluir código en mitad de un documento HTML una mala práctica de programación, al violar el paradigma Modelo-Vista-Controlador, introduciendo lógica de negocio en la capa de presentación. A pesar de no ser tan buena práctica, los millones de programadores PHP que exitosamente utilizan este paradigma demuestran que existe gran demanda por este estilo de programación web. No entraremos en discusiones dogmáticas en este artículo, esto es un artículo sobre Python y no sobre patrones de diseño.

La sintaxis de PSP es similar a la original de JSP, delimitando el código Python por medio de los símbolos <% y %>. De igual modo a JSP, PSP tiene cuatro tipos de entidades:

- **Código.** Representa el código fuente Python que contiene la lógica de cómo la salida final es producida. Está delimitado por los códigos de escape <% y %>.
- **Expresiones.** Es código fuente Python cuyo resultado en forma de un string forma parte de la salida final. Está delimitado por los códigos de escape <%= y %>.
- **Directivas.** Son instrucciones especiales para el procesador PSP. Son delimitadas por los códigos <%@ y %>. Actualmente mod_python sólo soporta la directiva <%@ include file='nombre-fichero'>, que reemplazará esta directiva por el contenido del fichero 'nombre-fichero'.
- **Comentarios.** En PSP, son eliminados por el procesador PSP, y son delimitados por los símbolos <%-- y --%>.

La parte más complicada de la programación PSP es la gestión de la tabulación sintáctica de Python. El intérprete de PSP recuerda la última tabulación Python usada a lo largo del código HTML, y deberemos ajustar nuestro código Python a la misma. Para simplificar esta tarea es recomendable usar comentarios que hagan el código más legible. Por ejemplo:

```
<%
if x == y:
    # comienzo
%>
```

**LISTADO 6** — Código de /cgi-bin/psp/login.psp

```
<%
import tresenrayaweb
import RegistroJugadoresDB

if not session.has_key('registro'):
   session['registro'] = RegistroJugadoresDB.RegistroJugadoresDB()

mensajeError = ""
if form.has_key('nombreUsuario') and form.has_key('clave'):
   try:
      session['registro'].login(form['nombreUsuario'], form['clave'])
      psp.redirect('/cgi-bin/psp/tresenraya.psp?nombreUsuario=' + form['nombreUsuario'])
   except:
      mensajeError = 'Los detalles de login introducidos son incorrectos'
saludo = 'Introduce tus detalles de logeo para jugar al Tres en raya'
# end
%>

<html>
<h1><%= saludo %></h1>
<form method="post" action="/cgi-bin/psp/login.psp">
        <table>
                <tr><td>Nombre usuario:</td>
                    <td><input type="text" name="nombreUsuario"></td></tr>
                <tr><td>Contraseña:</td><td><input type="password"
                                name="clave"></td></tr>
                <tr><td><input type="submit" value="Login"></td>
                    <td><input type="reset" name="Limpiar"></td></tr>
        </table>
</form>
<%
if len(mensajeError):
%>
   <p><%=mensajeError%></p>
<%
# end if
%>
</html>
```

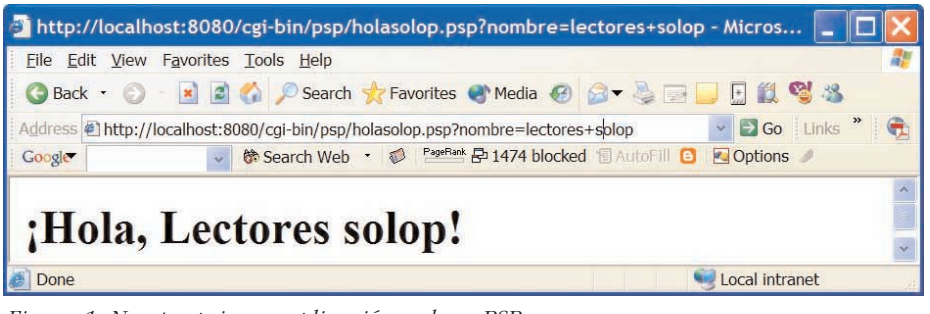


*Figura 1. Nuestra primera aplicación web en PSP.*

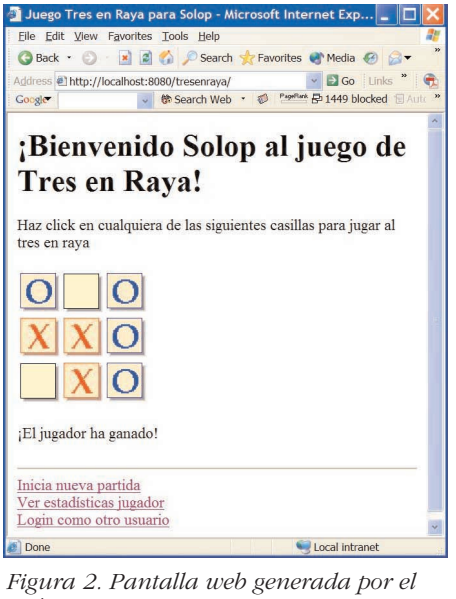


*Figura 2. Pantalla web generada por el script tresenraya.psp.*

```
... código html ...
<%
# fin
%>
```

### Hola Solop en el PSP de mod_python

En el listado 2 mostramos cómo se podría realizar una simple página PSP que reproduce el mensaje "Hola Solop" al invocar la URL http://localhost:8080/cgi-bin/psp/holasolop.psp. Si se le añade a la URL el sufijo "?nombre=lectores+Solop", el resultado generado sería "Hola, Lectores Solop". En la figura 1 se visualiza el resultado de realizar ésta última invocación.

### Variables globales

Existen varias variables accesibles en tiempo de ejecución de una página PSP. Estas variables pueden ser utilizadas directamente, sin habérseles asignado ningún valor previamente:

- **req,** el objeto Request de mod_python. Toda la funcionalidad avanzada de mod_python es disponible a través de este objeto en una página PSP.
- **psp,** corresponde a una instancia del objeto PSPInstance, que permite acceder a una API específica de PSP. Ofrece los siguientes métodos:
  - **set_error_page(filename),** permite especificar el nombre de una página PSP a invocar cuando ocurre un error en Python.
  - **apply_data(object),** invocará el objeto object, pasándole como argumentos los campos del formulario y devolviendo el resultado. Si estás familiarizado con JSP funciona como setProperty. Por ejemplo, dado el siguiente objeto:

```
class Coche:
    def __init__(self, color):
        self.color = color
```

Entonces, una página PSP invocada como resultado de una ejecución de un formulario conteniendo un campo con nombre color, podría hacer lo siguiente:

```
<%
coche = psp.apply_data(Coche)
%>
```

Esta sentencia invocaría el constructor del objeto Coche, pasando el valor del campo del formulario color. Como resultado, una instancia de Coche sería asignada a la variable coche.

- **redirect(location),** puede utilizarse para redireccionar entre páginas PSP. Debe invocarse como primera sentencia en una página, no se puede producir una redirección después de haber datos de salida al navegador.
- **form,** corresponde a los campos de un formulario en formato diccionario (objeto mod_python.FieldStorage).
- **session,** si PSP detecta que una página hace referencia a la variable session, automáticamente creará un objeto de tipo mod_python.Session, que generará cookies de sesión y permitirá mantener estado entre las invocaciones de una petición.

Para más información sobre los objetos definidos por mod_python, consultar su manual disponible en http://www.modpython.org/live/current/doc-html/.


*Figura 3. Pantalla web generada por script login.psp.*

## Añadiendo una interfaz web a la aplicación Tres en Raya

Una vez conocida la teoría básica sobre cómo crear y usar objetos mod_python dentro de un PSP, vamos a ilustrar su uso transforman-

**LISTADO 7** Código de /cgi-bin/psp/estadisticas.psp

```
<%
import tresenrayaweb
import RegistroJugadoresDB

# Es obligatorio hacer login para usar estadisticas.psp
if not form.has_key('nombreUsuario'):
   psp.redirect('/cgi-bin/psp/login.psp')
# end
%>

<html>
<%
estadisticasUsuario = session['registro'].getEstadisticas(form['nombreUsuario'])
%>
<h1>Las estadísticas de partidas jugadas por <%=form['nombreUsuario']%> son:</h1>
<table border="1">
   <tr><th>Ganadas</th><th>Empatadas</th><th>Perdidas</th></tr>
   <tr><td><%=estadisticasUsuario[0]%></td><td><%=estadisticasUsuario[1]%></td>
       <td><%=estadisticasUsuario[2]%></td></tr>
</table>

<hr>
<a href="/cgi-bin/psp/tresenraya.psp?nombreUsuario=<%=form['nombreUsuario']%>">Inicia
nueva partida</a><br>
<a href="/cgi-bin/psp/login.psp">Login como otro usuario</a>
</html>
```

**LISTADO 8** Código SQL para crear la base de datos tresenraya

```
CREATE DATABASE tresenraya;
GRANT ALTER, SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
            ON tresenraya.*
            TO tresenraya@localhost
            IDENTIFIED BY 'tresenraya';
CREATE TABLE usuario (
        nombreUsuario varchar(250) not null primary key,
        password varchar(250)
);
CREATE TABLE estadistica(
        nombreUsuario varchar(250) not null primary key references
                                                    usuario(nombreUsuario),
        ganadas int(11) default 0,
        empatadas int(11) default 0,
        perdidas int(11) default 0
);
```

LISTADO 9 Clase RegistroJugadoresDB

```
# importar módulos específicos de MySQL: MySQLdb
import MySQLdb, string, _mysql, _mysql_exceptions, tresenraya
class RegistroJugadoresDB(tresenraya.RegistroJugadores):
    def __init__(self):
        tresenraya.RegistroJugadores.__init__(self)
        db=MySQLdb.connect(host="localhost", user="tresenraya",
                           passwd="tresenraya", db="tresenraya")
        self.cursor = db.cursor()
        # Asegurarse que si no existe un usuario solop se añada
        usuarioSolop = self._executeSQLCommand(
"select * from usuario where nombreUsuario='solop'")
        # lógica para añadir usuario 'solop' si no existe

    def guardarEnDB(self):
        # Guardar por cada usuario sus detalles de login y sus estadísticas
        for usuario in self._RegistroJugadores__jugadores.keys():
            try:
                self._executeSQLCommand("insert into usuario values('" + usuario +
"', '" + self._RegistroJugadores__jugadores[usuario] + "')")
            except:
                self._executeSQLCommand("update usuario set password='" +
self._RegistroJugadores__jugadores[usuario] +
"' where nombreUsuario='" + usuario + "'")
            ...

    def _executeSQLCommand(self, command):
        # si la consulta devuelve resultados lo hará como una lista de tuplas (filas)
        # cada elemento de la tupla será una columna
        resultado = []
        command = string.strip(command)
        if len(command):
            try:
                resultCode = self.cursor.execute(command) # Ejecuta el comando
                if string.lower(command).startswith('select'): # si es una select ...
                    filas = self.cursor.fetchall() # recuperar todos los resultados
                    for fila in filas:
                        contenidoFila = []
                        for columna in fila:
                            if columna == None:
                                contenidoFila.append(None)
                            else:
                                contenidoFila.append(columna)
                        resultado.append(tuple(contenidoFila))
            except _mysql_exceptions.ProgrammingError, e:
                print e
                sys.exit()
        return resultado
```

do nuestra aplicación de tres en raya en una aplicación web.

En Apache los documentos estáticos se guardan, por defecto, en su directorio htdocs. Para nuestra aplicación crearemos un subdirectorio "tresenraya" debajo de htdocs, y colocaremos allí un fichero index.html que apunta al fichero .psp que actúa como controlador de la aplicación tres en raya. Además, crearemos un subdirectorio "images" donde guardaremos los ficheros .png que indican que una celda no está marcada (blank.png), está marcada por el jugador (o.png) o la máquina (x.png). El listado 3 muestra el fichero index.html que incrusta un marco (frame) que contiene otro marco de igual tamaño. El marco interior irá cambiando de contenido durante la ejecución de la aplicación, mientras que el marco exterior permanecerá haciendo que la URL que aparece en el navegador no cambie. Es un buen truco para evitar que el usuario de la aplicación web sepa qué PSP se está ejecutando en cada ocasión, e incluso con qué tecnología se ha desarrollado la aplicación.

Por otra parte, debajo del directorio cgi-bin, donde normalmente se colocan ficheros que generan documentos XML de manera dinámica, creamos el subdirectorio psp, en el que guardamos los siguientes ficheros:

- **tresenraya.psp:** script PSP que visualiza el tablero del juego de tres en raya y actúa también como controlador del resto de PSPs. Si un usuario intenta acceder a este PSP y no ha hecho login previamente será redirigido a login.psp. Por otro lado, si un usuario que ha hecho login desea ver las estadísticas de resultados de las partidas que ha jugado, este PSP le redirigirá a estadisticas.psp. El listado 4 muestra la lógica de control correspondiente a este script. Hace que se redireccione a otros PSPs o se invoque al método select_casilla del objeto JuegoTresEnRayaWeb, definido en tresenrayaweb.py. En la figura 2 podemos ver el resultado de haber invocado este PSP hasta completar una partida de tres en raya.
- **tresenrayaweb.py:** define la clase JuegoTresEnRayaWeb que hereda de la clase JuegoTresEnRaya, implementada en las anteriores entregas. Simplemente sobrescribe su método jugar() y declara un nuevo método público select_casilla. En tresenraya.psp se añade a la variable sesión una instancia de esta clase. El listado 5 muestra las partes más importantes de la implementación de esta clase.
- **login.psp:** script PSP que comprueba en primer lugar si existe una instancia del objeto de tipo RegistroJugadores (o una de sus subclases), implementada en las anteriores entregas, en la variable global session. Si no es así la crea. Si en la invocación se han pasado los parámetros nombreUsuario y clave, correspondientes a un usuario registrado, se redirecciona a tresenraya.psp. El listado 6 muestra el contenido de este PSP. Es importante resaltar que login.psp crea una instancia de la clase RegistroJugadoresDB que deriva de RegistroJugadores, aunque en esta ocasión guarda los datos en una base de datos. En la siguiente sección se describirá esta clase en más detalle. En la figura 3 se puede ver el resultado de invocar el script login.psp.
- **estadisticas.psp:** script PSP que visualiza en una tabla HTML las partidas ganadas, empa-


*Figura 4. Pantalla web generada por el script estadisticas.psp.*

LISTADO 10 Guardando estadísticas en /cgi-bin/psp/tresenraya.psp

```
<%
if ganador == tresenrayaweb.EMPATE:
   session['partida'].limpiar_tablero()
   session['registro'].registrarEmpate(form['nombreUsuario'])
   session['registro'].guardarEnDB()
%>
   <p>¡Ha habido empate!</p>
<%
elif ganador == tresenrayaweb.GANA_JUGADOR:
   session['partida'].limpiar_tablero()
   session['registro'].registrarVictoria(form['nombreUsuario'])
   session['registro'].guardarEnDB()
%>
   <p>¡El jugador ha ganado!</p>
<%
elif ganador == tresenrayaweb.PIERDE_JUGADOR:
   session['partida'].limpiar_tablero()
   session['registro'].registrarPerdida(form['nombreUsuario'])
   session['registro'].guardarEnDB()
%>
   <p>¡La máquina ha ganado!</p>
<%
# end if
%>
<hr>
<a href="/cgi-bin/psp/tresenraya.psp?nombreUsuario=<%=form['nombreUsuario']%>
                                                  ">Inicia nueva partida</a><br>
<a href="/cgi-bin/psp/estadisticas.psp?nombreUsuario=<%=form['nombreUsuario']%>
                                                  ">Ver estadísticas jugador</a><br>
<a href="/cgi-bin/psp/login.psp">Login como otro usuario</a>
</html>
```

tadas y perdidas por el jugador que ha hecho login. El listado 7 muestra el código fuente de este PSP. En la figura 4 se puede ver el resultado de invocar estadisticas.psp. En vez de producir una simple tabla HTML podríamos haber generado una imagen en formato PNG conteniendo un gráfico de barras como hicimos en la versión del programa en wxPython. Para ello podríamos haber utilizado por ejemplo el modulo matplotlib, disponible en http://matplotlib. sourceforge.net

## Programación de bases de datos en Python

Al igual que Java ofrece el estándar JDBC para la programación de bases de datos, Python hace lo propio con su DB-API. Con esta API Python consigue aislar el código fuente de la base de datos subyacente. El código Python se acomodará a los interfaces de la DB-API, mientras que una implementación de esta API para cada sistema de bases de datos, traducirá invocaciones de DB-API en llamadas de la base de datos. En la sección del portal Python, temas/basesdedatos (http://www.python.org/topics/database/) se puede encontrar información detallada sobre esta API.

### DB-API

A continuación enumeramos los principales interfaces ofrecidos por la DB-API:

- Para conectarnos a una base de datos usamos el método connect del módulo de base de datos utilizado que devuelve un objeto de tipo conection.
- El objeto conection ofrece el método cursor() que sirve para recuperar un cursor de la BD. Otros métodos definidos en connection son close(), commit(), rollback() y cursor().
- El objeto cursor define entre otros los siguientes métodos:
  - execute() nos permite enviar una sentencia SQL a la BD.
  - fetchone() recupera una fila.
  - fetchall() recuperar todas las filas.

Como se ha comentado, cada sistema relacional de bases de datos ha de ofrecer una implementación de DB-API. Dada la naturaleza de código abierto de Python, hay varios módulos que implementan este estándar:

- **DCOracle** (http://www.zope.org/Products/DCOracle/) creado por Zope para Oracle
- **MySQLdb** (http://sourceforge.net/projects/mysql-python) para MySQL
- **PyDB2** (http://sourceforge.net/projects/pydb2) para DB2.
- Y otros muchos más...

Nosotros utilizaremos la implementación de DB-API para MySQL, una de las bases de datos de código abierto más populares en Internet.

### MySQL y MySQLdb

MySQL fue creada por la empresa sueca MySQL AB (www.mysql.com). MySQL es una parte esencial del proyecto LAMP (Linux, Apache, MySQL, PHP/Perl/Python), una pila de herramientas de software abierto para desarrollar aplicaciones de empresa que está experimentando un gran crecimiento. Cada vez más compañías utilizan LAMP como una alternativa a software propietario.

La instalación de MySQL es muy sencilla tanto en Linux como Windows. En la siguiente URL se pueden obtener RPMs y ejecutables para instalar la última versión de MySQL (4.0) tanto en Linux como Windows: http://dev.mysql.com/downloads/mysql/4.0.html.

Por su parte, la instalación de MySQLdb, implementación para MySQL de la DB-API, es también trivial. Simplemente accede a su página web (http://sourceforge.net/projects/mysql-python) y usa el instalador provisto para Windows o el RPM para Linux.

## Usando MySQL para la aplicación Tres en Raya

Cómo último paso en nuestro proceso de mejora de la aplicación Tres en Raya, vamos a guardar los detalles de los jugadores autorizados y las estadísticas sobre los resultados de sus partidas en una base de datos. El listado 8 ilustra cómo crear en MySQL una base de datos de nombre tresenraya, a la que podemos hacer login de manera local con el usuario tresenraya y clave tresenraya, y que define las tablas usuario y estadistica.

Una vez creada la base de datos, creamos una nueva clase en Python a la que llamamos, RegistroJugadoresDB, que hereda de la clase RegistroJugadores que hemos ido utilizando y extendiendo desde el primer artículo. El listado 9 muestra un fragmento de la implementación de esta clase, dentro del módulo RegistroJugadoresDB. Obsérvese que el módulo MySQLdb es lo primero que se importa en este módulo. El método executeSQLCommand invoca toda operación SQL sobre la base de datos. Para más detalles, consultar el código incluido en el CD-ROM.

Finalmente sólo nos resta indicar cómo integrar el uso de esta clase dentro de nuestro código PSP. El listado 10 muestra cómo al finalizar una partida se procederá a guardar los detalles de los usuarios y los resultados de las partidas que han jugado, invocando el método de RegistroJugadoresDB, guardarEnDB().

## Conclusiones

En esta tercera entrega de la serie sobre Python hemos aprendido cómo desarrollar aplicaciones web que acceden a bases de datos a través de los módulos mod_python y MySQLdb, respectivamente. En la siguiente entrega concluiremos nuestro estudio de Python, ilustrando el potencial del mismo en el procesamiento de XML y mirando a dos primos hermanos de Python, JPython, la implementación Java de Python e IronPython, la implementación .NET de Python.

# ZOPE: EL SERVIDOR DE APLICACIONES WEB BASADO EN PYTHON


**Iniciamos una serie de artículos dedicados a este servidor de aplicaciones de código abierto que cuenta con versiones para las plataformas Windows, Solaris y Linux.**
**En esta primera parte, escribiremos una primera aplicación web de ejemplo.**

*Cada mes en tu quiosco*