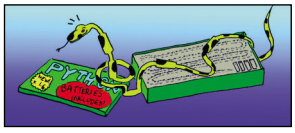
Pensando en Python (I): 3 en raya en modo texto

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 la serie de artículos que aquí comenzamos describiremos el lenguaje de programación Python. Para ilustrar sus virtudes desarrollaremos un juego de tres en raya, que según transcurra esta serie irá haciéndose cada vez más sofisticado. Pasará de ser un simple juego con interfaz en modo texto, a tener una interfaz gráfica, y de éste a ser una aplicación web que accede a una base de datos.

Introducción

En este primer artículo nos dedicaremos a introducir la sintaxis de Python y a iniciarnos en la programación orientada a objetos a través de Python. En un segundo artículo hablaremos de las facilidades provistas por Python para la programación de interfaces gráficas. En la tercera entrega explicaremos las bondades de Python para la programación de aplicaciones web con acceso a bases de datos. Finalmente, concluiremos este curso de Python en una cuarta entrega con la descripción de su primo hermano Jython. Python, el otro lenguaje de programación libre que empieza con 'P' y que no es ni Perl ni PHP, fue creado por Guido van Rossum (http://www.python.org/~guido/) en 1991. Dio este nombre al lenguaje inspirado por el popular grupo cómico británico Monty Python. Guido creó Python durante unas vacaciones de navidad en las que (al parecer) se estaba aburriendo.



Hola Mundo en Python

Como con cualquier otro lenguaje de programación, comencemos el aprendizaje de Python con el más simple de los ejemplos:

```
print "Hola Mundo" # "Hola Mundo"
```

En este primer extracto de código ya podemos observar la simplicidad de Python. Probablemente este es el ejemplo del programa "Hola Mundo" más sencillo que jamás has visto. La sentencia print imprime una cadena de caracteres a consola. Las cadenas de caracteres en Python se pueden denotar bien entre comillas dobles o comillas simples, por ejemplo 'Hola mundo'. Nótese que los comentarios de línea se indican con el símbolo #, mientras que los bloques de código pueden comentarse entre '"' y '"'.

Características de Python

Tras ver el primer extracto de código en Python, enumeremos sus principales características:

- Sintaxis elegante, minimalista y densa: En Python todo aquello innecesario no hay que escribirlo (;, {, }, \n). Además como veremos pocas líneas de código revierten en mucha funcionalidad.
- Moderno: Soporta objetos y estructuras de datos de alto nivel tales como cadenas de caracteres (strings), listas, diccionarios.
- Multi-paradigma: En vez de forzar a los programadores a adoptar un paradigma de programación único, Python permite el uso de programación orientada a objetos, programación estructurada o procedural e incluso programación funcional.
- Organizado y extendible: Dispone de múltiples formas de organizar código tales como funciones, clases, módulos, y paquetes. Si hay áreas que son lentas se pueden reemplazar por plugins en C o C++, siguiendo la API para extender o empotrar Python en una aplicación.
- Interpretado: No es necesario declarar constantes y variables antes de utilizarlas y no requiere paso de compilación/linkaje. La primera vez que se ejecuta un script de Python se compila y genera bytecode que es luego interpretado. Python es dinámico, encuentra errores de uso de tipos de datos en tiempo de ejecución y usa un recolector de basura para la gestión de memoria.
- Multiplataforma: Python genera código interoperable, que se puede ejecutar en múltiples plataformas (más aún que Java). Además, es posible embeber Python dentro de una JVM, como veremos en la cuarta entrega de esta serie.

C/Java vs Python

Código en C/Java	Código en Python

if (x) {
  if (y) {
  f1();
  }
  f2();
}
	if x:
	  if y:
	  f1()
	  f2()

- Open source: Razón por la cual la librería de módulos de Python (http://docs.python-.org/lib/lib.html) contiene un sinfín de módulos de utilidad y sigue creciendo continuamente.
- De propósito general: Toda aplicación programable con C# o Java también puede ser programada con Python.

Peculiaridades sintácticas

Python presenta unas diferencias sintácticas notables con respecto a otros lenguajes de programación. Usa tabulación (o espaciado) para mostrar una estructura de bloques. Se tabula una vez para indicar el comienzo de bloque y se des-tabula para indicar el final del bloque. Además, Python no usa el símbolo ';' para indicar el final de una sentencia sino simplemente un retorno de carro. El cuadro "C/Java vs. Python" compara un extracto de código en C/Java con otro de código en Python. Obsérvese que las peculiaridades sintácticas de Python traen consigo una reducción de los caracteres utilizados y líneas de código. Además, como efecto lateral, estas peculiaridades sintácticas obligan a diferentes programadores a escribir el código del mismo modo. No hace falta definir manuales de referencia de programación, para asegurarte que todos los programadores que intervienen en un proyecto utilizan las mismas convenciones. Como nota positiva, comentar que la mayoría de las palabras claves utilizadas por Python coinciden con las usadas en C o Java.

Dominios de aplicación de Python

Todo ingeniero de software, como buen artesano, debería dominar varios lenguajes de programación. Es inviable concebir que con un sólo lenguaje de programación podemos llevar a cabo todo tipo de desarrollos. Hay aplicaciones o parte de ellas que se adecuarán más a las características de un lenguaje de programación que otro. En este apartado vamos a resaltar las situaciones en las cuales deberíamos considerar el uso de Python y en cuales otras lo deberíamos desdeñar.

Python, al ser un lenguaje de scripting e interpretado no es adecuado para la programación de bajo nivel o de sistemas. Por ejemplo, la programación de drivers y kernels, dado que Python al ser de demasiado alto nivel, no tiene control directo sobre memoria y otras áreas de bajo nivel. Tampoco es adecuado para aplicaciones que requieren una alta capacidad de computo, tal es el caso de sistemas de procesamiento de imágenes. Para este tipo de aplicaciones el "viejo" C o C++ nunca podrá ser superado.

Python es ideal, sin embargo, como lenguaje "pegamento" para combinar varios componentes juntos, para llevar a cabo prototipos de sistemas, para la elaboración de aplicaciones cliente web o con interfaz gráfica, el desarrollo de sistemas distribuidos o la programación de tareas científicas, en las que hay que simular y prototipar rápidamente. Python ha sido utilizado para desarrollar muchos grandes proyectos de software como el servidor de aplicaciones Zope, el sistema de compartición de ficheros Mnet o parte de la implementación de Google. Actualmente, Python es uno de los tres lenguajes de programación que se usan en la framework de desarrollo de servidores de aplicaciones libres LAMP (Linux Apache MySQL y Python, Perl o PHP).

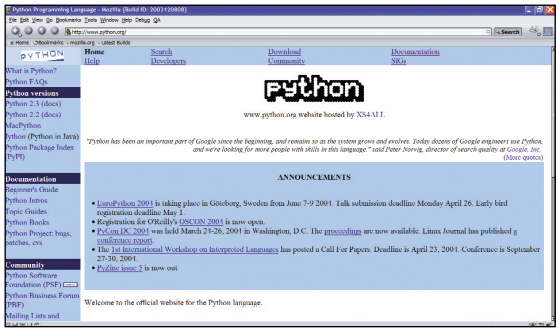


Figura 1.
Página principal de www.python.org.

Programando en Python

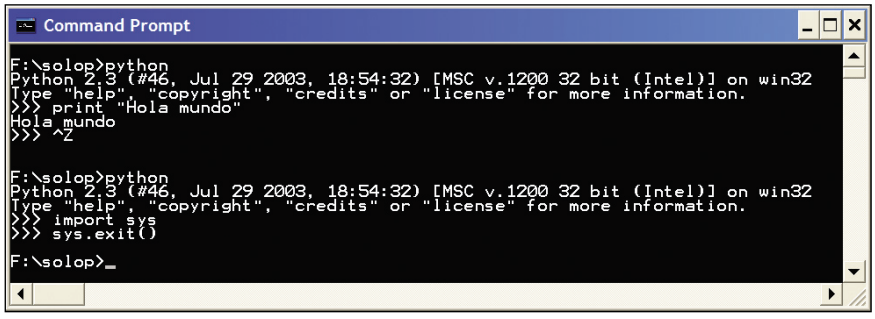
Una vez conocidas las características de Python es momento de empezar a usarlo. Para ello habremos de descargar una versión del mismo (actualmente 2.3.4) de la sección Download del portal de Python (http://www.python.org), mostrado en la figura 1 (también puede obtenerse del CD-ROM que acompaña la revista). La instalación para Windows es tan sencilla como hacer doble clic en el ejecutable descargado. Para Linux, podremos usar los rpms disponibles en http://www.python.org/2.3.3/rpms.html. El comando que te permitirá su instalación será algo similar a:

```
rpm -iv python2.3-2.3.4-pydotorg.i386.rpm
```

Modos de programación

Python se puede usar de dos modos diferentes: interactivamente o en modo script. Para arrancar Python en modo interactivo todo lo que se necesita es ejecutar en una consola de comandos (cmd en Windows o xterm en Linux) el intérprete de Python, esto es, el ejecutable python. Una vez arrancado el intérprete podrás introducir cualquier sentencia en Python. La figura 2 mues-

Figura 2.
Python en modo interactivo.



LISTADO 1 Nuestro primer bloque

```
# bloque.py
name = "Diego1" # asignación de valor a variable
if name == "Diego":
        print "Aupa Diego"
else:
        print "¿Quién eres?"
        print "¡No eres Diego!"
```

tra dos sesiones interactivas de Python. En la primera se ha codificado el ejemplo "Hola Mundo" mostrado con anterioridad. Para salir del intérprete se ha pulsado Ctrl-Z. Para realizar lo mismo en UNIX deberíamos haber pulsado Ctrl-D. Alternativamente, se podrían haber usado las dos sentencias mostradas en la segunda sesión interactiva de la figura 2. La primera importa el módulo sys (más detalles luego) y la segunda ejecuta el método exit() de ese módulo, que hace que el hilo de ejecución del programa finalice.
Para utilizar Python en modo script, se guardan las sentencias del programa en un fichero y luego se ejecuta el mismo escribiendo python <nombre-fichero.py>. Por ejemplo, en UNIX podríamos hacer lo siguiente para crear el fichero holamundo.py y luego ejecutarlo:

```
$ cat > holamundo.py
#!/usr/bin/env python
print "Hello World"
Ctrl^D
$ python holamundo.py
Hello World
```

Nótese que la sentencia:

```
#!/usr/bin/env python
```

se usaría en entornos UNIX para indicar al sistema que el intérprete Python se debería utilizar para ejecutar un fichero que tenga atributos de ejecución. En Windows es ignorada.

Sentencias y bloques

Para realizar un programa útil es necesario agrupar grupos de sentencias en unidades lógicas o bloques. Recuerda que en Python las sentencias se delimitan por el salto de línea y los bloques se indican por tabulación que sigue a una sentencia acabada en ':'. Por ejemplo, el código del listado 1 contiene la comparación de una variable con un valor que imprime diferentes mensajes según el resultado de su evaluación.
La ejecución del programa del listado 1 mediante el comando:

```
python bloque.py
```

generaría la siguiente salida:

```
¿Quién eres?
¡No eres Diego!
```

Identificadores

Los identificadores sirven para nombrar variables, funciones y módulos. Deben empezar con un carácter no numérico y contener letras, números y '_'. Python es sensible al uso de mayúsculas y minúsculas en identificadores. Las siguientes son palabras reservadas y no se pueden utilizar como identificadores: and, elif, global, or, assert, else, if, len, pass, break, except, import, print, class, exec, in, raise, continue, finally, is, return, def, for, lambda, try, del, from, not, while. Existen unos cuantos identificadores especiales con prefijo '__' para variables y funciones que corresponden a símbolos implícitamente definidos:

- __name__: nombre de función
- __doc__: documentación sobre una función
- __init__(): constructor de una clase

Variables y tipos de datos

Para declarar una variable en Python solamente es necesario darle un nombre y asignarle un valor. La variable es del tipo del valor asignado. Por defecto las variables son locales y accesibles solamente dentro del bloque de código donde han sido declaradas, para acceder a variables globales, es necesario preceder el nombre de la variable con el identificador global. Si no se quiere asignar ningún valor a una variable, se le puede asignar el valor None, equivalente a null en Java.
Numéricos. Python define los siguientes tipos numéricos: integer, long integer, floating-point, y complex. Sobre los valores numéricos se pueden aplicar los mismos operadores que en Java o C++, esto es, +, -, *, / y % (resto). Su uso se ilustra en los siguientes comandos escritos en modo interactivo:

```
>>> x = 4
>>> int (x)   # convierte x a entero
4
>>> long(x) # conviértelo a long
4L
>>> float(x) # a float
4.0
>>> complex (4, .2)
(4+0.2j)
```

Strings. Python soporta las cadenas de caracteres que son delimitadas por un par de ' o ". Dos strings juntos separados por un espacio se unen, y los separados por comas aparecen separados por un espacio:

```
>>> print "Hola" "Iñigo" # imprime HolaIñigo
>>> print "Hola", "Iñigo" # imprime Hola Iñigo
```

Se utilizan los mismos códigos de escape que en C y Java. Por ejemplo, para imprimir una nueva

línea utilizaríamos print '\n'. Las cadenas de caracteres cuyos caracteres especiales no queremos que sean interpretados se indican con el prefijo 'r' de raw.
Las cadenas de caracteres son objetos en Python a los cuales se les pueden aplicar algunas de las funciones predefinidas por Python, como len, que devuelve el número de elementos en una secuencia o los métodos que define tales como upper para convertir a mayúsculas y find para encontrar sub-cadenas en una cadena, entre otros muchos. Para más información tanto sobre la clase String como cualquier otra clase o módulo de Python es recomendable revisar la documentación de la librería estándar de Python (http://docs.python.org/lib/lib.html), cuya tabla de contenidos se muestra en la figura 3. A continuación mostramos algunas sentencias que hacen uso de strings:

```
>>> len('La vida es mucho mejor con Python.')
>>> 34
>>> 'La vida es mucho mejor con Python.'.upper()
'LA VIDA ES MUCHO MEJOR CON PYTHON'
>>> "La vida es mucho mejor con Python"
.find("Python")
27
>>> "La vida es mucho mejor con Python"
.find('Perl')
-1
>>> 'La vida es mucho mejor con Python'
.replace('Python', 'Jython')
'La vida es mucho mejor con Jython'
```

El carácter '%' representa al operador de formateo de cadenas, que se usa de una manera muy similar a los formateadores en C para la sentencia printf, tales como d para decimal, f para float o, x para hexadecimal:

```
>>> provincia = 'Álava'
>>> "La capital de %s es %s"  % (provincia,
"Vitoria-Gasteiz")
'La capital de Álava es Vitoria-Gasteiz'
```

Booleanos. Las constantes booleanas definidas por Python son True y False. El resultado de una condición siempre devuelve uno de esos dos valores.
Listas. Una lista representa una secuencia dinámica que puede crecer, y está indexada por un entero que comienza con el valor 0. Python define el operador ':' u operador de rodajas que permite indicar dando el índice inicial y final fragmentos del string a recuperar. Una lista es una clase definida por Python y por tanto tiene métodos para añadir (add) o insertar (insert) valores. Para borrar un elemento se utiliza la función predefinida del:

```
>>> meses = ["Enero", "Febrero"]
>>> print meses[0]
Enero
>>> meses.append("Marzo")
>>> print meses[1:2]
['Febrero']
>>> del meses[0]
>>> meses
['Febrero', 'Marzo']
```

Tuplas. Las tuplas son como las listas, pero no se pueden modificar. Son convenientes cuando queremos devolver varios valores como resultado de invocar a una función. Se definen con el operador (, por ejemplo, (1,2).
Diccionarios. Los diccionarios son arrays asociativos o mapas, indexados por una clave en vez de un índice numérico. La clave puede ser cualquier objeto Python, aunque normalmente es una tupla. Como se trata de una clase define una serie de métodos que podemos invocar tales como has_key. El operador in se utiliza para comprobar si un elemento forma parte de una secuencia:

```
>>> mydict = {"altura" : "media", "habilidad"
: "intermedia"}
>>> print mydict
{'altura': 'media', 'habilidad': 'interme
dia'}
>>> print mydict["habilidad"]
intermedia
>>>if mydict.has_key('altura'):
>>>     print 'Nodo encontrado'
Nodo encontrado
>>> if 'altura' in mydict:
>>>     print 'Nodo encontrado'
Nodo encontrado
```

Control de flujo

Condicionales. La sentencia if se utiliza para la definición de condiciones y puede ir seguida de varias partes elif y una else. En Python los operadores boleanos definidos son: or, and y not. Los operadores relacionales definidos son ==, >, < y =.

```
q = 4
h = 5
d = 3
if q < h:
   print "primer test pasado"
elif d<h:
   print "segundo test pasado"
```

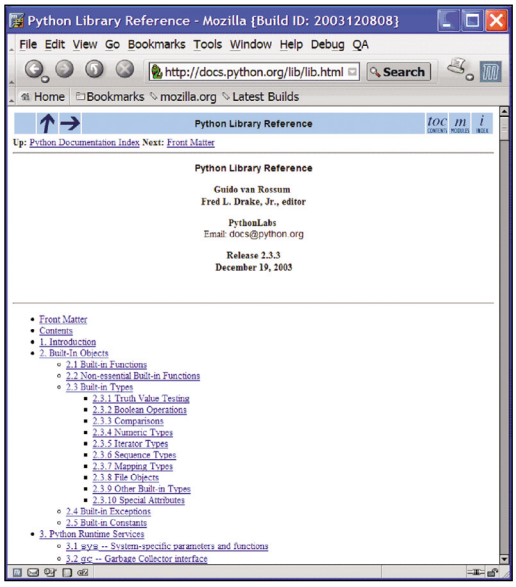


Figura 3.
Python Library Reference..

LISTADO 2 Un simple ejemplo de herencia

```
# clasesherencia.py
class Basic:
    def __init__(self, name):
        self.__name = name
    def show(self):
        print 'Basic -- name: %s' % self.__name
class Special(Basic): # entre paréntesis la clase base
    def __init__(self, name, edible):
        Basic.__init__(self, name)
        self.__upper = name.upper()
        self.__edible = edible
    def show(self):
        Basic.show(self)
        print 'Special -- upper name: %s.' % self.__upper,
        if self.__edible:
            print "It's edible."
        else:
            print "It's not edible."
    def edible(self):
        return self.__edible

obj1 = Basic('Manzana')
obj1.show()
print '=' * 30
obj2 = Special('Naranja', 1)
obj2.show()
print obj2._Basic__name
print obj2.__upper # lanzará una excepción
```

```
else:
    print "tercer test pasado"
```

```
>>> python condicional.py
primer test pasado
```

Bucles. Para la definición de bucles podemos usar el operador for o while. for se utiliza principalmente para iterar sobre los miembros de una secuencia: listas, tuplas o diccionarios. for se utiliza a menudo con la función predefinida range, que crea una secuencia descrita por ([start,] end [,step]), donde los campos start y step son opcionales. start es 0 y step es 1 por defecto:

```
>>> for x in range(1,5):
>>>     print x
>>> 1 2 3 4
```

La sentencia while es otra sentencia de repetición. Se utiliza para ejecutar un bloque de código hasta que una condición sea falsa. La sentencia break se utiliza a menudo dentro de un bucle while y sirve para salir del bucle:

```
>>> reply = 'repite'
>>> while reply == 'repite':
...     print 'Hola'
...     reply = raw_input('Introduce "repite" para hacerlo de nuevo: ')
...
Hola
Introduce "repite" para hacerlo de nuevo: adiós
>>>
```

Funciones

Una función se declara usando la palabra clave def, seguida del nombre de la función y entre paréntesis una lista de argumentos. Los argumentos si corresponden a tipos numéricos, booleanos o strings se pasan por valor y si corresponden a secuencias por referencia. A una función se le pueden asignar parámetros por defecto, preasignado un valor a cada parámetro de una función. Por ejemplo:

```
def myfunc(a,b=2):
    sum = a + b
    return sum
print myfunc(2) # devuelve el valor 4
```

A una función se le puede pasar un número variable de argumentos (argumento con prefijo *) y argumentos basados en palabras clave no predefinidas (argumento con prefijo **):

```
def testArgLists_1(*args, **kwargs):
    print 'args:', args
    print 'kwargs:', kwargs
testArgLists_1('aaa', 'bbb', arg1='ccc', arg2='ddd')
```

que visualizaría:

```
args: ('aaa', 'bbb')
kwargs: {'arg1': 'ccc', 'arg2': 'ddd'}
```

Clases

Las clases en Python son creadas usando la sentencia class. Una clase contiene una colección de métodos. Cada método contiene como primer parámetro (self) que hace referencia a la instancia de la clase, es equivalente al this de C++ o Java. Python soporta múltiple herencia de clases. Existe un soporte limitado para variables privadas, usando una técnica llamada name mangling. Todo identificador de la forma __spam es textualmente reemplazado por _classname__spam. Sin embargo, el identificador todavía podría ser accedido por código que usa la instancia de la clase mediante _classname__spam. El listado 2 muestra un simple ejemplo de herencia.

Excepciones

Cada vez que un error ocurre se lanza una excepción, visualizándose un extracto de la pila del sistema. La ejecución de la última sentencia del ejemplo anterior produciría:

```
Traceback (most recent call last):
  File "clasesherencia.py", line 28, in ?
    print obj2.__upper
AttributeError: Special instance has no attribute '__upper'
```

Para capturar una excepción se usa except y para lanzar una excepción se usa raise. Se pueden crear excepciones personalizadas cre-

ando una nueva clase que derive de la clase RuntimeError definida por Python. El siguiente fragmento ilustra el uso de excepciones:

```
# excepcion.py
try:
    fh=open("new.txt", "r")
except IOError, e:
    print e
$ python excepcion.py
[Errno 2] No such file or directory:
'new.txt'
```

Módulos y paquetes

Las librerías de métodos y clases se agrupan en Python en módulos. Un módulo es una colección de métodos o clases en un fichero que acaba en .py. El nombre del fichero determina el nombre del módulo en la mayoría de los casos. Para usar un módulo se usa la sentencia:

```
import <nombre-modulo>
```

que hace que un módulo y su contenido sean disponibles para su uso. Alternativamente se puede usar la sentencia:

```
from <nombre-module> import <elementos a
importar separados por comas o * para todo
el contenido>
```

Por ejemplo, un módulo se podría definir como:

```
# modulo.py
def one(a):
    print "in one"
def two (c):
    print "in two"
```

Este modulo se usaría como sigue. Nótese la diferencia que existe entre el uso de la cláusula import y la from:

```
>>> import modulo
>>> modulo.one(2)
in one
>>> from modulo import *
>>> one(2)
in one
>>>
```

Un conjunto de módulos puede agruparse como una unidad representada por un paquete. En la siguiente entrega veremos un ejemplo de creación y uso de paquetes.

LISTADO 3 La clase RegistroJugadores

```
# imports para toda la aplicación
import os
import sys
import random
from types import *

class RegistroJugadores:
    def __init__(self):
        self.__jugadores = {'solop':'solop'}
        self.__estadisticas = {'solop':[0, 0, 0]} # jugador -> [ganados, empa
tados, perdidos]

    def registrarJugador(self, jugador, clave):
        if len(trim(jugador)) == 0 or len(trim(clave)) == 0:
            raise "Los campos jugador y clave no pueden estar vacios"
        if self.__jugadores.has_key(jugador):
            raise "Jugador " + jugador + " ya ha sido registrado!"
        self.__jugadores[jugador] = clave
        self.__estadisticas[jugador] = [0, 0, 0]

    def login(self, jugador, clave):
        if not self.__jugadores.has_key(jugador):
            raise "Jugador " + jugador + " no registrado!"

        if not self.__jugadores[jugador] == clave:
            raise "Clave de jugador " + jugador + " es invalida"
        return True

    def registrarVictoria(self, userName):
        self.__estadisticas[userName][0] += 1

    def registrarEmpate(self, userName):
        self.__estadisticas[userName][1] += 1

    def registrarPerdida(self, userName):
        self.__estadisticas[userName][2] += 1

    def getEstadisticas(self, userName):
        return self.__estadisticas[userName]
```

Desarrollando tu primera aplicación

Para poner en práctica la sintaxis del lenguaje Python vamos a desarrollar una simple aplicación de tres en raya. Aparte de permitirnos jugar contra la máquina, este juego requiere que el jugador se autentifique antes de jugar y además guarda estadísticas en memoria sobre las partidas ganadas, perdidas o empatadas por cada jugador. Por el momento la máquina usa un algoritmo sencillísimo para la elección de la casilla a tachar por la máquina, la elige de manera aleatoria. En próximas entregas de esta serie discutiremos algoritmos mejores basados en inteligencia artificial. El listado 3 muestra la clase RegistroJugadores que es utilizada para mantener las estadísticas de las partidas de los jugadores.

El constructor de RegistroJugadores define dos atributos privados de tipo diccionario que guardan los jugadores registrados (self.__jugadores) y las estadísticas de las partidas (self.__estadisticas). Se carga un sólo jugador con nombre de usuario "solop" y contraseña "solop", e inicializamos las estadísticas para el jugador 'solop', con la lista [0,0,0] que indica que el usuario no ha ganado, ni empatado, ni perdido ningún juego todavía. La clase RegistroJugadores define el método registarJugador que permita registrar los detalles de login de un nuevo jugador. Si ya existe un usuario con el nombre de usuario pasado, se lanza una excepción y si no se crea una nueva entrada en el atributo self.__jugadores. El método login permite la autenticación de un usuario, y en caso de proveerse un nombre de usuario o clave erróneas se lanza una excepción. Por su parte, los métodos registrarVictoria, registrarEmpate y registrarPerdida sirven para modificar el atributo self.__estadisticas con los resultados de las partidas de cada jugador. Finalmente el método getEstadisticas devuelve una lista con los resultados obtenidos por un jugador.

La clase JuegoTresEnRaya (véase la figura 4), que define la lógica principal de la aplicación, no se ha incluido en estas páginas por razones de espacio, sin embargo el código de dicha clase puede encontrarse en el CD-ROM. El constructor de esta clase recibe como parámetros una referencia al registro de jugadores, el nombre de usuario y la clave del jugador que

LISTADO 4

El bloque main

```
if __name__ == '__main__':
    registro = RegistroJugadores()
    juego = JuegoTresEnRaya(registro, sys.argv[1], sys.argv[2])
    juego.jugar()
    while True:
        s = raw_input("¿Quieres jugar otra vez? (s) o (n): ")
        if s.strip().lower() == 's':
            juego = JuegoTresEnRaya(registro, 'solop', 'solop')
            juego.jugar()
        else:
            print 'Gracias por jugar al tres en raya!'
            print 'Estadísticas de juego de solop: victorias (' + str(registro.getEstadisticas('solop')[0]) +\
                  ') - empates (' + str(registro.getEstadisticas('solop')[1]) + \
                  ') - derrotas (' + str(registro.getEstadisticas('solop')[2]) + ')'
            break
```

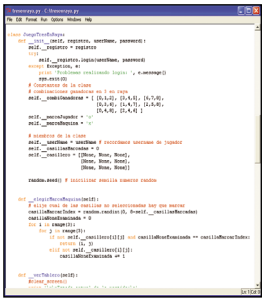


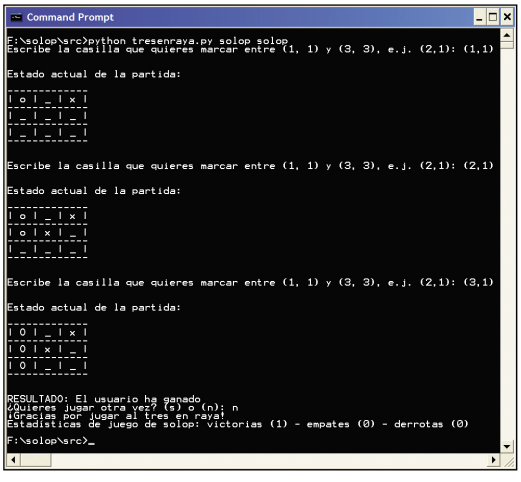
Figura 4.
La clase JuegoTresEnRaya

quiere competir con la máquina. El constructor en primer lugar guarda la referencia al registro en la variable privada self.__registro y valida que los datos de autentificación del jugador son correctos. A continuación, declara las constantes self.__combiGanadoras con las combinaciones ganadoras del juego, self.__marcaJugador con el carácter 'o' que indica la marca de un jugador y self.__marcaMaquina con el valor 'x' que denota las casillas marcadas por la máquina. Finalmente, el constructor declara las variables self.__userName donde memoriza el identificador del jugador, self.__casillasMarcadas donde cuenta cuántas casillas están marcadas hasta el momento y el array bidimensional self.__casillero que recoge los valores marcados en cada casilla, inicialmente ninguno. Esta clase también declara los siguientes métodos de ayuda privados: __elegirMarcaMaquina, __verTablero y __hayGanador. El método __elegirMarcaMaquina hace uso del módulo random para elegir de manera aleatoria una de las casillas no marcadas. El módulo __verTablero visualiza en modo texto el tablero. Finalmente, el método __hayGanador determina si alguna de las combinaciones ganadoras ha sido alcanzada.

La clase JuegoTresEnRaya sólo define el método público jugar() que define toda la lógica del juego de tres en raya entre un usuario y la máquina. Asume que el usuario siempre comienza. La partida no concluye hasta que se ha encontrado una combinación ganadora o se han tachado las nueve casillas del tablero. Mientras el usuario no introduce los datos de una casilla correctamente el método itera sobre un bucle. Para recoger la entrada del usuario se utiliza la función raw_string(<mensaje>). Además dado que se pretende que la entrada tenga el formato (x, y), se evalúa si los datos introducidos corresponden a una tupla, mediante la función eval que convierte un string al tipo de datos representado. Además se valida si el valor interpretado corresponde a una tupla haciendo uso del método type del módulo types y del operador is que permite la comparación entre un valor y su tipo: if not type(casillaCoords) is TupleType. Además en cada iteración se comprueba que el valor de la celda introducida esté dentro del rango de valores validos y además no haya sido ya marcado. Después de que el usuario introduzca una casilla válida se evalúa si ha ganado invocando el método __hayGanador. Si es así se registra la victoria del usuario invocando self.__registro.registrarVictoria(self.__userName). Si todavía no se han marcado las nueve casillas del tablero se pasa el turno a la máquina, eligiéndose la casilla a marcar mediante la llamada al método privado __elegirMarcaMaquina. A continuación, se comprueba si tras marcar la última casilla la máquina ha ganado la partida y si es así se registra que el usuario la ha perdido. Cuando las casillas marcadas sean nueve y no ha habido ningún ganador se registrará el hecho de que el juego ha acabado en empate.

El listado 4 muestra el bloque de entrada del programa. Todo bloque que comience con if __name__ == '__main__':, es equivalente a un bloque main en C o Java. El bloque main, en primer lugar, obtiene una referencia al registro. En futuras entregas recuperaremos ese registro bien de un fichero o de una base de datos. Por el momento, esto se consigue invocando al constructor de RegistroJugadores que mantiene un diccionario en memoria con los jugadores registrados y otro con las estadísticas de juego de los mismos. En segundo lugar, el bloque main crea una instancia de la clase JuegoTresEnRaya, pasando como referencia el registro obtenido y el nombre y contraseña del jugador. Estos dos últimos valores son recuperados a partir de los parámetros pasados en la invocación del programa, mediante la sentencia sys.argv[x], donde x indica el orden del parámetro, apuntando el índice 0 al nombre del script. A continuación, se permite que el usuario inicie una partida llamando al método jugar() de la instancia de JuegoTresEnRaya. Al finalizar esta partida se pregunta al usuario si quiere seguir jugando y se repite este proceso hasta que el usuario no responda "s." Finalmente, antes de terminar el programa se muestran las estadísticas de resultados de los enfrentamientos entre la máquina y el usuario indicado. La figura 5 muestra una sesión del juego de tres en raya.

Figura 5.
Ejecución del juego tres en raya.



Conclusiónes

Este artículo ha mostrado la sintaxis del lenguaje Python y ha ilustrado su uso a través de la elaboración de un simple juego de tres en raya. A través de la elaboración de este pequeño programa hemos podido comprobar la simplicidad de Python y sus capacidades para construir de manera rápida cualquier tarea de programación que tengamos en mente. En la siguiente entrega de esta serie veremos cómo proporcionar una interfaz gráfica a este juego y cómo mantener el registro de jugadores y sus estadísticas en varios ficheros.