BUILDING APPLICATIONS WITH PYTHON AND WEB FRAMEWORKS

2

Prerequisites

PYTHON FOR EVERYBODY

You can read Chapters 2,3,4,8,9,10,15 from http://do1.dr-chuck.com/pythonlearn/EN_us/pythonlearn.pdf

We could also watch the chapters from this book in video lectures from https://www.py4e.com/lessons

The **Web Server Gateway Interface** (WSGI) is a specification for simple and universal interface between web servers and web applications or frameworks for the Python programming language.

HTML

You must be familiar with simple HTML and understand forms and HTTP methods (post and get)

MVC

You must understand the concepts of web frameworks and MVC patterns.

3

Python and Databases

4

Python and Database manipulation

- We will next focus on using Python to work with data in MySQL.
- Please refer to “Part-1-Tools-for-exercise-3” slides to review the software needed for building a web application with Python.

5

Python and Database manipulation

Before connecting to a MySQL database, make sure you have created a database TESTDB.

- We will create a table EMPLOYEE in TESTDB.
- This table has fields FIRST_NAME, LAST_NAME, AGE, SEX and INCOME.
- User ID and password are set to access TESTDB.
- Python module PyMySQL should be installed properly on your machine.
- You understand MySQL Basics ;-).

6

Python and Database manipulation

Example Python MySQL connector

```
import pymysql

# Open database connection
db = pymysql.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# execute SQL query using execute() method.
cursor.execute("SELECT VERSION()")

# Fetch a single row using fetchone() method.
data = cursor.fetchone()
print ("Database version : %s " % data)

# disconnect the Database connection
db.close()
```

7

Python and Database manipulation

- A **cursor** is like a file handle that we can use to perform operations on the data stored in the database. Calling ***cursor()*** is very similar conceptually to calling *open()* when dealing with text files.
  - If a connection is established with the datasource, then a Connection Object is returned and saved into **db** for further use, otherwise **db** is set to None.

- Once we have the cursor, we can begin to execute commands on the contents of the database using the ***execute()*** method.
  - Execute returns an iterator over the results.
  - Rows are returned as python tuples

8

Python and Database manipulation

Let’s create the database Table using Python

```
import pymysql

# Open database connection
db = pymysql.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Drop table if it already exist using execute() method.
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")

# Create table as per requirement
sql = """CREATE TABLE EMPLOYEE (
   FIRST_NAME  CHAR(20) NOT NULL,
   LAST_NAME  CHAR(20),
   AGE INT,
   SEX CHAR(1),
   INCOME FLOAT )"""

cursor.execute(sql)

# disconnect from server
db.close()
```

9

Python and Database manipulation

An INSERT operation

```
import pymysql

# Open database connection
db = pymysql.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to INSERT a record into the database.
sql = """INSERT INTO EMPLOYEE(FIRST_NAME,
   LAST_NAME, AGE, SEX, INCOME)
   VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""
try:
   # Execute the SQL command
   cursor.execute(sql)
   # Commit your changes in the database
   db.commit()
except:
   # Rollback in case there is any error
   db.rollback()

# disconnect from server
db.close()
```

READ Operation

READ Operation on any database means to fetch some useful information from the database.

- **fetchone()** – It fetches the next row of a query result set. A result set is an object that is returned when a cursor object is used to query a table.
- **fetchall()** – It fetches all the rows in a result set. If some rows have already been extracted from the result set, then it retrieves the remaining rows from the result set.
- **rowcount** – This is a read-only attribute and returns the number of rows that were affected by an execute() method.

11

Python and Database manipulation

```python
import pymysql

# Open database connection
db = pymysql.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to INSERT a record into the database.
sql = "SELECT * FROM EMPLOYEE WHERE INCOME > '%d'" % (1000)
try:
   # Execute the SQL command
   cursor.execute(sql)
   # Fetch all the rows in a list of lists.
   results = cursor.fetchall()
   for row in results:
      fname = row[0]
      lname = row[1]
      age = row[2]
      sex = row[3]
      income = row[4]
      # Now print fetched result
      print ("fname = %s,lname = %s,age = %d,sex = %s,income = %d" % \
             (fname, lname, age, sex, income ))
except:
   print ("Error: unable to fetch data")

# disconnect from server
db.close()
```

12

Python and Database manipulation

UPDATE Operation on any database means to update one or more records, which are already available in the database.

```
#!/usr/bin/python3

import pymysql

# Open database connection
db = pymysql.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to UPDATE required records
sql = "UPDATE EMPLOYEE SET AGE = AGE + 1
                          WHERE SEX = '%c'" % ('M')
try:
   # Execute the SQL command
   cursor.execute(sql)
   # Commit your changes in the database
   db.commit()
except:
   # Rollback in case there is any error
   db.rollback()

# disconnect from server
db.close()
```

Uses two methods to either commit or rollback a transaction.

Transactions are a mechanism that ensures data consistency. Transactions have the following four properties

Atomicity – Either a transaction completes or nothing happens at all.

Consistency – A transaction must start in a consistent state and leave the system in a consistent state.

Isolation – Intermediate results of a transaction are not visible outside the current transaction.

Durability – Once a transaction was committed, the effects are persistent, even after a system failure.

14

Bottle web framework

15

Installation

- Bottle is micro web-framework
- A Web framework is a set of **components** designed to simplify your web development process.
- It has basic **structuring** tools in it, which serve as a solid base for your project.
- It allows you to focus on the most important details and project's goals instead of creating things, that you can simply pull out of the framework.

16

Installation

- Bottle does not depend on any external libraries.
- Bottle is micro web-framework

- You can just download *bottle.py* into your project directory and start coding. (examples located at eclass)

17

Hello example

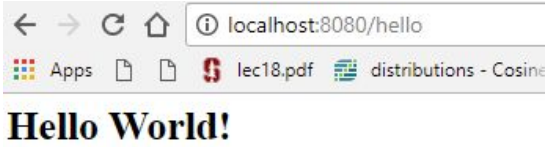
- Let's start with a very basic "Hello World" example:

```python
from bottle import route, run

@route('/hello')
def hello():
    return "Hello World!"

run(host='localhost', port=8080, debug=True)
```

- Write this script in a text editor of your choice, name it **mybottletutor.py** and run this script on command prompt

```
C:\Users\SasaPC\Documents\DB_Lab\MYTUTOR>python mybottletutor.py
Bottle v0.13-dev server starting up (using WSGIRefServer())...
Listening on http://localhost:8080/
Hit Ctrl-C to quit.
```

- Visit ***http://localhost:8080/hello*** from your web browser and you will see "Hello World!".



18

How it works

- The **route()** decorator binds a piece of code to an URL path. In this case, we link the /hello path to the hello() function.
- This is called a route and is the most important concept of this framework . ***Routing is the process of mapping a requested URL to the code responsible for generating the associated HTML.***
- You can define as many routes as you want.
- Whenever a browser requests an URL, the associated function is called and the return value is sent back to the browser.

```
from bottle import route, run

@route('/hello')
def hello():
    return "Hello World!"

run(host='localhost', port=8080, debug=True)
```

19

ROUTING STATIC FILES

Static files such as images, CSS or html files are not served automatically. You have to add a route and a callback to control which files get served and where to find them:

```
from bottle import static_file
@route('/static/<filename>')
def server_static(filename):
    return static_file(filename, root='/path/to/your/static/files')
```

The **static_file()** function is a helper to serve files in a safe and convenient way (see Static Files).

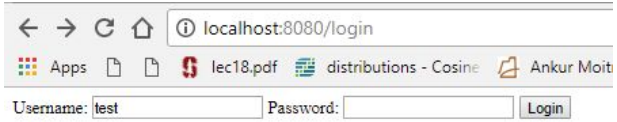
This example is limited to files directly within the /path/to/your/static/files directory because the <filename> wildcard won't match a path with a slash in it.

20

Forms and HTTP Requests

- Writing html code (in the following example we present a simple form and we display a simple message - **no form processing is performed in the followin example**)

```
from bottle import run, post, route

@route('/login')
def login():
    return '''
        <form action="/login" method="post">
            Username: <input name="username" type="text" />
            Password: <input name="password" type="password" />
            <input value="Login" type="submit" />
        </form>
    '''

@route('/login', method='POST')
def do_login():
    username = request.forms.get('username')
    password = request.forms.get('password')
    return "<p>your login information was correct! </p>"


run(host='localhost', port=8080, debug=True)
```



THANKS!

Any questions?

You can find me at

- akolovou@di.uoa.gr

22

CREDITS

References

- https://bottlepy.org/docs/dev/
- https://www.fullstackpython.com/bottle.html
- https://www.youtube.com/watch?v=g_9nsFJS_pk
- http://do1.dr-chuck.com/pythonlearn/EN_us/pythonlearn.pdf
- https://www.blog.pythonlibrary.org/2013/07/22/bottle-creating-a-python-todo-list-web-app/
-