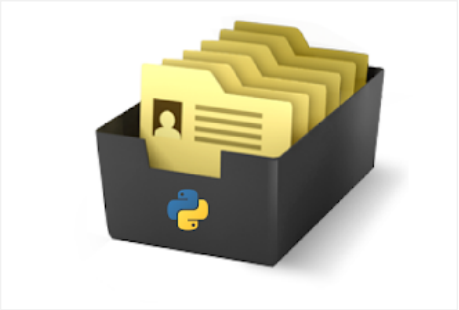
jueves, 14 de mayo de 2015

Bases de datos documentales con TinyDB (I)



Introducción

TinyDB es una base de datos documental idónea para proyectos pequeños o personales, fácil de usar y muy versátil.

Para trabajar con esta base de datos es necesario instalar los módulos escritos íntegramente en Python del alemán **Markus Siemens**. Dichos módulos no dependen de otro software para su funcionamiento y están disponibles tanto para Python 2.x como 3.x.

En relación al formato en que se guardan las bases de datos TinyDB indicar que, por defecto, se utiliza **JSON**. Además, TinyDB permite usar la memoria convencional como soporte de almacenamiento para las mismas.

Instalación con Pip

Para instalar TinyDB desde la consola:

En Linux:

$ sudo pip3 install tinydb

En Windows:

C:\> pip3 install tinydb

También, es posible descargar y descomprimir el archivo del módulo y después instalarlo con:

$ python3 setup.py install

Operaciones básicas con TinyDB

Para realizar los siguientes ejemplos recomendamos el entorno interactivo de Python u otro alternativo como IPython.

Importar módulos para realizar operaciones básicas

```
from tinydb import TinyDB, where
```

Abrir/Crear base de datos. Tabla por defecto ("_default")

```
cielo = TinyDB('cielo.db')
```

Abre la base de datos indicada o la crea si no existe. Cuando se crea una base de datos se agrega, automáticamente, una tabla llamada "**_default**". A continuación, si se insertan registros o se realizan otras operaciones propias de tablas se aplicarán a la misma.

Buscar

Buscar

Python para impacientes

Python
IPython
EasyGUI
Tkinter
JupyterLab
Numpy

Anexos

Guía urgente de MySQL
Guía rápida de SQLite3

Entradas + populares

Dar color a las salidas en la consola
En Python para dar color a las salidas en la consola (o en la terminal de texto) existen varias posibilidades. Hay un método basado ...

Instalación de Python, paso a paso
Instalación de Python 3.6 A finales de 2016 se produjo el lanzamiento de Python 3.6 . El propósito de esta entrada es mostrar, pas...

Añadir, consultar, modificar y suprimir elementos en Numpy
Acceder a los elementos de un array. [], [,], ... Acceder a un elemento de un array. Para acceder a un elemento se utiliz...

Variables de control en Tkinter
Variables de control Las variables de control son objetos especiales que se asocian a los widgets para almacenar sus valore...

Cálculo con arrays Numpy
Numpy ofrece todo lo necesario para obtener un buen rendimiento cuando se trata de hacer cálculos con arrays. Por como está concebid...

Tkinter: interfaces gráficas en Python
Introducción Con Python hay muchas posibilidades para programar una interfaz gráfica de usuario (GUI) pero Tkinter es fácil d...

Operaciones con fechas y horas. Calendarios
Los módulos datetime y calendar amplían las posibilidades del módulo time que provee funciones para manipular expresiones de ti...

Convertir, copiar, ordenar, unir y dividir arrays Numpy
Esta entrada trata sobre algunos métodos que se utilizan en Numpy para convertir listas en arrays y viceversa; para copiar arrays d...

Tkinter: Tipos de ventanas
Ventanas de aplicación y de diálogos En la entrada anterior tratamos los distintos gestores de geometría que se utilizan para di...

Threading: programación con hilos (I)
En programación, la técnica que permite que una aplicación ejecute

Abrir/Crear una tabla con un nombre determinado

```
constelacion = cielo.table('constelacion')
```

Abre la tabla indicada o la crea si no existe en la base de datos. Después de abrir una base de datos si no se abre una tabla específica se abrirá la tabla "**_default**", quedando disponible para operar con ella.

Obtener los nombres de las tablas de una base de datos

```
tablas = cielo.tables()
tablas  #  {'_default', 'constelacion'}
```

Insertar registros en la tabla 'constelacion'

```
constelacion.insert({'nombre': 'Casiopea',
                     'abrev': 'CAS',
                     'sup': 598, 'obs':
                     'Todo el año'})  # 1
constelacion.insert({'nombre': 'Cefeo',
                     'abrev': 'CEP',
                     'sup': 500, 'obs':
                     'Todo el año'})  # 2
```

Cuando se añaden registros a una tabla, TinyDB les asigna un identificador (ID) numérico a cada registro.

Consultar todos los registros de la tabla 'constelacion'

```
registros = constelacion.all()
for registro in registros:
    print(registro['nombre'],
          registro['abrev'],
          registro['sup'],
          registro['obs'])
```

Casiopea CAS 598 Todo el año
Cefeo CEP 500 Todo el año

Consultar todos los registros que cumplan una condición

```
registros = constelacion.search(where('obs') == 'Todo el año')
for registro in registros:
    print(registro['nombre'],
          registro['abrev'],
          registro['sup'],
          registro['obs'])
```

Casiopea CAS 598 Todo el año
Cefeo CEP 500 Todo el año

La función **search()** devuelve una lista con los registros que cumplan la condición indicada. El primer registro encontrado se corresponderá con el elemento 0 de la lista, el segundo con el 1 y así, sucesivamente.

Consultar el primer registro que cumpla una condición

```
registros = constelacion.search(where('obs') == 'Todo el año')
print(registros[0]['nombre'],
      registros[0]['abrev'],
      registros[0]['sup'],
      registros[0]['obs'])
```

Casiopea CAS 598 Todo el año

En el ejemplo se hace referencia al elemento 0 de la lista que se corresponde con el primer registro encontrado. Si no existen registros que cumplan la condición, al intentar visualizar se producirá una excepción del tipo **IndexError** que es necesario capturar:

simultáneamente varias operaciones en el mismo espacio de proceso se...

Archivo

mayo 2015 (3)

python.org

pypi.org

Sitios

- ActivePython
- Anaconda
- Bpython
- Django
- Flask
- Ipython
- IronPython
- Matplotlib
- MicroPython
- Numpy
- Pandas
- Pillow
- PortablePython
- PyBrain
- PyCharm
- PyDev
- PyGame
- Pypi
- PyPy
- Pyramid
- Python.org
- PyTorch
- SciPy.org
- Spyder
- Tensorflow
- TurboGears

```
try:
    registros = constelacion.search(where('obs') == 'Todo el mes')
    print(registros[0]['nombre'],
          registros[0]['abrev'],
          registros[0]['sup'],
          registros[0]['obs'])

except IndexError:
    print("No existen registros")
```

No existen registros

Actualizar un dato de un registro que cumpla una condición

```
constelacion.update({'sup': 588}, where('nombre') == 'Cefeo')
constelacion.all()
```

[{'abrev': 'CAS', 'nombre': 'Casiopea', 'obs': 'Todo el año', 'sup': 598},
{'abrev': 'CEP', 'nombre': 'Cepeo', 'obs': 'Todo el año', 'sup': 588}]

En el registro de la constelación '*Cefeo*' se actualiza el campo de la superficie 'sup' con el valor *588*.

Suprimir los registros que cumplan una condición

```
constelacion.remove(where('sup') > 590)
constelacion.all()
```

[{'abrev': 'CEP', 'nombre': 'Cepeo', 'obs': 'Todo el año', 'sup': 588}]

En la tabla "**constelacion**" se borra el registro cuya superficie supera el valor de *590* grados cuadrados, es decir, en este caso el de la constelación '*Casiopea*' que ocupa una superficie mayor, de *598* grados cuadrados.

Añadir un nuevo registro a la tabla 'constelacion'

```
constelacion.insert({'nombre': 'Dragón',
                     'abrev': 'DRA',
                     'sup': 1083,
                     'obs': 'Todo el año'})  # 3
```

Mostrar el contenido del archivo de base de datos (cielo.db) desde la consola

En Linux:

```
$ cat cielo.db
```

En Windows:

```
c:\>type cielo.db
```

Contenido:

{"_default": {}, "constelacion": {"2": {"sup": 500, "obs": "Todo el a\u00f1o", "abrev": "CEP", "nombre": "Cepeo"}, "3": {"sup": 1083, "obs": "Todo el a\u00f1o", "abrev": "DRA", "nombre": "Drag\u00f3n"}}}

Como puede observarse el formato utilizado para almacenar los datos es totalmente abierto.

La base de datos del ejemplo es un diccionario con dos pares de claves/valor.

Las claves se corresponden con los nombres de las tablas disponibles: "**_default**" y "**constelacion**".

Los valores se corresponden con los datos de cada tabla. Éstos están organizados en una serie de diccionarios anidados (uno por cada registro). En el caso de la tabla "**_default**" las dos llaves **{}** indican que la tabla está vacía. Sin embargo, la tabla "**constelación**" contiene dos registros identificados con los ID "**2**" y "**3**" con sus campos/datos asociados contenidos igualmente en otro diccionario.

Borrar todos los registros de una tabla

```
constelacion.purge()
```

Borrar todas las tablas de la base de datos

```
cielo.purge_tables()
```

Cerrar una base de datos

```
cielo.close()
```

En el próximo capítulo: Uso avanzado de TinyDB

Ir al índice del tutorial de Python

Publicado por Pherkad en 14:10

Etiquetas: Python3, TinyDB

Entrada más reciente Inicio Entrada antigua