Week 8 Tutorial Notes

Welcome to your first (or n-th) exposure to Python interacting with a database. If you're not familiar with Python, don't worry too much because we'll help you go through it, but if you are familiar with Python, make sure you help the people around you that aren't as confident with the code.

Installation Instructions (for your own computers)

How do you get this working on your computers?

1. Install python3 (windows, mac and linux): https://www.python.org/downloads/
2. Make sure you have *pip3* [python package installer] installed on your system
    - https://pip.pypa.io/en/stable/installing/
    - http://stackoverflow.com/questions/6587507/how-to-install-pip-with-python-3
    - http://stackoverflow.com/questions/4750806/how-do-i-install-pip-on-windows
    - If you're on ubuntu: `sudo apt-get install python3-pip`
3. After you have installed pip run: `sudo pip3 install -r requirements.txt`
    - Note: you are pointing to the 'requirements' text inside this folder, you can put the path in as well if needed.
    - PIP varies between distributions, if you have python3 and python2 on your computer, `pip3` will access the pip for python3
4. Try running: `python3 main.py`
5. *ta-daaa*

Errors

If you run into some errors, it may stop you from installing them, there are ways to fix it.

- Installing packages from pip errors:
    - [UBUNTU]: pg_config executable not found
        - Do the command: `sudo apt-get install libpq-dev python-dev`
    - Failed to install
        - Did you have `sudo pip3 ....` ? That fixes a lot of errors
- *[Don't be scared to google away errors :) If you can't let us know and we can help]*

Running the python code

On the uni servers

Windows

1. Download the tutorial ZIP and save it in your U: drive
    - Make sure to unzip it!

2. Open PuTTY
3. Connect to UCPU1 or UCPU2
    - ucpu1: ucpu1.ug.it.usyd.edu.au
    - ucpu2: ucpu2.ug.it.usyd.edu.au

4. Navigate to the folder
5. Edit the files needed:
    - config.ini : edit your details
    - main.py : find a unique port

6. now run: `python3 main.py`
7. Open a web browser
    - http://172.16.65.241:YOUR*PORT*NUMBER_HERE

OR

1. Download the zip and unzip it
2. Edit the files needed:
    - config.ini : edit your details
    - main.py : find a unique port

3. Open the ***'server.bat'*** file
4. Go to **FIREFOX**

    1. Disable the proxy (settings -> advanced -> network)
    2. Go to: http://localhost:YOUR*PORT*NUMBER_HERE

Linux (Fedora)

1. Download the tutorial ZIP and save it somewhere you can access
2. Open up the terminal
3. Navigate to the folder
4. Edit the files needed:
    - config.ini : edit your details

    - main.py : find a unique port

5. now run: `python3 main.py`
6. Open up **FIREFOX**

    1. Disable the proxy (settings -> advanced -> network)
    2. Go to: http://localhost:YOUR*PORT*NUMBER_HERE

On your own laptop

Windows

1. Download the zip and unzip it
2. Edit the files needed:
    - config.ini : edit your details
    - main.py : find a unique port

3. Open the ***'server.bat'*** file
4. Go to your web browser
    1. Go to: http://localhost:YOUR*PORT*NUMBER_HERE

MAC/LINUX

1. Download the tutorial ZIP and save it somewhere you can access
2. Open up the terminal
3. Navigate to the folder
4. Edit the files needed:
    - config.ini : edit your details
    - main.py : find a unique port

5. now run: `python3 main.py`
6. Open up your browser
    1. Disable the proxy (settings -> advanced -> network)
    2. Go to: http://localhost:YOUR*PORT*NUMBER_HERE

We are using 2 main modules today

Psycopg2

This is how Python talks to our **Postgres** database, so we can interact with it, give it queries and make sure

that we can get results to display.

So, how do we use it.

If you look in the `database.py` file, this is where the main interaction between the code and the database occurs.

Opening a connection

Opening a connection is done easier with functions. In the database python file, you can see the `database_connect()` function. This function opens up a database connection with our postgres.

NOTE: You will need to have your `config.ini` file updated with your details before starting

So when we call `database_connect()` it will return a connection for us that we can use.

Making the cursor and executing queries

Making the connection

```python
connection = database_connect()
```

Now we can make a cursor and execute queries.

```python
cursor = connection.cursor()
try:
    cursor.execute("SELECT * FROM students WHERE sid=%s", [sid])
    cursor.execute("""SELECT *
                        FROM students
                        WHERE sid=%s AND password=%s""", (sid, password))
except:
    # This happens if there is an error executing the query
    print("Error executing function")
```

Get the values and close the cursor

```python
results = cursor.fetchall()

cursor.close()              # IMPORTANT: close cursor
connection.close()          # IMPORTANT: close connection

if(results is None):
    # What happens when results is NULL?
    results = []

return results
```

Flask

Flask is a nice and simple web framework for Python to interact with backend and frontend.

If you look at `main.py` you won't see much, it will just be starting the app which is just for cleanliness, the `routes.py` is where we want to be looking for the main functionality.

Making a route

First off, what is a route? This is up in the url when you have the different paths: e.g. `http://example.com/routeOne` in the python is `@app.route('/routeOne')`

If you look in the routes python file, most of them are complete.

A good place to look is http://flask.pocoo.org/docs/0.10/tutorial/views/

Rendering a HTML template?

There are two aspects to HTML, the **template** files and the **static** files. The static files are where we keep the CSS (styling of the page [making it pretty]) and scripts (like JavaScript), things we don't have to worry about as much, we are more interested in the templates.

The two parts we check are the HTML and the routes.py

Python (routes.py)

If you check the routes.py, you are able to see that we have one line we want to concentrate on:

```python
return render_template('page_name.html', variable_in_html=variable_in_python)
```

The variable in python is just a normal variable, e.g. `name='Chris'`. This can then be used in the HTML to show up, so we can make it dynamic :) You will need to show which variable is equal to which.

HTML

Most of you might think that HTML is boring, but the important part is understanding what it's doing. With Flask, there is special notation so that you can interact with the python code.

What we want to look at are these notations:

(For help: http://flask.pocoo.org/docs/0.10/tutorial/templates/#tutorial-templates)

```
{{ some_variable }}
```

This is getting the variable and is going to display it on our website.

So if in the python we have:

Python

```python
# ...
# ...
name = 'Chris'
return render_template('page_name.html', name=name)
```

in the HTML we can now do

Markup

```
<h1>{{ name }}</h1>
```

and it will come up as `<h1>Chris</h1>`

This is really cool so now we can ask for things in the database and show it on our webpage dynamically :) Yay

Make sure you play around with this, it can be very powerful, just check out the files.

That lays the basic foundations of what we need to do today. Some interesting files are the `units.html` and the `routes.py` with `database.py` for units of study, it shows you how we can run through our files.

Don't be scared to look on the internet, I've linked the tutorials for the packages, so that should be nice.

If you're having trouble make sure to **ask your tutor**, post on **ed** or you can email me: christopher.natoli@sydney.edu.au