
Lab #18

Flask

The lab is due on Monday, April 20th by the end of the calendar day. It is worth 5 points. Submit your flaskapp.py file via blackboard.

Part 1: Introduction to Flask

In this lab, you'll be creating a web server that has the ability to query a SQL database. In this case, we'll be using a web framework called Flask.

What is Flask?

Flask is a **web framework**. A web framework provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website. Flask uses the programming language Python. Applications that use the Flask framework include Pinterest and LinkedIn.

We'll be using Flask to create an interface to interact with (query) a database.

Part 2: Establishing a Flask Application on your local Machine

It's been my experience that developing web applications is much easier to be done on your local machine rather than in the cloud or on an external web server. This lab will be done entirely on your laptop. In the future, we'll move all of the files to a web server to make them publicly available. However, in the interest of writing and debugging the code, staying local is best.

This means you will need to make sure that you have Python installed in your local environment, and that you can execute it from the command line. We are also going to install the flask module for Python using pip, the Python package management utility.

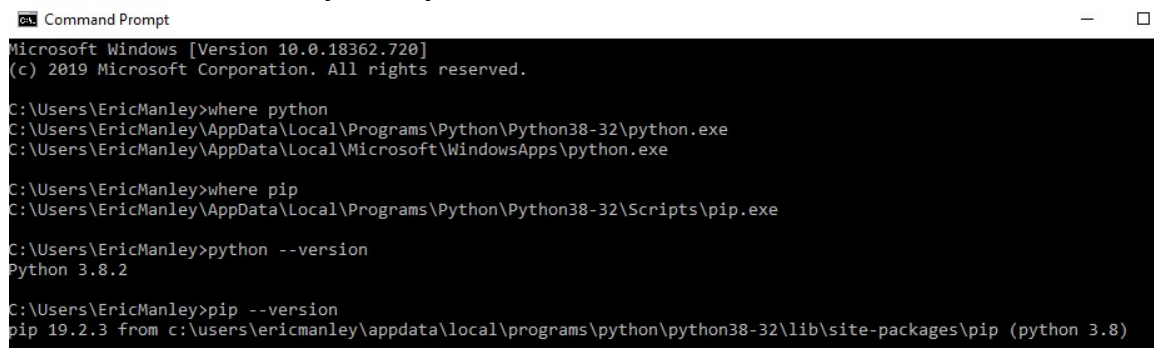
To see if your terminal/console knows where to find Python and pip, you can try the `which` (Mac/Linux/Windows-Ubuntu) or `where` (Windows command prompt) command. This should show you the directory where it's running Python/pip from. To see which version of these you're running, you can execute them with the `--version` option as shown here:

Mac Example

```
(base) Coll-Manley:~ 000794593$ which python
//anaconda3/bin/python
(base) Coll-Manley:~ 000794593$ which pip
//anaconda3/bin/pip
(base) Coll-Manley:~ 000794593$ python --version
Python 3.7.3
(base) Coll-Manley:~ 000794593$ pip --version
pip 19.1.1 from //anaconda3/lib/python3.7/site-packages/pip (python 3.7)
```

Ubuntu-Windows Example

```
manley@MSI:~$ which pip
/usr/bin/pip
manley@MSI:~$ which python
/usr/bin/python
manley@MSI:~$ python --version
Python 2.7.17
manley@MSI:~$ pip --version
pip 9.0.1 from /usr/lib/python2.7/dist-packages (python 2.7)
```

Windows Command Prompt Example


```
Command Prompt
Microsoft Windows [Version 10.0.18362.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\EricManley>where python
C:\Users\EricManley\AppData\Local\Programs\Python\Python38-32\python.exe
C:\Users\EricManley\AppData\Local\Microsoft\WindowsApps\python.exe

C:\Users\EricManley>where pip
C:\Users\EricManley\AppData\Local\Programs\Python\Python38-32\Scripts\pip.exe

C:\Users\EricManley>python --version
Python 3.8.2

C:\Users\EricManley>pip --version
pip 19.2.3 from c:\users\ericmanley\appdata\local\programs\python\python38-32\lib\site-packages\pip (python 3.8)
```

What to look for

There are a few things to look for here – usually these will all be fine, but occasionally they’re messed up and can cause problems later:

1. Python and pip should be running from the same directory – otherwise it means your pip is probably the pip that goes with a *different* distribution of Python somewhere else on your computer.
2. Double-check that pip is for the version of Python you’re using. I’ve tested this lab in a version of Python 2 as well as 3, so I think you should be ok using whichever one you’re comfortable with.

If something isn’t installed or doesn’t match up, I suggest doing a new install of the Python and pip. Here are some instructions on how to do that for different systems:

Ubuntu-Windows Python/pip install instructions

If you are using Ubuntu-Windows and you need to install Python or pip, use the following command to update your Ubuntu installer:

```
sudo apt-get update
```

Then install python and/or pip with the command

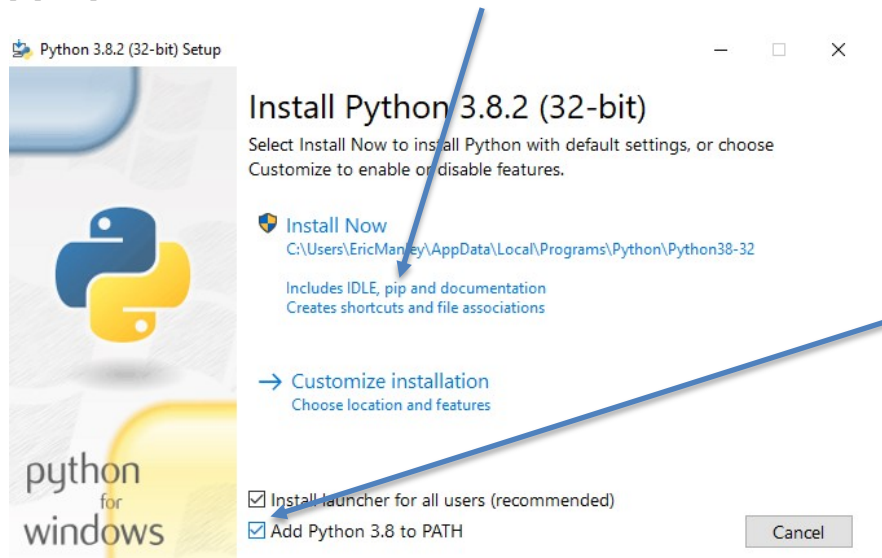
```
sudo apt install python
sudo apt install python-pip
```

Mac Python/pip install instructions

If you are using Mac, download and run the installer from <https://www.python.org/downloads/> . It should automatically install pip, and your terminal should usually be able to find them (at least after a re-start).

Windows (command prompt) Python/pip install instructions

If you are a non-Ubuntu Windows user, download and run the installer from <https://www.python.org/downloads/> . When you get to this install screen, make sure that you check the box to have it add Python to your PATH, and note that it should be installing pip as part of it.



You can then re-start the command prompt and Python/pip should work.

Installing the Flask module for Python (everyone needs to do this)

After ensuring Python and pip are installed correctly, run the following pip command to install the Flask module.

```
pip install flask
```

If that command gives you some kind of permissions error, you may need to run it as an administrator, usually with this command:

```
sudo pip install flask
```

Part 3. Creating a Flask App

Let's create a "hello world" Flask App.

1. Create a folder in your CS178 folder called "FlaskApp"
2. Open a terminal window and cd to this directory
3. Open Sublime (or any text editor) and create a file called flaskapp.py. Save the file to your FlaskApp directory
4. and enter the following code:

```
from flask import Flask
app = Flask(__name__)

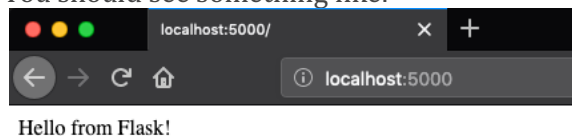
@app.route('/')
def hello():
    return 'Hello from Flask!'

# these two lines of code should always be the last in the file
if __name__ == '__main__':
    app.run(debug=True)
```

5. The most important piece of the above Python script is the line that begins with `@app.route('/')`. This indicates that if the user enters in the URL of the webserver (the IP address) the `hello()` function will be called. In this case, it will just return "Hello from Flask!". Let's try it.
6. After saving the flaskapp.py file, you'll need to start a *webserver* using flask on your machine. In your terminal, enter:

```
python flaskapp.py
```

7. Open a web browser and enter the url: <http://localhost:5000/>
You should see something like:



What is 5000? This is the default port that our web page will be hosted on.

One of the most important concepts in Flask is that of "routes". These are the different paths (or strings) we type into the browser to go to different pages. So far, we've defined one route that is simply "/"

The route is going to return a string that is essentially HTML code that the page will display.

8. Update the flaskapp.py file to look like this:

```

1 from flask import Flask
2 app = Flask(__name__)
3
4 @app.route('/')
5 def hello():
6     return '<h1>Hello from Flask!</h1>'
7
8 @app.route('/about')
9 def about():
10    return '<h2>An about page!</h2>'
11
12 if __name__ == '__main__':
13    app.run(debug=True)

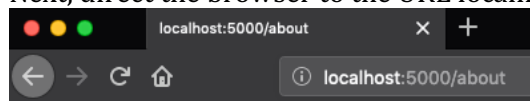
```

Note the addition of html tags around the strings

Note the additional route for the path “/about”

9. Now, refresh your web browser. You should note that the “Hello from Flask!” looks larger.

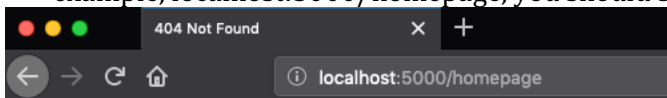
10. Next, direct the browser to the URL localhost:5000/about



An about page!

Note that the page shows up.

11. If you were to navigate to a location that our site hasn't yet implemented, for example, localhost:5000/homepage, you should see a Not Found error.



Not Found

The requested URL was not found on the server. If you entered the URL manually

In the terminal, you should also see a response that indicates a 404 (not found) error

```
127.0.0.1 - - [14/Apr/2020 12:15:41] "GET /homepage HTTP/1.1" 404 -
```

If the entire website was just supplying html via Python functions, the website would be messy very quickly. Flask templates will help organize things a bit.

Part 4. Flask Templates

One of the main ideas behind a web framework such as Flask is that a template can be established that provides much of the html code to be generated, then the important components can be created using a programming language such as Python. We'll put some starter html code in a `templates` directory so that all the Python code has to do is reference that html file and it will be formatted as we expect it to be.

1. Create a folder called `templates` within the `FlaskApp` folder
2. Download `layout.html` file from blackboard and place them in the `templates` folder.
3. Open the `layout.html` file in Sublime, (also shown below). Note that it uses a CSS (cascading style sheet) on lines 4 - 17 to help define how HTML elements (h1 tags) should be displayed.
4. Also note the syntax of `{{name}}` on line 22. This will be how we can pass some information in from the `flaskapp.py` file into this formatting file.

```

1 <html>
2 <head>
3   <title>Website</title>
4 <style>
5 @import url(http://fonts.googleapis.com/css?family=Amatic+SC:700);
6
7 body{
8   text-align: center;
9 }
10 h1{
11   font-family: 'Amatic SC', cursive;
12   font-weight: normal;
13   color: #8ac640;
14   font-size: 2.5em;
15 }
16
17 </style>
18
19 </head>
20 <body>
21 <div class="block1">
22 <h1>Hello {{name}}!</h1>
23   <h2>Here is an interesting xkcd cartoon for you: </h2>
24 
25 </div>
26 </body>
27 </html>

```

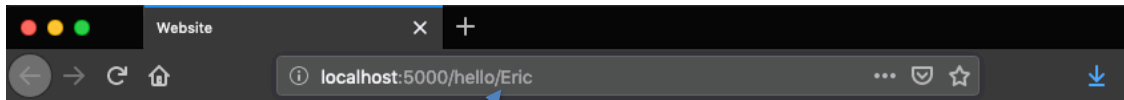
Next, add the following code to `flaskapp.py`: (add this to the end of the code already in the file, but *before* the call to `if __name__ == '__main__': ...`)

```
from flask import render_template

@app.route("/hello/<username>/")
def hello_user(username):
    return render_template('layout.html', name=username)

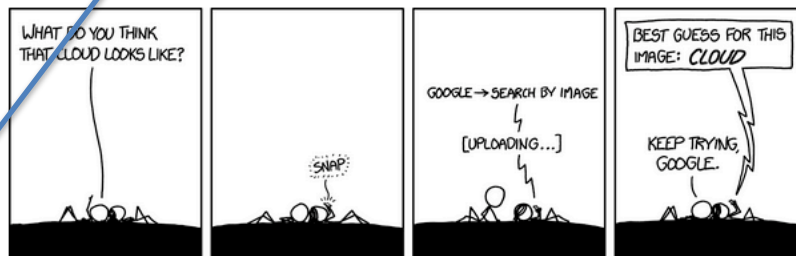
# this line of code should be the last in the file
if __name__ == '__main__':
    app.run(debug=True)
```

Now, you should be able to use a browser and point to your localhost:5000, followed by /hello/your-name. Note that your-name will be passed as the parameter <username> into the python code. This will be passed into the layout.html which will display the page using the html template. The result should look like this:



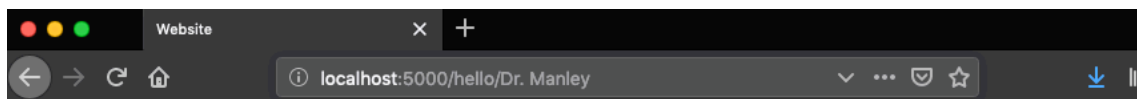
HELLO ERIC!

Here is an interesting xkcd cartoon for you:



Notice how the string after IPaddress/hello/ matches the text in the webpage!!!

Try it with different strings!!



HELLO DR. MANLEY!

Part 5. Flask and Python

Next, we'll explore how we can use Python explicitly within flask web pages.

The `@app.route('path')` is a critical piece of syntax in Flask. This part of the code defines how a particular function will be called. If the 'path' matches the string following the IP address, the function that follows the `@app.route('path')` will be called.

Note that the flask Functions can also take a parameter. Adding `<parameter>` to the route path will allow the Python function to incorporate the input into its calculations

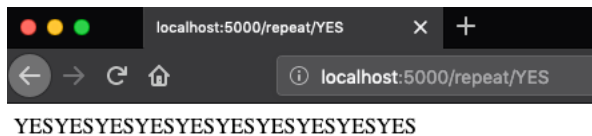
Next, add the following code at the end of `flaskapp.py` (before the `if __name__` part):

```
@app.route("/repeat/<var>")
def repeater(var):
    result = ""
    for i in range(10):
        result += var
    return result
```

Next, direct your browser at your `localhost:5000` followed by `/repeat/YES`

(note that `repeat` is specified in the path, and can be considered the "name" of the function called)

You should see the string YES repeated ten times.



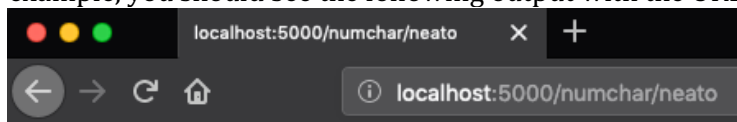
Challenge #1

Create a function that will return the count of the number of characters in the input. It should be called with the syntax

```
@app.route("/numchar/<var>")
def numchar(var):
```

Note that you will need to return a string value of the number.

For example, you should see the following output with the URL `/numchar/neato`

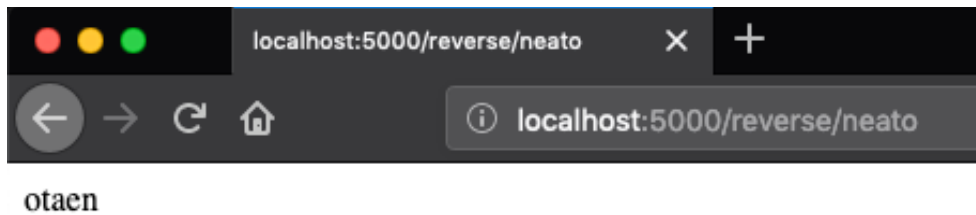


Challenge #2

Create a function that will REVERSE an input string. It should be called with the syntax

```
@app.route("/reverse/<var>")  
def reverse(var):
```

Feel free to use google to help with how to reverse a string using Python. You should see the following output with the URL `/reverse/neato`

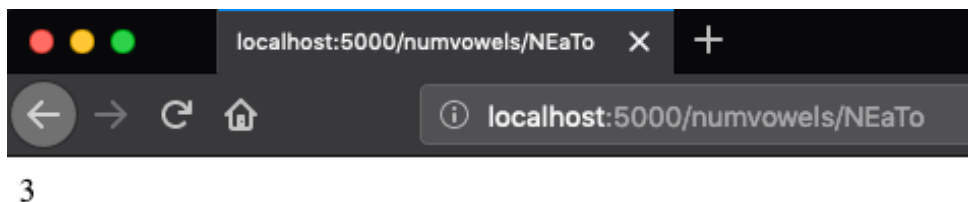


Challenge #3

Create a function that will return the number of vowels in the input (assume y is NOT a vowel). It should be called with the syntax:

```
@app.route("/numvowels/<var>")  
def numvowels(var):
```

You should see the following output with the URL `/numvowels/NEaTo`



Part 6. SQLite

What is SQLite?

SQLite is a library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. In short, it is a light-weight database that uses SQL syntax but doesn't support multi-users or concurrency. Unlike most other SQL

databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files.

I've created a SQLite database of national parks in the United States from a database acquired from here:

https://en.wikipedia.org/wiki/List_of_areas_in_the_United_States_National_Park_System#National_parks

I've put the data into a file called `natlpark.db`

Download the database file from blackboard and put it into your FlaskApp folder

Note that the data has 4 fields: Name of the Park, State, year the park was founded, and the area of the park (in acres). The code used to create the table looks like this:

```
CREATE TABLE natlpark (name text, state text, year integer, area float)
```

Part 7. Connecting the Database with Flask

Copy and paste the following code at the end of `flaskapp.py`. I've added a file, `dbcode.txt` to blackboard for copy-and-pasting. Make sure the last line of the `.py` file is still

```
# this line of code should be the last in the file
if __name__ == '__main__':
    app.run(debug=True)
```

The code in `dbcode.txt`:

```
import csv
import sqlite3

from flask import Flask, request, g

DATABASE = 'natlpark.db'

app.config.from_object(__name__)

def connect_to_database():
    return sqlite3.connect(app.config['DATABASE'])

def get_db():
    db = getattr(g, 'db', None)
    if db is None:
        db = g.db = connect_to_database()
    return db

@app.teardown_appcontext
def close_connection(exception):
    db = getattr(g, 'db', None)
    if db is not None:
        db.close()

def execute_query(query, args=()):
    cur = get_db().execute(query, args)
    rows = cur.fetchall()
    cur.close()
    return rows

#display the sqlite query in a html table
def display_html(rows):
    html = ""
    html += "<table><tr><th>Park</th><th>State</th><th>Year</th><th>Area</th></tr>"
    for r in rows:
        html += "<tr><td>" + str(r[0]) + "</td><td>" + str(r[1]) + "</td><td>" + str(r[2]) + "</td><td>" + str(r[3]) + "</td></tr>"
    html += "</table></body>"
    return html

@app.route("/viewdb")
def viewdb():
    rows = execute_query("SELECT * FROM natlpark")
    return display_html(rows)
```

The last three methods are the most noteworthy.

```
def execute_query(query, args=()):
```

will execute the sql query as input and returns the resulting rows as a list of strings;

```
def display_html(rows):
```

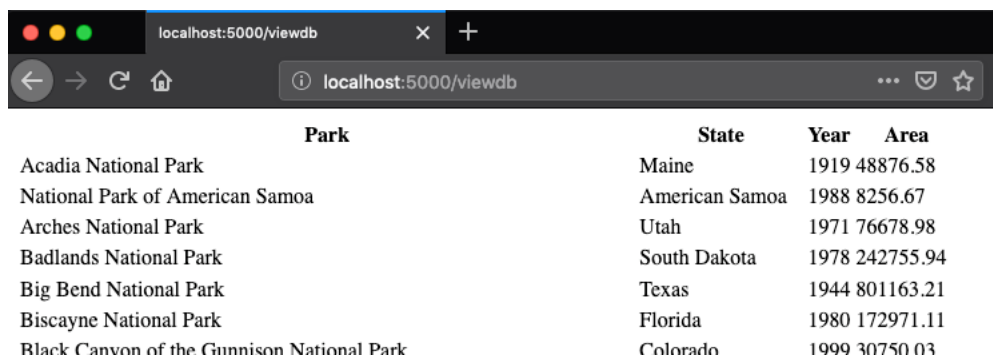
will take a list of rows as a parameter and output an HTML table

```
@app.route("/viewdb")
def viewdb():
    rows = execute_query("""SELECT * FROM natlpark""")
    return display_html(rows)
```

will execute the SQL query `SELECT * FROM natlpark` and will return an HTML table of the results.

Give it a try. Use `localhost:5000/viewdb`

You should see:



The screenshot shows a web browser window with the address bar displaying `localhost:5000/viewdb`. The browser window contains a table with the following data:

Park	State	Year	Area
Acadia National Park	Maine	1919	48876.58
National Park of American Samoa	American Samoa	1988	8256.67
Arches National Park	Utah	1971	76678.98
Badlands National Park	South Dakota	1978	242755.94
Big Bend National Park	Texas	1944	801163.21
Biscayne National Park	Florida	1980	172971.11
Black Canyon of the Gunnison National Park	Colorado	1999	30750.03

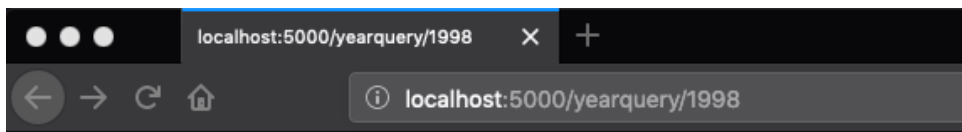
Note that you are using SQL commands within Python within a web page. Be impressed!!

Next, we could use the variable-input via URL to input query specifics for the database. For example, add the following function to `flaskapp.py`

```
@app.route("/yearquery/<year>")
def viewyears(year):
    rows = execute_query("""SELECT * FROM natlpark WHERE year > ? order by year""", [year])
    return display_html(rows)
```

Note that the `<year>` input will be placed into the `?` part of the query, resulting in a query of the database of all of the parks that were founded later than the input year.

Inputting `localhost:5000/yearquery/1998` into the browser will display all parks founded after the year 1998. Give it a try.



localhost:5000/yearquery/1998

Park	State	Year	Area
Black Canyon of the Gunnison National Park	Colorado	1999	30750.03
Cuyahoga Valley National Park	Ohio	2000	32831.18
Congaree National Park	South Carolina	2003	26275.82
Great Sand Dunes National Park and Preserve	Colorado	2004	94687.84
Pinnacles National Park	California	2013	26605.73

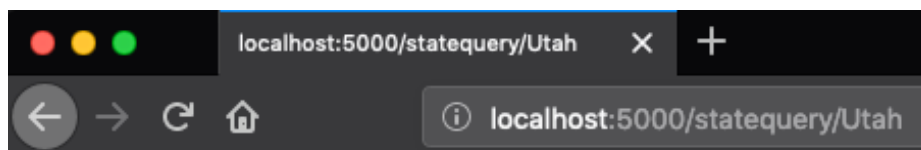
Experiment with different years.

Challenge #4

Create a function that will allow the URL to specify the state of the parks that will be output in the resulting table. The syntax should look like this:

```
@app.route("/statequery/<st>")
def viewstates(st):
```

The result should allow the user to query the parks from a specific state, and list the park name alphabetically. For example, the parks from Utah are listed below.



localhost:5000/statequery/Utah

Park	State	Year	Area
Arches National Park	Utah	1971	76678.98
Bryce Canyon National Park	Utah	1928	35835.08
Canyonlands National Park	Utah	1964	337597.83
Capitol Reef National Park	Utah	1971	241904.26
Zion National Park	Utah	1919	147237.02

Part 8. An interface with Flask and SQLite

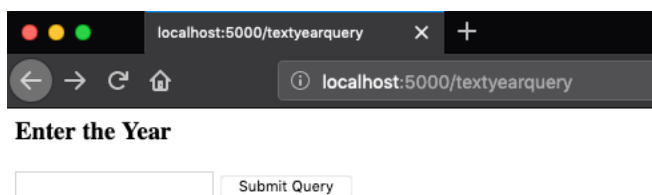
The query approach using the URL is nice, but not the most user-friendly. Let's explore how messages can be sent with Flask using text boxes instead of URLs. Download one more template (textbox.html) to be placed into the templates folder

Add the following code to flaskapp.py file.

```
@app.route('/textyearquery')
def year_form():
    return render_template('textbox.html', fieldname = 'Year')

@app.route('/textyearquery', methods=['POST'])
def year_form_post():
    text = request.form['text']
    return viewyears(text)
```

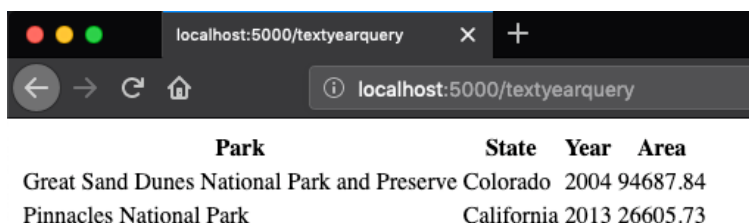
When the path of IP-address/textyearquery is in the browser, it will render the templet for textbox.html which should just look like this



Enter the Year

When a year is typed in the box, and the “submit” button is pressed, it generates a “post” event, which calls the `@app.route('/textyearquery', methods=['POST'])` command. As a result, the text from the form gets passed to the `viewyears(text)` method, and the appropriate table is generated.

Typing in 2003 in the input and pressing Submit generates:



Park	State	Year	Area
Great Sand Dunes National Park and Preserve	Colorado	2004	94687.84
Pinnacles National Park	California	2013	26605.73

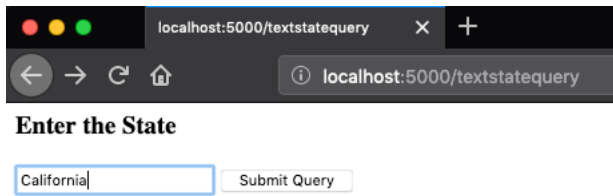
Challenge #5

Create a function with the following syntax that will allow the user to enter the **State** in a textbox and pressing the button will result in the appropriate values to be displayed.

```
@app.route('/textstatequery')
def state_form():
    return render_template('textbox.html', fieldname = 'State')

@app.route('/textstatequery', methods=['POST'])
def state_form_post():
```

For example, typing in California

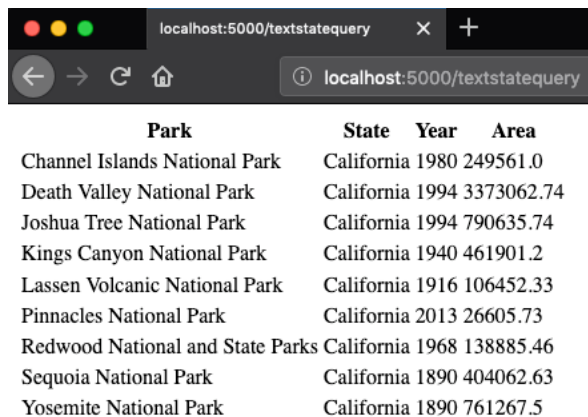


localhost:5000/textstatequery

Enter the State

California Submit Query

Should generate this table:



Park	State	Year	Area
Channel Islands National Park	California	1980	249561.0
Death Valley National Park	California	1994	3373062.74
Joshua Tree National Park	California	1994	790635.74
Kings Canyon National Park	California	1940	461901.2
Lassen Volcanic National Park	California	1916	106452.33
Pinnacles National Park	California	2013	26605.73
Redwood National and State Parks	California	1968	138885.46
Sequoia National Park	California	1890	404062.63
Yosemite National Park	California	1890	761267.5

Upload your flaskapp.py to blackboard for consideration for the points for this lab.

=====

References used in creating this lab:

<https://www.sqlite.org/about.html>
<https://pythonspot.com/flask-web-app-with-python/>
<https://www.datasciencebytes.com/bytes/2015/02/28/using-flask-to-answer-sql-queries/>