MySQL

1. Time to install MySQL
    a. Go to dev.mysql.com/downloads/mysql
    b. Download the Mac OS X 10.8 (x86, 64-bit), DMG archive
        i. or get the DMG that works for your OS and processor (32 or 64 bit). I'm assuming that you've updated your OS and have a newish computer.
        ii. You'll get directed to a page asking you to login or signup. Ignore. Click the link "No thanks, just start my download", which is in really tiny font on lower left.
    c. Install using dmg: Right click to do "Open with" and select the installer
       (You may get a message saying that it can't be opened because it is from an unidentified developer if you just double click.)
2. Download MySQL Workbench and install.
3. Download the world_innodb.sql dump from here, and unzip
4. Move this file into the directory where you have been working
5. Open a terminal and cd to your working directory.
   Pro tip! If you want select a file with your mouse and open a terminal that's already at the directory that contains that file, you can set this up:
        i. Mac: To do this easily, check here.
        ii. Ubuntu: 14.04 LTS, check here.

if mysqld: command not found:
mysqld is in: /usr/local/mysql/bin/mysql (and so is mysql)
run the following at command line:

```
$ sudo ln -s /usr/local/mysql/bin/mysql /usr/bin/mysql
$ sudo ln -s /usr/local/mysql/bin/mysqld /usr/bin/mysqld
```

6. Start MySQL server: **mysqld**
    a. If you have problems, check out this page.

if mysql gives an ERROR 200 when you run it on a Mac:
go to System Preferences: MySql: Start Server.
then
$ mysql
will work

7. Create table and load data.
    a. Option 1:
        i. `shell> mysql -u root -p`

           if it asks for password and you don't have one, just press Enter.
        ii. `mysql> CREATE DATABASE world_innodb;`

           Result: Query OK, 1 row affected (0.01 sec)

iii. `mysql> USE world_innodb;`

   Database changed

iv. `mysql> SOURCE world_innodb.sql;`

   If you are in a different directory than your file, you'll get an error. You should see lots of stuff!

v. `mysql> SHOW TABLES;`

```
+-----------------+
| Tables_in_world |
+-----------------+
| City            |
| Country         |
| CountryLanguage |
+-----------------+
```

b. Option 2 -- easier! That's why it's option 2.

i.
```
>> mysql -u root -p
[Enter password]
[mysql] > CREATE DATABASE world_innodb;
[mysql] > EXIT;
>> mysql world_innodb <world_innodb.sql
```

ii. `mysql> SHOW TABLES;`

8.
```
>> mysql -u root -p
mysql> USE world_innodb;
mysql> SHOW TABLES;
mysql> SELECT * FROM City LIMIT 10;
+----+----------------+-------------+---------------+------------+
| ID | Name           | CountryCode | District      | Population |
+----+----------------+-------------+---------------+------------+
|  1 | Kabul          | AFG         | Kabol         |    1780000 |
|  2 | Qandahar       | AFG         | Qandahar      |     237500 |
|  3 | Herat          | AFG         | Herat         |     186800 |
|  4 | Mazar-e-Sharif | AFG         | Balkh         |     127800 |
|  5 | Amsterdam      | NLD         | Noord-Holland |     731200 |
|  6 | Rotterdam      | NLD         | Zuid-Holland  |     593321 |
|  7 | Haag           | NLD         | Zuid-Holland  |     440900 |
|  8 | Utrecht        | NLD         | Utrecht       |     234323 |
|  9 | Eindhoven      | NLD         | Noord-Brabant |     201843 |
| 10 | Tilburg        | NLD         | Noord-Brabant |     193238 |
+----+----------------+-------------+---------------+------------+
10 rows in set (0.00 sec)
```

9. Yay! You have MySQL working and you've just queried some data. Note that this is structured data, like a spreadsheet. You can learn more about how to use SQL and select for the data that you want on SQLZoo or Mode Analytics SQL School. But let's keep going for now.

10. [Optional] Open MySQL Workbench.
    Click on the top left box “Local Instance blah” to see if a new connection pops up with a white box and blinking cursor at a number 1.
    Pro tip! You are creating an instance on localhost (127.0.0.1) as root (which is frowned upon) but you can create a user account for yourself later and grant the necessary permissions. No one does this.
11. [Optional] In MySQL Workbench, make a connection to the world_innodb table and explore. In the white box, type `USE world_innodb;` then `SELECT * FROM City LIMIT 10;`

PyMySQL: a MySQL connector to Python

1. >>[sudo] pip install pymysql
2. >>mysql -u root -p
3. >>CREATE DATABASE testdb;
4. >>USE testdb;
5. >>EXIT;
6. Create a new file in the working directory called test_db.py with the following content:

```
import pymysql as mdb

con = mdb.connect('localhost', 'root', '', 'testdb') #host, user, password,
#database

with con:
    cur = con.cursor()
    cur.execute("DROP TABLE IF EXISTS Writers")
    cur.execute("CREATE TABLE Writers(Id INT PRIMARY KEY AUTO_INCREMENT,Name
VARCHAR(25))")
    cur.execute("INSERT INTO Writers(Name) VALUES('Jack London')")
    cur.execute("INSERT INTO Writers(Name) VALUES('Honore de Balzac')")
    cur.execute("INSERT INTO Writers(Name) VALUES('Lion Feuchtwanger')")
    cur.execute("INSERT INTO Writers(Name) VALUES('Emile Zola')")
    cur.execute("INSERT INTO Writers(Name) VALUES('Truman Capote')")

with con:
    cur = con.cursor()
    cur.execute("SELECT * FROM Writers")
    rows = cur.fetchall()
    for row in rows:
        print row
```

7. Run the file: ipython test_db.py
8. You just used Python to write data into MySQL (the first ‘with con’ block) and then to query it (the second ‘with con’ block). Bam!

Flask: a micro web development framework (Shamelessly copied from here.)

1. If you haven’t installed Flask yet, install Flask.
    a. [sudo] pip install flask

2. Make some directories
```
mkdir app
mkdir app/static
mkdir app/templates
mkdir tmp
```
3. The `app` folder will be where we will put our application package. The `static` sub-folder is where we will store static files like images, javascripts, and style sheets. The `templates` sub-folder is obviously where our templates will go.
4. Make a new file called `app/__init__.py` with the following content:
```
from flask import Flask
app = Flask(__name__)
from app import views
```
5. The script above simply creates the application object (of class `Flask`) and then imports the views module, which we haven't written yet.
6. The views are the handlers that respond to requests from web browsers. In Flask views are written as Python functions. Each view function is mapped to one or more request URLs.
7. Let's write our first view function in a new file called `app/views.py`:
```
from app import app

@app.route('/')
@app.route('/index')
def index():
    return "Hello, World!"
```
8. This view is actually pretty simple, it just returns a string, to be displayed on the client's web browser. The two `route` decorators above the function create the mappings from urls `/` and `/index` to this function.
9. Next we need to point to python. On the command line, type `which python` and note the path.
10. The final step to have a fully working web app is to create a script that starts up the development web server with our application. Let's make a new file called `run.py` (in the folder a level above app):
```
#!(the path to python goes here)
from app import app
app.run(debug = True)
```
    The script simply imports the `app` variable from our app package and invokes its `run` method to start the server. Remember that the `app` variable holds the `Flask` instance, we created it above.

11. To start the app you just run this script. First you have to indicate that this is an executable file before you can run it: `chmod a+x run.py`
12. Then the script can simply be executed as follows: `./run.py`
13. After the server initializes it will listen on port 5000 waiting for connections. Now open up your web browser and enter the following URL in the address field:

    http://localhost:5000

This will also work: http://localhost:5000/index

Do you see the route mappings in action? The first URL maps to `/`, while the second maps to `/index`. Both routes are associated to our view function, so they produce the same result. If you enter any other route you will get an error, since only these two have been mapped to a view function.

14. When you are done playing with the server you can just hit Ctrl-C in the Terminal to stop it.

Flask Templates

1. You should have this in your directory:

```
app\
        static\
        templates\
        __init__.py
        views.py
tmp\
run.py
```

2. Option 1 to expand your app is to edit your views file to be this:

```
from app import app

@app.route('/')
@app.route('/index')
def index():
    user = { 'nickname': 'Miguel' } # fake user
    return '''
<html>
  <head>
    <title>Home Page</title>
  </head>
  <body>
    <h1>Hello, ''' + user['nickname'] + '''</h1>
  </body>
</html>
'''
```

I hope you agree that the solution above is very ugly. Consider how complex the code will become if you have to return a large and complex HTML page with lots of dynamic content. And what if you need to change the layout of your web site in a large app that has dozens of views, each returning HTML directly? This is clearly not a scalable option.

Templates to the rescue! If you could keep the logic of your application separate from the layout or presentation of your web pages things would be much better organized, don't you think? You could even hire a web designer to create a killer web site while you code the site's behaviors in Python. Templates help implement this separation.

3. Let's write our first template in a new file `app/templates/index.html` :

```
<html>
  <head>
     <title>{{title}} - microblog</title>
  </head>
  <body>
     <h1>Hello, {{user.nickname}}!</h1>
  </body>
</html>
```

As you see above, we just wrote a mostly standard HTML page, with the only difference that there are some placeholders for the dynamic content enclosed in `{{ ... }}` sections.

4. Now let's edit the file `app/views.py` to the following:

```
from flask import render_template
from app import app

@app.route('/')
@app.route('/index')
def index():
    user = { 'nickname': 'Miguel' } # fake user
    return render_template("index.html",
        title = 'Home',
        user = user)
```

5. Try the application at this point to see how the template works.
   a. `./run.py`
   b. go to http://localhost:5000/ -- Hello Miguel!

Once you have the rendered page in your browser you may want to view the source HTML and compare it against the original template.

To render the template we had to import a new function from the Flask framework called `render_template`. This function takes a template name and a variable list of template arguments and returns the rendered template, with all the arguments replaced.

Under the covers, the `render_template` function invokes the Jinja2 templating engine that is part of the Flask framework. Jinja2 substitutes `{{...}}` blocks with the corresponding values provided as template arguments.

We just covered about half of what's found at http://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-ii-templates, but you should read the following sections later: Control statements in templates, Loops in templates, and Template inheritance.

Time to Query and display data from SQL

1. Let's make a different page where we can pull data from SQL. Change your views.py file to the following:

```
from flask import render_template
from app import app
import pymysql as mdb

db = mdb.connect(user="root", host="localhost", db="world_innodb", charset='utf8')

@app.route('/')
@app.route('/index')
def index():
       return render_template("index.html",
        title = 'Home', user = { 'nickname': 'Miguel' },
        )

@app.route('/db')
def cities_page():
       with db:
              cur = db.cursor()
              cur.execute("SELECT Name FROM city LIMIT 15;")
              query_results = cur.fetchall()
       cities = ""
       for result in query_results:
              cities += result[0]
              cities += "<br>"
       return cities
```

2. Check your python indentation matches that of above, save and do run.py, go to http://localhost:5000/ -- you'll still see Miguel there.

   Go to http://localhost:5000/db -- We just pulled data from the world_innodb database and displayed it on the website! But, it looks… not so great at this point.

Twitter Bootstrap time!

1. Go to http://getbootstrap.com/ and download Bootstrap.
2. When you extract the file, you'll find the directories: `css`, `fonts`, and `js`.
3. Copy these directories into your `app/static` folder.
4. Let's create an html template in the `app/templates` folder. This time, we're going to use Twitter Bootstrap to make stuff pretty. Look at some templates. Let's use the Starter Template.
   a. Click Starter Template.
   b. View the page source (right-click in Chrome for example) and copy that into a new file called cities.html.
   c. Now, open cities.html using your web browser. (On the command line, you can say "open cities.html" on a mac.) It should open in a browser and not look the same. In fact, it should look really bad.
   d. Go back and edit cities.html so that the path to Bootstrap core CSS is correct.
      i. `<link href="../static/css/bootstrap.min.css" rel="stylesheet">`
      ii. Open it again -- This should look almost right!

e. Go to the `starter-template.css` and create a file with that content and the same name in the same directory as cities.html
   i. Open it again (open cities.html) and it should look nice.

3. Change the cities.html in the following highlighted areas.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="description" content="">
    <meta name="author" content="">
    <link rel="icon" href="../../favicon.ico">

    <title>Starter Template for Bootstrap</title>

    <!-- Bootstrap core CSS -->
    <link href="/static/css/bootstrap.min.css" rel="stylesheet">

    <!-- Custom styles for this template -->
    <link href="starter-template.css" rel="stylesheet">

    <!-- Just for debugging purposes. Don't actually copy these 2 lines! -->
    <!--[if lt IE 9]><script
src="../../assets/js/ie8-responsive-file-warning.js"></script><![endif]-->
    <script src="../../assets/js/ie-emulation-modes-warning.js"></script>

    <!-- IE10 viewport hack for Surface/desktop Windows 8 bug -->
    <script src="../../assets/js/ie10-viewport-bug-workaround.js"></script>

    <!-- HTML5 shim and Respond.js IE8 support of HTML5 elements and media queries
-->
    <!--[if lt IE 9]>
      <script
src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js"></script>
      <script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
    <![endif]-->
  </head>

  <body>

    <div class="navbar navbar-inverse navbar-fixed-top" role="navigation">
      <div class="container">
        <div class="navbar-header">
          <button type="button" class="navbar-toggle" data-toggle="collapse"
data-target=".navbar-collapse">
            <span class="sr-only">Toggle navigation</span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
          </button>
          <a class="navbar-brand" href="#">Project name</a>
        </div>
        <div class="collapse navbar-collapse">
          <ul class="nav navbar-nav">
            <li class="active"><a href="#">Home</a></li>
            <li><a href="#about">About</a></li>
            <li><a href="#contact">Contact</a></li>
          </ul>
        </div><!--/.nav-collapse -->
      </div>
    </div>

<br><br>

    <div class="container">
      <div class="starter-template">
        <h1>Bootstrap starter template</h1>
        <p class="lead">Use this document as a way to quickly start any new
project.<br> All you get is this text and a mostly barebones HTML document.</p>
      </div>
    <table class="table table-hover">
    <tr><th>Name</th><th>Country</th><th>Population</th></tr>
    {% for city in cities %}
    <tr><td>{{ city['name'] }}</td><td>{{ city['country']}}</td><td> {{
city['population'] }}</td></tr>
    {% endfor %}
    </table>
    <script src="https://code.jquery.com/jquery-1.10.2.min.js"></script>
    <script src="static/dist/js/bootstrap.min.js"></script>

    </div><!-- /.container -->


    <!-- Bootstrap core JavaScript
    ================================================== -->
    <!-- Placed at the end of the document so the pages load faster -->
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
    <script src="../../dist/js/bootstrap.min.js"></script>
  </body>
</html>
```

4. Edit the views.py again for the bolded text:

```
from flask import render_template
from app import app
import pymysql as mdb

db= mdb.connect(user="root", host="localhost", db="world_innodb", charset='utf8')
```

```
@app.route('/')
@app.route('/index')
def index():
       return render_template("index.html",
         title = 'Home',
         )

@app.route('/db')
def cities_page():
       with db:
              cur = db.cursor()
              cur.execute("SELECT Name FROM city LIMIT 15;")
              query_results = cur.fetchall()
       cities = ""
       for result in query_results:
              cities += result[0]
              cities += "<br>"
       return cities

@app.route("/db_fancy")
def cities_page_fancy():
       with db:
              cur = db.cursor()
              cur.execute("SELECT Name, CountryCode,
                     Population FROM city ORDER BY Population LIMIT 15;")

              query_results = cur.fetchall()
       cities = []
       for result in query_results:
              cities.append(dict(name=result[0], country=result[1],
population=result[2]))
       return render_template('cities.html', cities=cities)
```

Double-check your python indentation again!

5. Now, if you go to http://127.0.0.1:5000/db_fancy, you should see a pretty-ified version of the table. Oh, the power of bootstrap.