



Rendering Your Data With Flask

Data Boot Camp
Lesson 12.3



WELCOME



Class Objectives

By the end of today's class you will be able to:



Render templates with Flask using data retrieved from Mongo database.



Use Beautiful Soup to scrape data.



Use PyMongo to save data to a Mongo database.



Use Flask to render templates.



Instructor Demonstration

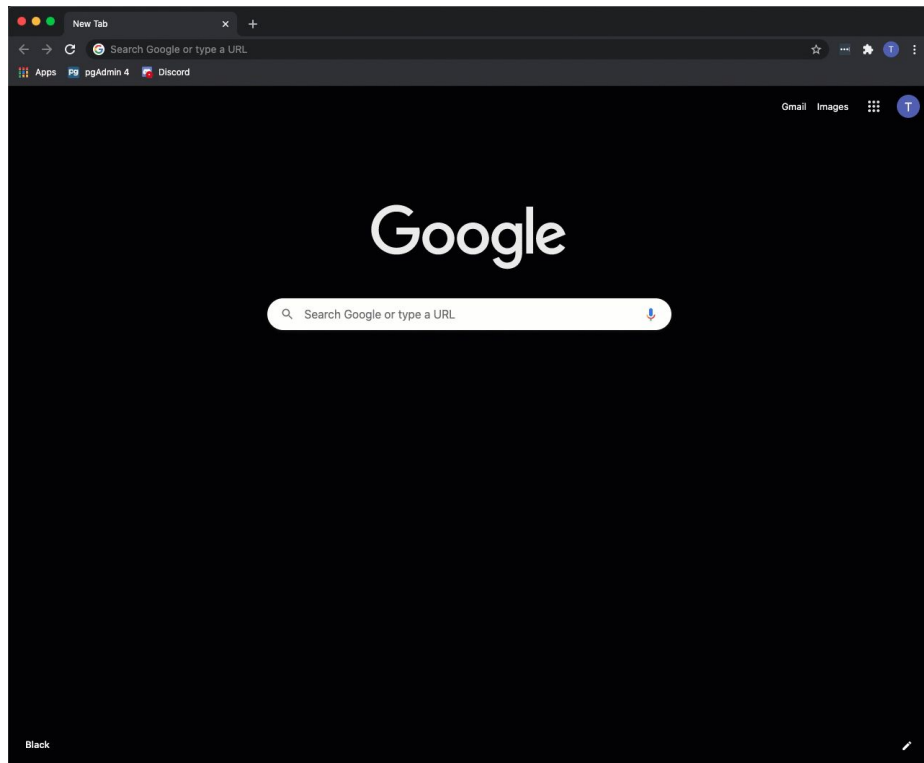
Intro to Flask Render

Intro to Flask Render

How it works - The basics of rendering a template with Flask!



1. In your CLI navigate to the appropriate folder.
2. Run: `python app.py`
3. Open your browser and visit



Intro to Flask Render

/app.py
/templates
 /index.html



```
1 # import necessary libraries
2 from flask import Flask, render_template
3
4 # create instance of Flask app
5 app = Flask(__name__)
6
7
8 # create route that renders index.html template
9 @app.route("/")
10 def echo():
11     return render_template("index.html", text="Serving up cool text from the Flask server!!")
12
13
14 if __name__ == "__main__":
15     app.run(debug=True)
```

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <meta http-equiv="X-UA-Compatible" content="ie=edge">
8   <title>Templates 101</title>
9   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
10 </head>
11
12 <body>
13   <div class="container">
14
15     <div class="jumbotron text-center">
16       <!-- Render our data -->
17       <h1>{{ text }}</h1>
18     </div>
19
20   </div>
21 </body>
22
23 </html>
```



Activity: Rendering A String With Flask

In this activity, you will create a webpage rendering a string with Flask.

Suggested Time:
10 Minutes



Instructions: Activity: Rendering A String With Flask

- Create a webpage that will return a welcome message with a name returned from your flask app.
- Add a paragraph underneath to display a hobby of your own; this will also be returned from the back end..
- Create a link to a bonus page that routes you to an entirely new static html page and also returns both your name and hobby from the back end.
- **Bonus:**
 - Add a link back to the home page in your bonus page.
- **Hint:**
 - Consult the Flask Render Docs for reference.



Time's Up! Let's Review.



Instructor Demonstration

Rendering a List

Rendering a List

/app.py
/templates
 /index.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <meta http-equiv="X-UA-Compatible" content="ie=edge">
8   <title>Teams!</title>
9   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
10 </head>
11
12 <body>
13   <div class="container text-center">
14     <h1 class="jumbotron">Team Rosters</h1>
15     <div>
16       <ul style="list-style: none;">
17         {% for name in list %}
18         <li>{{ name }}</li>
19         {% endfor %}
20       </ul>
21     </div>
22   </div>
23 </body>
24 </html>
```

```
1 # import necessary libraries
2 from flask import Flask, render_template
3
4 # create instance of Flask app
5 app = Flask(__name__)
6
7
8 # create route that renders index.html template
9 @app.route("/")
10 def index():
11     team_list = ["Jumpers", "Dunkers", "Dribblers", "Passers"]
12     return render_template("index.html", list=team_list)
13
14
15 if __name__ == "__main__":
16     app.run(debug=True)
```

```
{% for name in list %}
  <li>{{ name }}</li>
{% endfor %}
```



Activity: Rendering A List

In this activity, you will create a web page rendering a list with Flask.

Suggested Time:
10 Minutes



Instructions:

Activity: Rendering A List

- Create a web page that will display a list of your top five favorite movies.
- Add style to your webpage by using [bootstrap cards](#) add whatever info you like.



Time's Up! Let's Review.



Instructor Demonstration

Rendering a Dictionary

Rendering a Dictionary

/app.py
/templates
 /index.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <meta http-equiv="X-UA-Compatible" content="ie=edge">
8   <title>Sports!</title>
9   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
10 </head>
11
12 <body>
13   <div class="container text-center">
14     <h1 class="jumbotron">Player Roster</h1>
15     <div>
16       <ul style="list-style: none;">
17         <li>{{ dict.player_1 }}</li>
18         <li>{{ dict.player_2 }}</li>
19       </ul>
20     </div>
21   </div>
22 </body>
23
24 </html>
```

```
1 # import necessary libraries
2 from flask import Flask, render_template
3
4 # create instance of Flask app
5 app = Flask(__name__)
6
7
8 # create route that renders index.html template
9 @app.route("/")
10 def index():
11     player_dictionary = {"player_1": "Jessica",
12                          "player_2": "Mark"}
13     return render_template("index.html", dict=player_dictionary)
14
15
16 if __name__ == "__main__":
17     app.run(debug=True)
```

```
<ul style="list-style: none;">
  <li>{{ dict.player_1 }}</li>
  <li>{{ dict.player_2 }}</li>
</ul>
```




Activity: Rendering a Dictionary

In this activity, you will create a web page rendering a dictionary with Flask.

Suggested Time:
10 Minutes



Instructions: Activity: Rendering a Dictionary

- Create a list of dictionaries that include the name and type of animal.
- Loop through the list and display an unordered list on the webpage.
- Each line should include the name of the animal and type.
- Add some CSS styling to each list item.



Time's Up! Let's Review.



Countdown timer

40:00

(with alarm)



Instructor Demonstration

Rendering Data from MongoDB

Rendering Data from MongoDB

```
/app.py  
/templates  
  /index.html
```

- Pymongo is imported and a Flask app is created.
- A connection is set up to the Mongo client.
- Connect to a database called `team_db` if the database is not already available one will be created.
- Here, the collection is dropped to avoid the data inserting and duplicating every time the server is reset.
- The collection will be remade each time and the documents are inserted into the collection.

```
1 from flask import Flask, render_template  
2  
3 # Import our pymongo library, which lets us connect our Flask app to our Mongo database.  
4 import pymongo  
5  
6 # Create an instance of our Flask app.  
7 app = Flask(__name__)  
8  
9 # Create connection variable  
10 conn = 'mongodb://localhost:27017'  
11  
12 # Pass connection to the pymongo instance.  
13 client = pymongo.MongoClient(conn)  
14  
15 # Connect to a database. Will create one if not already available.  
16 db = client.team_db  
17  
18 # Drops collection if available to remove duplicates  
19 db.team.drop()  
20  
21 # Creates a collection in the database and inserts two documents  
22 db.team.insert_many(  
23     [  
24         {  
25             'player': 'Jessica',  
26             'position': 'Point Guard'  
27         },  
28         {  
29             'player': 'Mark',  
30             'position': 'Center'  
31         }  
32     ]  
33 )  
34  
35  
36 # Set route  
37 @app.route('/')  
38 def index():  
39     # Store the entire team collection in a list  
40     teams = list(db.team.find())  
41     print(teams)  
42  
43     # Return the template with the teams list passed in  
44     return render_template('index.html', teams=teams)  
45  
46  
47 if __name__ == "__main__":  
48     app.run(debug=True)
```



Activity: Rendering Data from MongoDB

In this activity, you will set a connection to the Mongo Client and render data from MongoDB

Suggested Time:
25 Minutes



Instructions: Activity: Rendering A String With Flask

- Create a file called `insert_data.py` and setup a connection to mongo using PyMongo.
- Next, insert at least five store items that each include, type, cost, and stock into a mongo databases and collection.
- Run the file (Why would we not want this in the app.py file?).
- Setup a Flask app that makes a connection to the database and collection you created.
- Return to a list of all the full inventory.
- Display the type of item and cost of the item on the webpage.
- **Bonus:**
 - Display cost for each item by $(\text{cost} * \text{stock})$.
- **Hints:**
 - Use bootstrap cards to clean up the look.





Time's Up! Let's Review.



Instructor Demonstration

Scrape, Save and Render Data

Scrape, Save and Render Data

/app.py
/scrape_phone.py
/templates
/index.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <meta http-equiv="X-UA-Compatible" content="ie=edge">
8   <title>Find a Phone</title>
9   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
10 </head>
11
12 <body>
13   <div class="container">
14
15     <div class="jumbotron text-center">
16       <h1>Phone Shoppin</h1>
17       <a href="#" class="btn btn-primary btn-lg" href="/scrape" role="button">Find a phone!</a>
18     </div>
19
20     <div class="row">
21       <div class="col-md-12">
22         <h2 class="heading">{{ listings.headline }}</h2>
23         <small>{{ listings.price }}</small>
24         <small>{{ listings.reviews }}</small>
25         <div>
26           Ahmed Haque [3 years, 5 months ago] • Added read all sqls
27         </div>
28       </div>
29     </div>
30 </body>
31 </html>
```

```
1 from splinter import Browser
2 from bs4 import BeautifulSoup
3
4
5 def init_browser():
6     """NOTE: Replace the path with your actual path to the chromedriver
7     executable_path = ("executable_path": "/usr/local/bin/chromedriver")
8     return Browser("chrome", **executable_path, headless=False)
9
10
11 def scrape():
12     browser = init_browser()
13     listings = []
14
15     url = "https://webscraper.io/test-sites/e-commerce/allinone/phones/touch"
16     browser.visit(url)
17
18     html = browser.html
19     soup = BeautifulSoup(html, "html.parser")
20
21     listings["headline"] = soup.find("a", class_="title").get_text()
22     listings["price"] = soup.find("h4", class_="price").get_text()
23     listings["reviews"] = soup.find("p", class_="pull-right").get_text()
24
25     return listings
```

```
app.py > ...
1 from flask import Flask, render_template, redirect
2 from flask_pymongo import PyMongo
3 import scrape_phone
4
5 app = Flask(__name__)
6
7 # Use flask_pymongo to set up mongo connection
8 app.config["MONGO_URI"] = "mongodb://localhost:27017/phone_app"
9 mongo = PyMongo(app)
10
11 # Or set inline
12 # mongo = PyMongo(app, uri="mongodb://localhost:27017/phone_app")
13
14
15 @app.route("/")
16 def index():
17     listings = mongo.db.listings.find_one()
18     return render_template("index.html", listings=listings)
19
20
21 @app.route("/scrape")
22 def scraper():
23     listings = mongo.db.listings
24     listings_data = scrape_phone.scrape()
25     listings.update({}, listings_data, upsert=True)
26     return redirect("/", code=302)
27
28
29 if __name__ == "__main__":
30     app.run(debug=True)
```



Activity: Scrape and Render

In this activity, you will scrape data into a mongo database and then use that data to build a new webpage.

Suggested Time:
35 Minutes



Instructions: Activity: Scrape and Render

- Complete the code in `scrape_costa.py` to scrape typical min and max temperatures from the [Costa Rica Vacation Page](#). The `scrape_info` function should return the typical min and max temperatures as a Python Dictionary.
- In `app.py`, complete the `/scrape` route to store the Python dictionary as a document in a mongo database collection.
- In `app.py`, complete the `/` route to read one entry from mongo and render the flask template with the mongo data.
- **Bonus:**
 - If time remains, try to scrape the image source from the Vacation page. Note that this will require building a path that consists of the website url and the relative image path.
 - Web scraping often includes data from multiple sources. Try and incorporate data from a secondary webpage into your scraper.



Time's Up! Let's Review.

*The
End*