10.5.1 Use Flask to Create a Web App

We've really come a long way in helping Robin prepare to build her web application. After familiarizing ourselves with HTML and its attributes, we've created code to scrape live data from scraping-friendly websites. Once the application is complete, we'll get the latest featured image, news article and its summary, and fact table at the push of a button.

Robin has also studied and practiced with Mongo, a NoSQL database that she'll be using to display the scraped data we've pulled. The next part is actually building the framework for the app using Flask and Mongo together.

One really great part about how we interact with Mongo through the terminal is that it works really well with Python script and Flask.

REWIND

Flask is a web microframework that helps developers build a web application. The Pythonic tools and libraries it comes with

provide the means to create anything from a small webpage or blog or something large enough for commercial use.

In your code editor, first make sure you're in your Mission-to-Mars directory, then create a new `.py` file named `app.py`. This is where we'll use Flask and Mongo to begin creating Robin's web app. Let's begin by importing our tools. In our new Python file, add the following lines of code:

```
from flask import Flask, render_template, redirect, url_for
from flask_pymongo import PyMongo
import scraping
```

Let's break down what this code is doing.

- The first line says that we'll use Flask to render a template, redirecting to another url, and creating a URL.
- The second line says we'll use PyMongo to interact with our Mongo database.
- The third line says that to use the scraping code, we will convert from Jupyter notebook to Python.

Under these lines, let's add the following to set up Flask:

```
app = Flask(__name__)
```

We also need to tell Python how to connect to Mongo using PyMongo. Next, add the following lines:

```
# Use flask_pymongo to set up mongo connection
app.config["MONGO_URI"] = "mongodb://localhost:27017/mars_app"
```

```
mongo = PyMongo(app)
```

- `app.config["MONGO_URI"]` tells Python that our app will connect to Mongo using a URI, a uniform resource identifier similar to a URL.
- `"mongodb://localhost:27017/mars_app"` is the URI we'll be using to connect our app to Mongo. This URI is saying that the app can reach Mongo through our localhost server, using port 27017, using a database named "mars_app".

Set Up App Routes

The code we create next will set up our Flask routes: one for the main HTML page everyone will view when visiting the web app, and one to actually scrape new data using the code we've written.

REWIND

Flask routes bind URLs to functions. For example, the URL "`ourpage.com/`" brings us to the homepage of our web app. The URL "`ourpage.com/scrape`" will activate our scraping code.

These routes can be embedded into our web app and accessed via links or buttons.

First, let's define the route for the HTML page. In our script, type the following:

```
@app.route("/")
def index():
   mars = mongo.db.mars.find_one()
   return render_template("index.html", mars=mars)
```

This route, `@app.route("/")`, tells Flask what to display when we're looking at the home page, index.html (index.html is the default HTML file that we'll use to display the content we've scraped). This means that when we visit our web app's HTML page, we will see the home page.

Within the `def index():` function the following is accomplished:

`mars = mongo.db.mars.find_one()` uses PyMongo to find the "mars" collection in our database, which we will create when we convert our Jupyter scraping code to Python Script. We will also assign that path to the `mars` variable for use later.

`return render_template("index.html"` tells Flask to return an HTML template using an index.html file. We'll create this file after we build the Flask routes.

`, mars=mars)` tells Python to use the "mars" collection in MongoDB.

This function is what links our visual representation of our work, our web app, to the code that powers it.

Our next function will set up our scraping route. This route will be the "button" of the web application, the one that will scrape updated data when we tell it to from the homepage of our web app. It'll be tied to a button that will run the code when it's clicked.

Let's add the next route and function to our code. In the editor, type the following:

```
@app.route("/scrape")
def scrape():
   mars = mongo.db.mars
```

```
    mars_data = scraping.scrape_all()
    mars.update_one({}, {"$set":mars_data}, upsert=True)
    return redirect('/', code=302)
```

Let's look at these six lines a little closer.

The first line, `@app.route("/scrape")` defines the route that Flask will be using. This route, `"/scrape"`, will run the function that we create just beneath it.

The next lines allow us to access the database, scrape new data using our `scraping.py` script, update the database, and return a message when successful. Let's break it down.

First, we define it with `def scrape():`.

Then, we assign a new variable that points to our Mongo database: `mars = mongo.db.mars`.

Next, we created a new variable to hold the newly scraped data: `mars_data = scraping.scrape_all()`. In this line, we're referencing the `scrape_all` function in the scraping.py file exported from Jupyter Notebook.

Now that we've gathered new data, we need to update the database using `.update_one()`. Let's take a look at the syntax we'll use, as shown below:

```
.update_one(query_parameter, {"$set": data}, options)
```

Here, we're inserting data, but not if an identical record already exists. In the query_parameter, we can specify a field (e.g. `{"news_title": "Mars Landing Successful"}`), in which case MongoDB will update a document with a matching `news_title`. Or it can be left empty (`{}`) to update the first matching document in the collection.

Next, we'll use the data we have stored in `mars_data`. The syntax used here is `{"$set": data}`. This means that the document will be modified ("$set") with the `data` in question.

Finally, the option we'll include is `upsert=True`. This indicates to Mongo to create a new document if one doesn't already exist, and new data will always be saved (even if we haven't already created a document for it).

The entire line of code looks like this: `mars.update_one({}, {"$set":mars_data}, upsert=True)`.

Finally, we will add a redirect after successfully scraping the data: `return redirect('/', code=302)`. This will navigate our page back to `/` where we can see the updated content.

The final bit of code we need for Flask is to tell it to run. Add these two lines to the bottom of your script and save your work:

```
if __name__ == "__main__":
   app.run()
```

ADD/COMMIT/PUSH

Don't forget to commit and push your work!