MINDFUL LEARNING

python

A language is not worth knowing unless it teaches you to think differently

- Larry Wall, Randal Schwartz

Professional

python

Database and Persistance

MINDFUL LEARNING

python

Databases and SQL

- Database is a structured set of data held in a computer
- **Structured Query Language** (**SQL**) is a standard computer **language** for relational database management and data manipulation
- **SQL** is used to **query**, insert, update and modify data

DBM Files

- DBM files are a standard tool in the Python library for database management
- DBM files are very much like dictionaries
  - Indexing by key fetches data from the file.
  - Assigning to an index stores data in the file.

MINDFUL LEARNING

python

Working with DBM

```
>>> import dbm # get interface: bsddb, gnu, ndbm, dumb
>>> file = dbm.open('movie', 'c') # make a DBM file called 'movie'
>>> file['Batman'] = 'Pow!' # store a string under key 'Batman'
>>> file.keys() # get the file's key directory
[b'Batman']
>>> file['Batman'] # fetch value for key 'Batman'
b'Pow!'
>>> who = ['Robin', 'Cat-woman', 'Joker']
>>> what = ['Bang!', 'Splat!', 'Wham!']
>>> for i in range(len(who)):
... file[who[i]] = what[i] # add 3 more "records"
...
>>> file.keys()
[b'Cat-woman', b'Batman', b'Joker', b'Robin']
>>> len(file), 'Robin' in file, file['Joker']
>>> for key in file.keys(): print(key, file[key])
...
b'Cat-woman' b'Splat!'
b'Batman' b'Pow!'
b'Joker' b'Wham!'
b'Robin' b'Bang!'
```

anydbm

- **anydbm** is a front-end for DBM-style databases that use simple string values as keys to access records containing strings.
- It uses the whichdb module to identify **dbhash**, **gdbm**, and **dbm** databases, then opens them with the appropriate module.
- It is used as a backend for **shelve**, which knows how to store objects using **pickle**
- The **anydbm** module offers an alternative to choose the best DBM module available.

MINDFUL LEARNING

python

Working with anydbm

```
>>> import anydbm
>>> # open a DB. The c option opens in read/write mode and
   creates the file if needed.
>>> db = anydbm.open('websites', 'c')
>>> # add an item
>>> db["item1"] = "First example"
>>> print db['item1']
"First example"
>>> # close and save
>>> db.close()
```

Mindful Learning

python

Pickle

- Text files are convenient because you can read and manipulate them with any text editor, but they're limited to storing a series of characters.
- Sometimes you may want to store more complex information, like a list or a dictionary, for example
- Pickling means to preserve—and that's just what it means in Python
  - You can pickle a complex piece of data, like a list or dictionary, and save it in its entirety to a file
- The **pickle** module allows you to pickle and store more complex data in a file.

What Can Be Pickled and Unpickled?

- The following types can be pickled:
  - None, True, and False
  - integers, long integers, floating point numbers, complex numbers
  - normal and Unicode strings
  - tuples, lists, sets, and dictionaries containing only picklable objects
  - functions defined at the top level of a module
  - built-in functions defined at the top level of a module
  - classes that are defined at the top level of a module
  - instances of such classes whose **__dict__** or the result of calling **__getstate__()** is picklable

Mindful Learning

python

Shelve

- The **shelve** module allows you to store and randomly access pickled objects in a file

MINDFUL LEARNING

python

Pickling and Shelving Functions

Function	Purpose
pickle.dump(obj, file_handle)	Write a pickled representation of *obj* to the open file object *file*
pickle.load(file)	Read a string from the open file object *file* and interpret it as a pickle data stream, reconstructing and returning the original object hierarchy
shelve.open(*filename*, *flag='c'*, *protocol=None*, *writeback=False*)	Open a persistent dictionary. The filename specified is the base filename for the underlying database
Shelf.sync()	Write back all entries in the cache if the shelf was opened with *writeback* set to True
Shelf.close()	Synchronize and close the persistent *dict* object.

Refer:
https://docs.python.org/2/library/shelve.html
https://docs.python.org/2/library/pickle.html

MINDFUL LEARNING

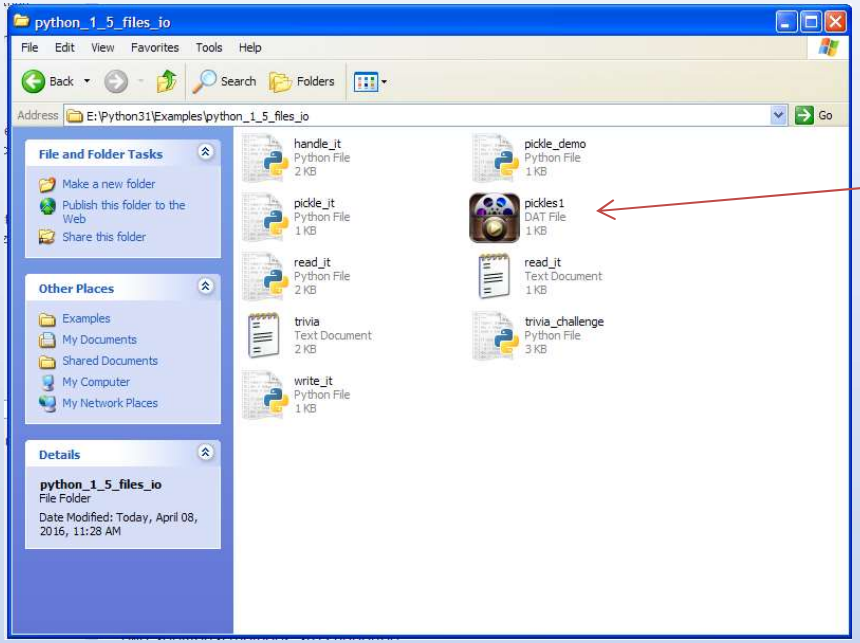
python

Pickle: Example

```
# pickle_demo.py: Demonstrates pickling data

import pickle

print("Pickling lists")

variety = ["sweet", "hot", "dill"]
shape   = ["whole", "spear", "chip"]
brand   = ["Claussen", "Heinz", "Vlassic"]
f       = open("pickles1.dat", "wb")

pickle.dump(variety, f)
pickle.dump(shape, f)
pickle.dump(brand, f)

f.close()

print("\nUnpickling lists")
f = open("pickles1.dat", "rb")

variety = pickle.load(f)
shape   = pickle.load(f)
brand   = pickle.load(f)
print(variety)
print(shape)
print(brand)

f.close()
```

Output

```
>>>
Pickling lists

Unpickling lists
['sweet', 'hot', 'dill']
['whole', 'spear', 'chip']
['Claussen', 'Heinz', 'Vlassic']
```



pickles1.dat

MINDFUL LEARNING

python

Shelve: Example

```
# shelve_demo.py: Demonstrates shelving data

import shelve

print("\nShelving lists")
s = shelve.open("pickles2.dat")
s["variety"] = ["sweet", "hot", "dill"]
s["shape"] = ["whole", "spear", "chip"]
s["brand"] = ["Claussen", "Heinz", "Vlassic"]
s.sync()    # make sure data is written

print("\nRetrieving lists from a shelved file:")
print("brand -", s["brand"])
print("shape -", s["shape"])
print("variety -", s["variety"])
s.close()

input("\n\nPress the enter key to exit.")
```

MINDFUL LEARNING

python

Output

```
>>>

Shelving lists

Retrieving lists from a shelved file:
brand - ['Claussen', 'Heinz', 'Vlassic']
shape - ['whole', 'spear', 'chip']
variety - ['sweet', 'hot', 'dill']


Press the enter key to exit.
```

Relational Databases

- A **relational database** is a collection of data items organized as a set of formally-described tables from which data can be accessed or reassembled in many different ways without having to reorganize the **database** tables
- Some examples are:
  - MySQL
  - PostgreSQL
  - Oracle

Python DB - API

- The API is a specification that states a set of required objects and database access mechanisms to provide consistent access across the various database adapters and underlying database systems
- The DB API provides a minimal standard for working with databases using Python structures and syntax wherever possible. This API includes the following:
  - Importing the API module.
  - Acquiring a connection with the database.
  - Issuing SQL statements and stored procedures.
  - Closing the connection

python

Connecting to MySQL

- To access the MySQL database from Python, you need a database driver.
- MySQL connector/Python is a standardized database driver provided by MySQL.

```
>>> import mysql.connector
>>> mysql.connector.connect(host='localhost',database='
  mysql',user='root',password='')
```

python_mysql_connect1.py

Mindful Learning

python

MySQLConnection Object

- We can create a database configuration file named **config.ini** and define a section with four parameters as follows:

```
[mysql]
host = localhost
database = python_mysql
user = root
password =
```

- We use **ConfigureParser** package to read the configuration file.

MINDFUL LEARNING

python

MySQLConnection Object

```
from configparser import ConfigParser


def read_db_config(filename='config.ini', section='mysql'):
    """ Read database configuration file and return a dictionary object
    :param filename: name of the configuration file
    :param section: section of database configuration
    :return: a dictionary of database parameters
    """
    # create parser and read ini configuration file
    parser = ConfigParser()
    parser.read(filename)

    # get section, default to mysql
    db = {}
    if parser.has_section(section):
        items = parser.items(section)
        for item in items:
            db[item[0]] = item[1]
    else:
        raise Exception('{0} not found in the {1} file'.format(section, filename))

    return db
```

python_mysql_dbconfig.py

```
>>> from python_mysql_dbconfig import read_db_config
>>> read_db_config()
{'password': '', 'host': 'localhost', 'user': 'root', 'database': 'python_mysql'}
```

Mindful Learning

python

MySQLConnection Object

```
from mysql.connector import MySQLConnection, Error
from python_mysql_dbconfig import read_db_config


def connect():
    """ Connect to MySQL database """

    db_config = read_db_config()

    try:
        print('Connecting to MySQL database...')
        conn = MySQLConnection(**db_config)

        if conn.is_connected():
            print('connection established.')
        else:
            print('connection failed.')

    except Error as error:
        print(error)

    finally:
        conn.close()
        print('Connection closed.')


if __name__ == '__main__':
    connect()
```

ython_mysql_connect2.py

python

Python MySQL Query

- To query data in a MySQL database from Python, you need to do the following steps:
  - Connect to the MySQL Database - you get a MySQLConnection object.
  - Instantiate a MySQLCursor object from the the MySQLConnection object.
  - Use the cursor to execute a query by calling its execute() method.
  - Use fetchone() , fetchmany() or fetchall() method to fetch data from the result set.
  - Close the cursor as well as the database connection by calling the close() method of the corresponding object.

fetchone()

MINDFUL LEARNING

python

```
dbconfig = read_db_config()
conn = MySQLConnection(**dbconfig)
cursor = conn.cursor()
cursor.execute("SELECT * FROM books")

row = cursor.fetchone()

while row is not None:
    print(row)
    row = cursor.fetchone()
```

fetchall()

MINDFUL LEARNING

python

```
dbconfig = read_db_config()
conn = MySQLConnection(**dbconfig)
cursor = conn.cursor()
cursor.execute("SELECT * FROM books")
rows = cursor.fetchall()

print('Total Row(s):', cursor.rowcount)
for row in rows:
    print(row)
```

fetchmany()

MINDFUL LEARNING

python

- First, we develop a generator that chunks the database calls into a series of fetchmany() calls as follows:

```
def iter_row(cursor, size=10):
    while True:
        rows = cursor.fetchmany(size)
        if not rows:
            break
        for row in rows:
            yield row
```

MINDFUL LEARNING

python

fetchmany()

```
def query_with_fetchmany():
    try:
        dbconfig = read_db_config()
        conn = MySQLConnection(**dbconfig)
        cursor = conn.cursor()

        cursor.execute("SELECT * FROM books")

        for row in iter_row(cursor, 10):
            print(row)

    except Error as e:
        print(e)

    finally:
        cursor.close()
        conn.close()
```

Inserting Data

- To insert new rows into a MySQL table, you follow the steps below:
  - Connect to the MySQL database server by creating a new MySQLConnection object.
  - Initiate a MySQLCursor object from the MySQLConnection object.
  - Execute the INSERT statement to insert data into the intended table.
  - Close the database connection.

MINDFUL LEARNING

python

Inserting One Row

- Create a query

```
def insert_books(books):
    query = "INSERT INTO books(title,isbn) " \
            "VALUES(%s,%s)"
    args = (title, isbn)
```

- Define the function

```
db_config = read_db_config()
        conn = MySQLConnection(**db_config)

        cursor = conn.cursor()
        cursor.execute(query, args)
```

- Execute

```
insert_book('A Sudden Light','9781439187036')
```

MINDFUL LEARNING

python

Inserting Multiple Rows

- Create query

```
def insert_books(books):
    query = "INSERT INTO books(title,isbn) " \
            "VALUES(%s,%s)"
```

- Define function

```
db_config = read_db_config()
        conn = MySQLConnection(**db_config)
        cursor = conn.cursor()
        cursor.executemany(query, books)
        conn.commit()
```

- Execute

```
books = [
('Harry Potter And The Order Of The Phoenix', '9780439358071'),
('Gone with the Wind', '9780446675536'),
('Pride and Prejudice '9780679783268')]

insert_books(books)
```

Updating Rows

- To update data in a MySQL table in Python, you follow the steps below:
  - Connect to the database by creating a new MySQLConnection object.
  - Create a new MySQLCursor object from the MySQLConnection object and call the execute()method of the MySQLCursor object.
  - To accept the changes, you call the commit() method of the MySQLConnection object after calling the execute() method. Otherwise, no changes will be made to the database.
  - Close the cursor and database connection

Updating Rows

- Create the query

```
def update_book(book_id, title):
    query = """ UPDATE books
                SET title = %s
                WHERE id = %s """

    data = (title, book_id)
```

- Define function

```
conn = MySQLConnection(**db_config)
        cursor = conn.cursor()
        cursor.execute(query, data)
        conn.commit()
```

- Execute

```
update_book(37, 'The Giant on the Hill *** TEST ***')
```

Deleting Rows

- To delete rows in a MySQL table from Python, you need to do the following steps:
  - Connect to the database by creating a new MySQLConnection object.
  - Instantiate a new cursor object and call its execute() method.
  - To commit the changes, you should always call the commit() method of the MySQLConnection object after calling the execute() method.
  - Close the cursor and database connection by calling close() method of the corresponding objects.

Deleting

MINDFUL LEARNING

python

- Create the query

```
def delete_book(book_id):
    db_config = read_db_config()

    query = "DELETE FROM books WHERE id = %s"
```

- Define function

```
conn = MySQLConnection(**db_config)
cursor = conn.cursor()
cursor.execute(query, (book_id,))
conn.commit()
```

- Execute

```
delete_book(102)
```

Updating BLOB

MINDFUL LEARNING

python

- Create function to read a file

```
def read_file(filename):
    with open(filename, 'rb') as f:
        photo = f.read()
    return photo
```

- Use UPDATE query

```
def update_blob(author_id, filename):
    # read file
    data = read_file(filename)

    # prepare update query and data
    query = "UPDATE authors " \
            "SET photo = %s " \
            "WHERE id  = %s"

    args = (data, author_id)
```

Reading BLOB

MINDFUL LEARNING

python

- Create a function to write file

```
def write_file(data, filename):
    with open(filename, 'wb') as f:
        f.write(data)
```

- Use SELECT query

```
def read_blob(author_id, filename):
    query = "SELECT photo FROM authors WHERE id = %s"

        …
        photo = cursor.fetchone()[0]

        # write blob data into a file
        write_file(photo, filename)
```

Calling MySQL Procedures

- To call a stored procedure in Python, you follow the steps below:
  - Connect to MySQL database by creating a new MySQLConnection object
  - Instantiate a new MySQLCursor object from the MySQLConnection object by calling the cursor() method
  - Call callproc() method of the MySQLCursor object.
    - You pass the stored procedure's name as the first argument of the callproc() method.
    - If the stored procedure requires parameters, you need to pass a list as the second argument to the callproc() method.
    - In case the stored procedure returns a result set, you can invoke the stored_results() method of the MySQLCursor object to get a list iterator and iterate this result set by using the fetchall() method.
  - Close the cursor and database connection as always

Example

- The first stored procedure gets all books with author's information from books and authors tables:

```
DELIMITER $$

USE python_mysql$$

CREATE PROCEDURE find_all()
BEGIN
SELECT title, isbn, CONCAT(first_name,' ',last_name) AS author
FROM books
INNER JOIN book_author ON book_author.book_id =  books.id
INNER JOIN AUTHORS ON book_author.author_id = authors.id;
END$$

DELIMITER ;
```

- To call the procedure you can say:

```
CALL find_all();
```

Example

- The second stored procedure named find_by_isbn() that is used to find a book by its ISBN as follows:

```
DELIMITER $$

CREATE PROCEDURE find_by_isbn(IN p_isbn VARCHAR(13),OUT
  p_title VARCHAR(255))
    BEGIN
            SELECT title INTO p_title FROM books
            WHERE isbn = p_isbn;
    END$$

DELIMITER ;
```

- To call the procedure you can say:

```
CALL find_by_isbn('1235927658929',@title);
SELECT @title;
```

MINDFUL LEARNING

python

Example

- Calling the first procedure from Python script

```
cursor.callproc('find_all')
  # print out the result
  for result in cursor.stored_results():
      print(result.fetchall())
```

- Calling the second procedure from Python script

```
args = ['1236400967773', 0]
result_args = cursor.callproc('find_by_isbn', args)
print(result_args[1])
```

MySQLdb

MINDFUL LEARNING

python

- **MySQLdb** is an interface for connecting to a MySQL database server from Python.
- It implements the Python Database API v2.0 and is built on top of the MySQL C API

```
import MySQLdb
db = MySQLdb.connect("localhost","testuser","test123","TESTDB")
cursor = db.cursor()
cursor.execute("SELECT VERSION()")
data = cursor.fetchone()

print "Database version : %s " % data
db.close()
```

Database on a Different Machine

- You can specify the host name which run the database server

```
db =
  mysql.connect(host="192.168.1.300",user="boss",
  passwd="boss",db="office")
```

- Alternatively, you can specify it in the config.ini file

python

Working with other Databases

- **pymongo** allows you to work on the Mongo DB dabase
- **psycopg2** allows you to work with PostgreSQL
- **cx_Oracle** allows you to work with Oracle Database