PYTHON

Lenguaje de programación

Ing. Pablo E. Argañaras

parganaras@unrn.edu.ar

COIL 2023

1

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

2

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

8 Registros y archivos

8.1 Definición de registros

Un registro es un dispositivo para contener datos que pueden tener diferente tipo. Los registros normalmente están asociados a dispositivos externos de almacenamiento como discos. Algunos lenguajes de programación tienen un tipo especial de datos para representar registros.

Python no dispone en su librería estándar este tipo de datos, sin embargo, se puede usar una variable de tipo **lista** como un recipiente natural para almacenar un registro puesto que las listas en Python pueden contener componentes de diferente tipo y además pueden modificarse.

Si se requiere manejar varios registros, cada uno de ellos almacenado en una lista, se puede definir una lista de listas. Los componentes se pueden manejar mediante índices.

3

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

Ejemplo. Defina registros para almacenar la descripción de los artículos de una empresa. Agregar cada uno a una lista.

Cada artículo contendrá la siguiente descripción:
- Código del artículo
- Cantidad
- Precio
- Nombre.

Variables:
- `reg:` **Lista.** Es el contenedor de un registro
- `datos:` **Lista de listas**. Es el contenedor de los registros en memoria

```
>>> datos =[]                                  Inicia el contenedor de registros (lista)

>>> reg=[123,20,12.5,'libro']                  Descripción del primer artículo
>>> datos=datos+[reg]                          Agregar registro a la lista

>>> reg=[234,30,2.5,'cuaderno']                Descripción del segundo artículo
>>> datos=datos+[reg]                          Agregar registro a la lista

>>> datos
[[123, 20, 12.5, 'libro'], [234, 30, 2.5, 'cuaderno']]
>>> datos[1]
[234, 30, 2.5, 'cuaderno']
>>> datos[1][2]
2.5
>>> datos[1][3][0:4]
'cuad'
>>> datos[1][3][0:]
'cuaderno'
```

4

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

8.3 Funciones y métodos para manejo de archivos secuenciales en disco

El lenguaje Python provee las instrucciones para manejo de archivos en el disco.
En la forma básica, la información que se puede almacenar en disco son cadenas de texto.

En las siguientes instrucciones se usará la notación:

- **f:** Es el nombre de una variable u objeto creado de tipo archivo
- **n:** Es una cadena con el nombre del archivo en el disco. Es adecuado agregar la extensión **.txt** para reconocerlo como archivo de tipo texto con algún programa desde fuera de Python

Apertura de un archivo

Esta función se usa para crear o abrir el archivo para su uso

```
f=open(n,t)
```

t: es el tipo de operación que se realizará con el archivo:

Tipos de operación

- **'w'** Crear el archivo y agregar datos. Si el archivo existe, lo borra y lo crea.
- **'a'** Agregar datos al archivo. Si el archivo no existe lo crea y luego agrega.
- **'r'** Leer datos del archivo
- **'r+'** Leer y escribir en el archivo

Escritura de texto en el archivo

```
f.write(s)
```

s es alguna variable que contiene la línea de texto
Escribe en el archivo una línea de texto, normalmente finalizada con el carácter de fin de línea **'\n'**

5

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

Lectura del contenido de un archivo

```
v=f.readline()
```

En donde **v** es alguna variable que recibe la linea de texto
Lee una linea de texto del archivo hasta encontrar el carácter de fín de linea: **'\n'**
Si no quedan líneas, retorna una línea vacía.

Cierre de un archivo

```
f.close()
```

Al finalizar la operación con un archivo, debe cerrarse. En el ingreso de datos, esta operación se necesita para completar el ingreso de los datos al archivo en el disco.

Detectar la posición actual del dispositivo de lectura del archivo

```
p=f.tell()
```

p contendrá un entero con la posición actual. La posición inicial en el archivo es **0**

Ubicar el dispositivo de lectura del archivo en una posición especificada

```
f.seek(d)
```

d es el desplazamiento contado a partir del inicio que es la posición **0**

6

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

8.4 Conversión de registros a líneas de texto para almacenar en archivos secuenciales en el disco

Para almacenar los registros en el archivo, los datos pueden convertirse a texto con la función **str** separándolos con comas y armar una línea de texto, agregando al final la marca de fin de línea **'\n'**. Esta línea conteniendo los datos de un registro, es almacenada en el archivo en disco .

Almacenar un registro en una línea de texto en el archivo 'nuevo'

```
>>> ========= RESTART =========
>>> f=open('nuevo.txt','a')                    Abrir archivo para agregar
>>> reg=[123,20,12.5,'libro']
>>> cod=reg[0]
>>> cant=reg[1]
>>> pre=reg[2]
>>> nom=reg[3]
>>> linea=str(cod)+','+str(cant)+','+str(pre)+','+nom+'\n'    Armar la linea
>>> linea
'123,20,12.5,libro\n'
>>> f.write(linea)                             Escribir la linea en el archivo
>>> f.close()                                  Terminar de grabar y cerrar archivo
```

7

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

Al traer la línea de texto del disco se la convierte en una lista con la función **split**. Esta función reconoce las comas que separan los datos en la línea de texto. De esta lista se toman los componentes convirtiéndolos al tipo original.

Leer la línea de texto (registro) del disco y reconstruir los datos con el tipo original

```
>>> ========= RESTART =========
>>> f=open('nuevo.txt','r')                    Abrir archivo para lectura
>>> linea=f.readline()
>>> s=linea.split(',')                         Separar componentes en una lista
>>> s
['123', '20', '12.5', 'libro\n']
>>> cod=int(s[0])                              Convertir componentes a su tipo original
>>> cant=int(s[1])
>>> pre=float(s[2])
>>> nom=s[3]
>>> reg=[cod,cant,pre,nom]
>>> reg
[123, 20, 12.5, 'libro\n']
>>> f.close()                                  Cerrar archivo
```

El carácter **'\n'** es la marca de fin de linea para la lectura del **disco**. También produce un salto de línea al imprimir la línea en la pantalla. Si no se la desea, se puede eliminar del registro en **memoria**.

8

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

El módulo sqlite3 permite trabajar con bases de datos de tipo **SQLite**[1].

8.1.1 ¿Qué es SQLite?

SQLite es un sistema gestor de bases de datos relacional contenido en una pequeña librería escrita en C (~275kB).

A continuación se muestran algunas de sus **principales características**:

- Tablas, índices, «triggers» y vistas ilimitadas.
- Hasta 32K columnas en una tabla y filas ilimitadas.
- Índices multi-columna.
- Restricciones de tipo `CHECK`, `UNIQUE`, `NOT NULL` y `FOREIGN KEY`.
- Transacciones planas usando `BEGIN`, `COMMIT` y `ROLLBACK`
- Transacciones anidadas usando `SAVEPOINT`, `RELEASE` y `ROLLBACK TO`.
- Subconsultas.
- «Joins» de hasta 64 relaciones.

9

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

- «Joins» de tipo «left», «right» y «full outer».
- Uso de `DISTINCT`, `ORDER BY`, `GROUP BY`, `HAVING`, `LIMIT` y `OFFSET`.
- Uso de `UNION`, `UNION ALL`, `INTERSECT` y `EXCEPT`.
- Una amplia librería de funciones SQL estándar.
- Funciones de agregación.
- Funciones de ventana.
- Por supuesto el uso de `UPDATE`, `DELETE` e `INSERT`.
- Cláusula UPSERT.
- Soporte para valores JSON.

Y muchas otras que se pueden consultar en la página del proyecto.

10

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

8.1.2 Conexión a la base de datos

Una base de datos SQLite no es más que un fichero binario, habitualmente con extensión `.db` o `.sqlite`. Antes de realizar cualquier operación es necesario «conectar» con este fichero.

La **conexión a la base de datos** se realiza a través de la función connect() que espera recibir la ruta al fichero de base de datos y devuelve un objeto de tipo Connection:

```
>>> import sqlite3

>>> con = sqlite3.connect('python.db')

>>> con
<sqlite3.Connection at 0x106ea8210>
```

Advertencia: El módulo se llama **sqlite3** (no olvidarse del 3 al final).

La función `connect()` creará el fichero `python.db` (si es que no existe ya). En un principio no tiene contenido alguno:

```
>>> !file python.db
python.db: empty
```

11

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

Una vez que disponemos de la conexión ya podemos obtener un Cursor mediante la función cursor(). Un **cursor** se podría ver como un manejador para realizar operaciones sobre la base de datos:

```
>>> cur = con.cursor()

>>> cur
<sqlite3.Cursor at 0x106a63960>
```

12

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

8.1.3 Creación de tablas

Para poder crear una tabla primero debemos manejar los tipos de datos SQLite disponibles. Aunque hay alguno más, con los siguientes nos será suficiente para la inmensa mayoría de diseños de bases de datos que podamos necesitar:

- `INTEGER` para valores enteros.
- `REAL` para valores flotantes.
- `TEXT` para cadenas de texto.

Prudencia: Aunque `INT` también está permitido, se desaconseja su uso en favor de `INTEGER` especialmente cuando trabajamos con la librería Python `sqlite3` y no queremos obtener resultados inesperados.

13

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

Empecemos creando la tabla `pyversions` a través de un código SQL similar al siguiente:

```
CREATE TABLE pyversions (
    branch TEXT PRIMARY KEY,
    released_at_year INTEGER,
    released_at_month INTEGER,
    release_manager TEXT
)
```

Haremos uso del cursor creado para **ejecutar** estas instrucciones:

```
>>> sql = '''CREATE TABLE pyversions (
...     branch TEXT PRIMARY KEY,
...     released_at_year INTEGER,
...     released_at_month INTEGER,
...     release_manager TEXT
... )'''

>>> cur.execute(sql)
<sqlite3.Cursor at 0x106a63960>
```

Consejo: Las *cadenas multilínea* son grandes aliadas a la hora de escribir sentencias SQL.

Ya hemos creado la tabla `pyversions` de manera satisfactoria.

Si comprobamos ahora el contenido del fichero `python.db` podemos observar que nos indica la versión de SQLite y la última escritura:

```
>>> !file python.db
python.db: SQLite 3.x database, last written using SQLite version 3032003
```

14

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

8.1.4 Añadiendo datos

Para tener contenido sobre el que trabajar, vamos primeramente a añadir ciertos datos a la tabla. Como básicamente seguimos ejecutando sentencias SQL (en este caso de inserción) podemos volver a hacer uso de la función execute():

```
>>> sql = 'INSERT INTO pyversions VALUES ("2.6", 2008, 10, "Barry Warsaw")'

>>> cur.execute(sql)
<sqlite3.Cursor at 0x106a63960>
```

Aparentemente todo ha ido bien. Vamos a usar – temporalmente – la herramienta cliente `sqlite3`[2] para ver el contenido de la tabla:

```
$ sqlite3 python.db "select * from pyversions"
$ # Vacío
```


PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

Resulta que no obtenemos ningún registro. ¿Por qué ocurre esto? Se debe a que la transacción está aún pendiente de confirmar. Para consolidarla tendremos que hacer uso de la función commit():

```
>>> con.commit()
```

Ver también:

Cada vez que usamos la función `execute()` comienza una **nueva transacción** a la base de datos que debe confirmarse con `commit()` o bien deshacerse con rollback().

Ahora podemos comprobar que sí se han guardado los datos correctamente:

```
$ sqlite3 python.db "select * from pyversions"
2.6|2008|10|Barry Warsaw
```

[2] Herramienta cliente de sqlite para terminal.

Nota: La función `commit()` pertenece al objeto conexión, no al objeto cursor.

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

Cerrando la conexión

Al igual que ocurre con un fichero de texto, es necesario **cerrar la conexión abierta** para que se liberen los recursos asociados y se debloquee la base de datos.

La forma más directa de hacer esto sería:

```
>>> con.close()
```

Atención: Si hay alguna transacción pendiente, esta no será guardada al cerrar la conexión con la base de datos, si previamente no se consolidan los cambios.

17

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

Gestor de contexto

En SQLite también es posible utilizar un *gestor de contexto* sobre la conexión, que funciona de la siguiente manera:

- Si todo ha ido bien ejecutará un «commit» al final del bloque.
- Si ha habido alguna excepción ejecutará un «rollback» para que todo quede como al principio y deshacer los posibles cambios efectuados.

Ejemplo en el que todo va bien:

```
>>> try:
...     with con:
...         cur.execute('INSERT INTO pyversions VALUES ("3.13", 2024, 10, "Thomas␣
↪Wouters")')
... except sqlite3.IntegrityError:
...     print('Error: Duplicated Python version')
...

>>> con.close()
```

18

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

Ejemplo donde se produce un error:

```
>>> try:
...     with con:
...         cur.execute('INSERT INTO pyversions VALUES ("3.12", 2023, 10, "Thomas
↪Wouters")')
... except sqlite3.IntegrityError:
...     print('Error: Duplicated Python version')
...
Error: Duplicated Python version

>>> con.close()
```

Nótese que en ambos casos **debemos cerrar la conexión**. Esto no se realiza de forma automática.

Es interesante conocer las distintas excepciones que pueden producirse al trabajar con este módulo a la hora del control de errores y de plantear posibles escenarios de mejora.

19

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

8.1.5 Consultas

La manera más sencilla de hacer una consulta es **utilizar un cursor**. Existen dos aproximaciones en el tratamiento de los resultados de la consulta:

1. Registros como tuplas.
2. Registros como filas.

20

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

Registros como tuplas

Cuando ejecutamos una consulta **el resultado es un objeto iterable** que podemos recorrer de la misma manera que hemos hecho hasta ahora. Los datos nos vienen en forma de **tuplas**:

```
>>> for row in cur.execute('SELECT * FROM pyversions'):
...     print(row)
...
('2.6', 2008, 10, 'Barry Warsaw')
('2.7', 2010, 7, 'Benjamin Peterson')
('3.0', 2008, 12, 'Barry Warsaw')
('3.1', 2009, 6, 'Benjamin Peterson')
('3.2', 2011, 2, 'Georg Brandl')
('3.3', 2012, 9, 'Georg Brandl')
('3.4', 2014, 3, 'Larry Hastings')
('3.5', 2015, 9, 'Larry Hastings')
('3.6', 2016, 12, 'Ned Deily')
('3.7', 2018, 6, 'Ned Deily')
('3.8', 2019, 10, 'Łukasz Langa')
('3.9', 2020, 10, 'Łukasz Langa')
('3.10', 2021, 10, 'Pablo Galindo Salgado')
('3.11', 2022, 10, 'Pablo Galindo Salgado')
('3.12', 2023, 10, 'Thomas Wouters')
('3.13', 2024, 10, 'Thomas Wouters')
```

21

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

También tenemos la opción de utilizar las funciones fetchone() y fetchall() para obtener una o todas las filas de la consulta:

```
>>> res = cur.execute('SELECT * FROM pyversions')
```

```
>>> res.fetchone()
('2.6', 2008, 10, 'Barry Warsaw')

>>> res.fetchall()
[('2.7', 2010, 7, 'Benjamin Peterson'),
 ('3.0', 2008, 12, 'Barry Warsaw'),
 ('3.1', 2009, 6, 'Benjamin Peterson'),
 ('3.2', 2011, 2, 'Georg Brandl'),
 ('3.3', 2012, 9, 'Georg Brandl'),
 ('3.4', 2014, 3, 'Larry Hastings'),
 ('3.5', 2015, 9, 'Larry Hastings'),
 ('3.6', 2016, 12, 'Ned Deily'),
 ('3.7', 2018, 6, 'Ned Deily'),
 ('3.8', 2019, 10, 'Łukasz Langa'),
 ('3.9', 2020, 10, 'Łukasz Langa'),
 ('3.10', 2021, 10, 'Pablo Galindo Salgado'),
 ('3.11', 2022, 10, 'Pablo Galindo Salgado'),
 ('3.12', 2023, 10, 'Thomas Wouters'),
 ('3.13', 2024, 10, 'Thomas Wouters')]
```

22

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

Registros como filas

Este módulo también nos permite obtener los resultados de una consulta como objetos de tipo Row lo que facilita acceder a los valores de cada registro **tanto por el índice como por el nombre de la columna**.

Para «activar» este modo tendremos que fijar el valor de la factoría de filas en la conexión:

```
>>> con = sqlite3.connect('python.db')
>>> con.row_factory = sqlite3.Row
```

Importante: Para que las consultas usen esta factoría hay que fijar el atributo `row_factory` **antes** de crear el cursor correspondiente.

Ahora creamos un cursor, ejecutamos la consulta y accedemos a la primera fila del resultado **como si fuera un diccionario**:

23

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

```
>>> cur = con.cursor()
>>> res = cur.execute('SELECT * FROM pyversions')

>>> row = res.fetchone()
>>> row
<sqlite3.Row at 0x107b76190>

>>> row.keys()
['branch', 'released_at_year', 'released_at_month', 'release_manager']

>>> row['branch']
'2.6'
>>> row['released_at_year']
2008
>>> row['released_at_month']
10
>>> row['release_manager']
'Barry Warsaw'
```

Pero también es posible seguir accediendo a la cada columna **a través del índice**:

```
>>> row[0]
'2.6'
>>> row[1]
2008
>>> row[2]
10
>>> row[3]
'Barry Warsaw'
```

24

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

Desempaquetando filas

Cuando disponemos de una fila como resultado de una consulta (ya sea en formato tupla o en formato `sqlite3.Row`) podemos realizar un desempaquetado para separar sus campos en variables únicas:

```
>>> sql = 'SELECT * FROM pyversions'
>>> result = cur.execute(sql)
>>> row = result.fetchone()

>>> row
<sqlite3.Row at 0x102e71ab0>

>>> branch, released_at_year, released_at_month, release_manager = row
```

```
>>> branch
'2.6'
>>> released_at_year
2008
>>> released_at_month
10
>>> release_manager
'Barry Warsaw'
```

25

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

Número de filas

Hay ocasiones en las que lo que necesitamos obtener no es el dato en sí mismo, sino el **número de filas vinculadas a una determinada consulta**. En este sentido hay varias alternativas.

La primera aproximación es **utilizar herramientas Python** para obtener la longitud del resultado de la consulta:

```
>>> result = cur.execute('SELECT * FROM pyversions')

>>> rows = result.fetchall()

>>> len(rows)
15
```

La segunda aproximación es **mediante la sentencia SQL para contar**: `COUNT()` y obtener su resultado:

```
>>> result = cur.execute('SELECT COUNT(*) FROM pyversions')

>>> rows = result.fetchone()

>>> rows[0]  # sólo hay una columna
15
```

Obviamente si lo único que necesitamos es obtener el número de filas afectadas, esta segunda opción a través de `COUNT()` tiene más sentido.

26

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

Comprobando si hay resultados

Hay ocasiones en las que necesitamos comprobar si la consulta tiene algún registro.

Una manera de enfocar este escenario es utilizando el *operador morsa* teniendo en cuenta que `fetchone()` devuelve `None` si la consulta es vacía. Veamos su implementación:

```
>>> con = sqlite3.connect(db_path)
>>> cur = con.cursor()

>>> # Consulta vacía
>>> res = cur.execute('SELECT * FROM pyversions WHERE branch=4.0')

>>> if row := res.fetchone():
...     print(row)
... else:
...     print('Empty query')
...
Empty query

>>> # Consulta con datos
>>> res = cur.execute('SELECT * FROM pyversions WHERE branch=3.0')

>>> if row := res.fetchone():
...     print(row)
... else:
...     print('Empty query')
...
('3.0', 2008, 12, 'Barry Warsaw')
```

27

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

Información de filas

Cuando ejecutamos una sentencia de modificación sobre la base de datos podemos obtener el **número de filas modificadas**.

Este dato lo sacamos del atributo `rowcount` del cursor. Veamos un ejemplo:

```
>>> con = sqlite3.connect('python.db')
>>> cur = con.cursor()

>>> cur.execute('SELECT * FROM pyversions').fetchall()
[('2.6', 2008, 10, 'Barry Warsaw'),
 ('2.7', 2010, 7, 'Benjamin Peterson'),
 ('3.0', 2008, 12, 'Barry Warsaw'),
 ('3.1', 2009, 6, 'Benjamin Peterson'),
 ('3.2', 2011, 2, 'Georg Brandl'),
 ('3.3', 2012, 9, 'Georg Brandl'),
 ('3.4', 2014, 3, 'Larry Hastings'),
 ('3.5', 2015, 9, 'Larry Hastings'),
 ('3.6', 2016, 12, 'Ned Deily'),
 ('3.7', 2018, 6, 'Ned Deily'),
```

28

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

```
 ('3.8', 2019, 10, 'Łukasz Langa'),
 ('3.9', 2020, 10, 'Łukasz Langa'),
 ('3.10', 2021, 10, 'Pablo Galindo Salgado'),
 ('3.11', 2022, 10, 'Pablo Galindo Salgado'),
 ('3.12', 2023, 10, 'Thomas Wouters'),
 ('3.13', 2024, 10, 'Thomas Wouters')]

>>> cur.execute('UPDATE pyversions SET released_at_year=2000')
<sqlite3.Cursor at 0x105593dc0>

>>> cur.rowcount
16  # filas modificadas
```

Igualmente cuando insertamos un registro en la base de datos podemos obtener cuál es el **identificador de la últila fila insertada**:

```
>>> cur.execute('INSERT INTO pyversions VALUES ("3.14", 2025, 10, "Guido Van Rossum")
↪')
<sqlite3.Cursor at 0x105593dc0>

>>> cur.lastrowid
17
```

29

PYTHON

Lenguaje de programación

Acceso a datos

COIL 2023

¿Preguntas?

30