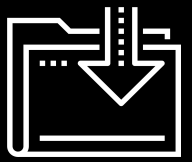
Rendering Your Data With Flask

Data Boot Camp

Lesson 12.3

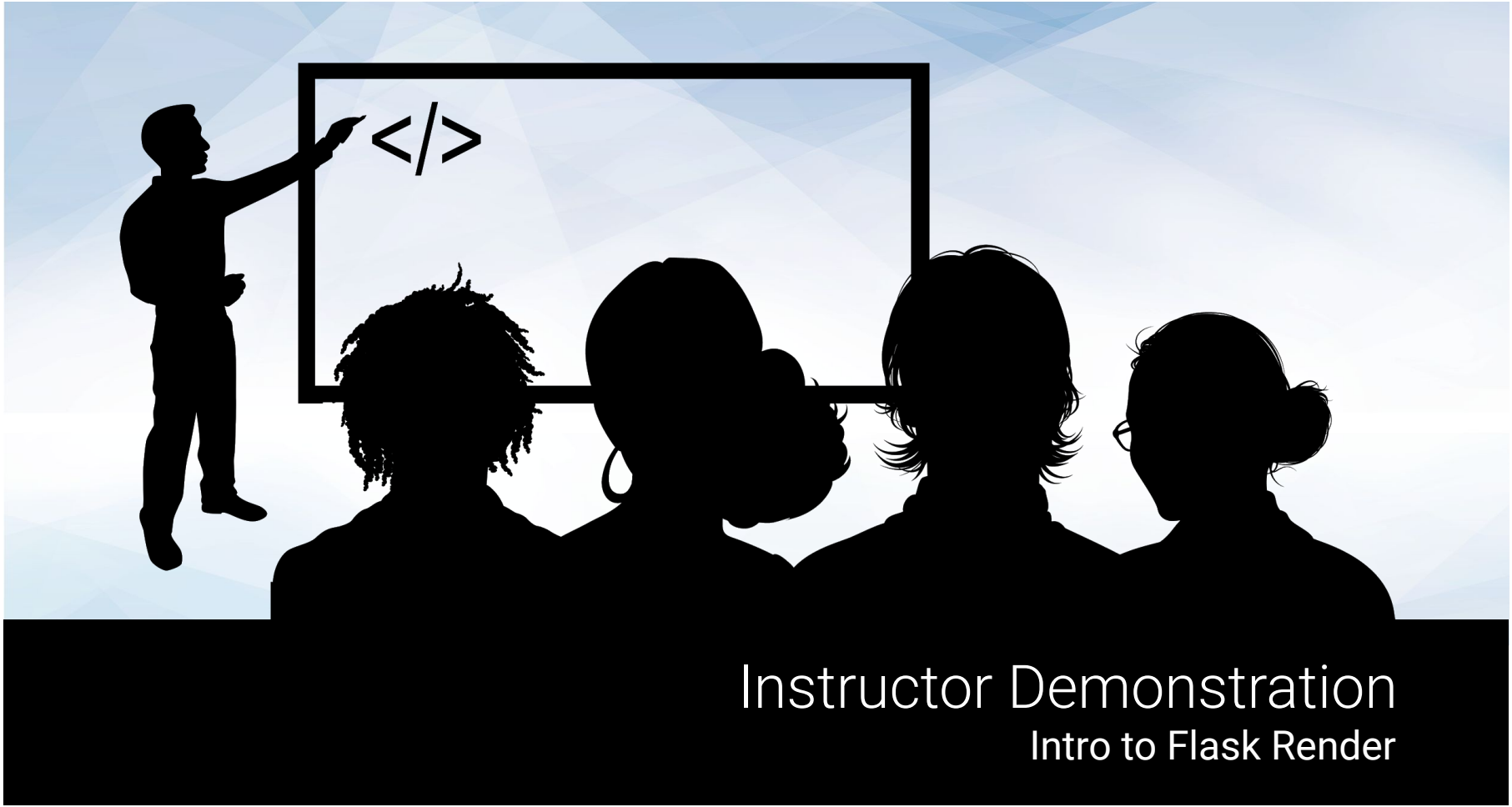



WELCOME

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.

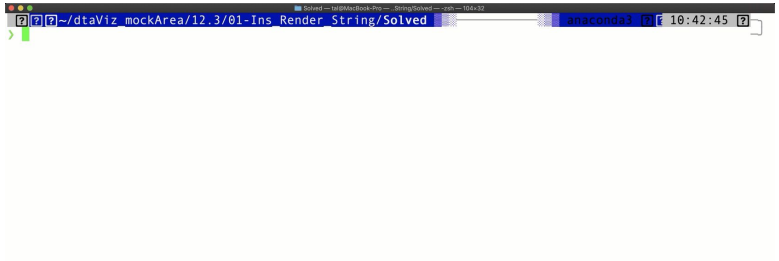


Instructor Demonstration

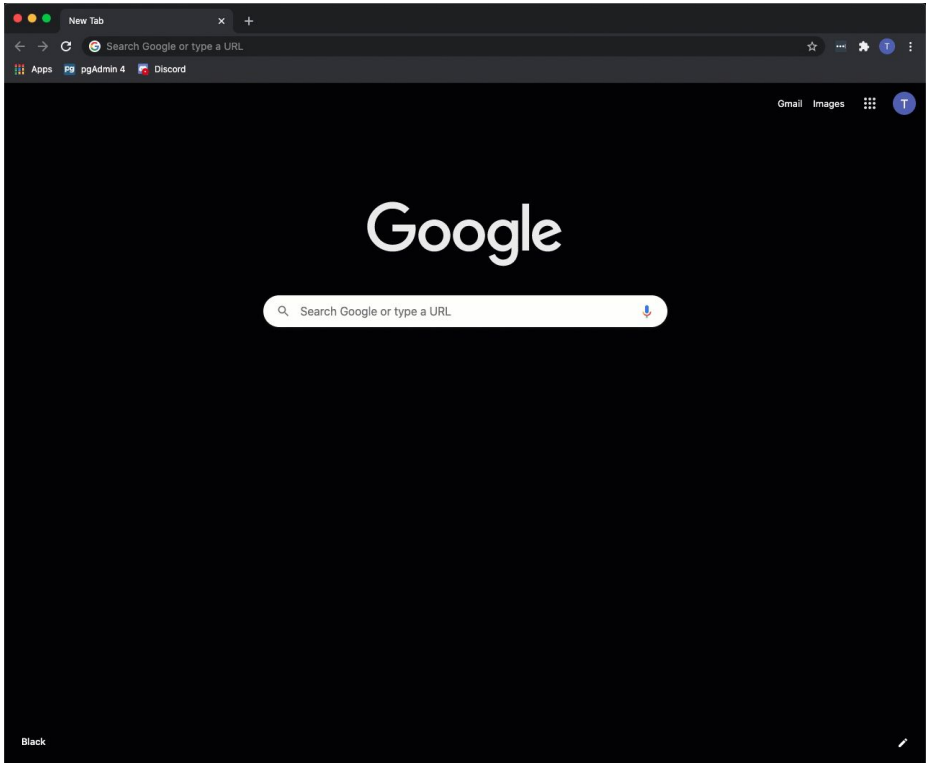
Intro to Flask Render

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

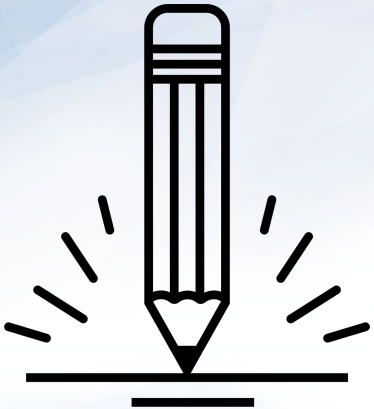


Intro to Flask Render

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

```python
# import necessary libraries
from flask import Flask, render_template

# create instance of Flask app
app = Flask(__name__)


# create route that renders index.html template
@app.route("/")
def echo():
    return render_template("index.html", text="Serving up cool text from the Flask server!!")


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

```html
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Templates 101</title>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
</head>

<body>
  <div class="container">

    <div class="jumbotron text-center">
      <!-- Render our data -->
      <h1>{{ text }}</h1>
    </div>

  </div>
</body>

</html>
```



Activity: Rendering A String With Flask

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

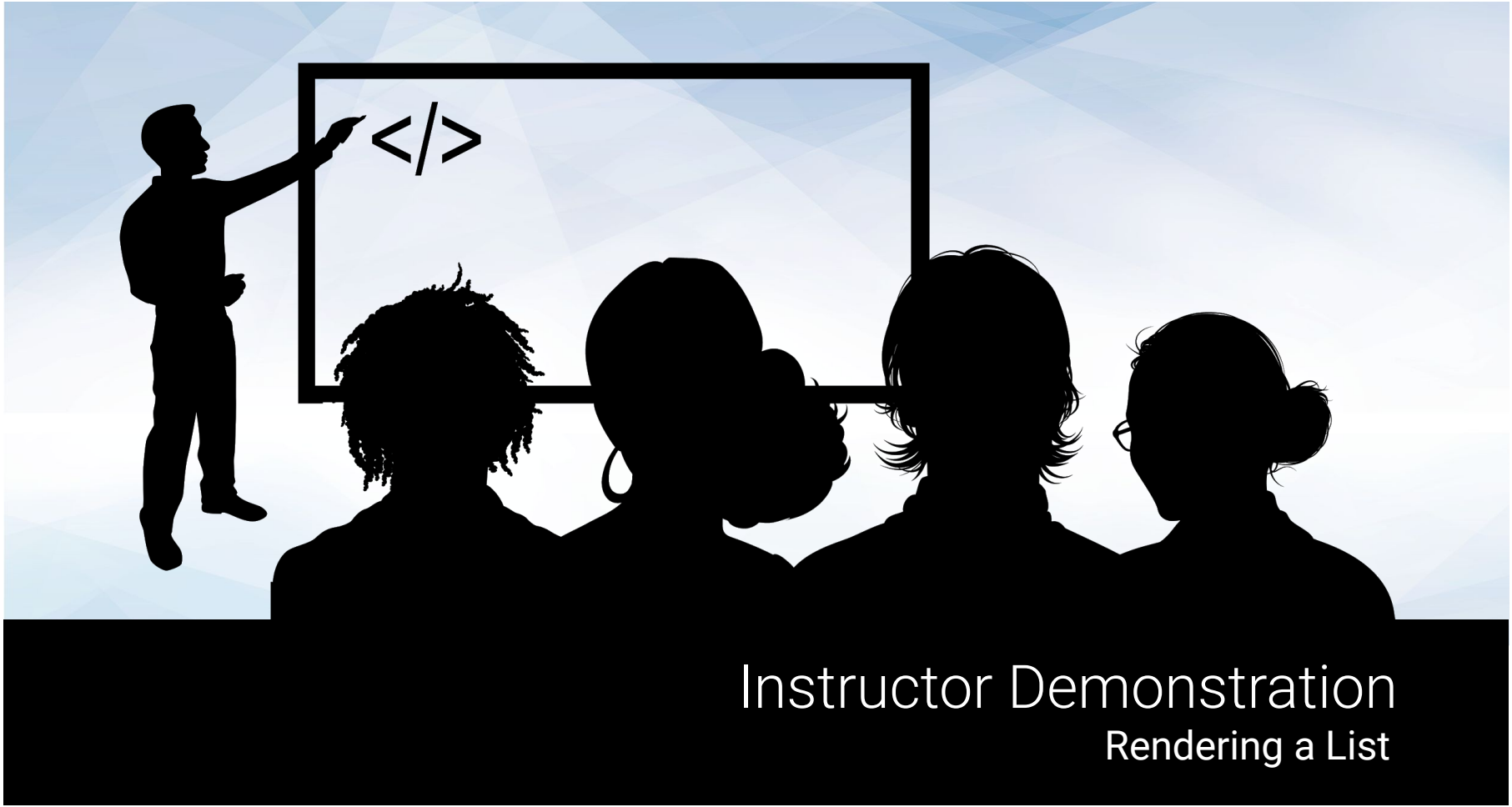
Suggested Time:
10 Minutes

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

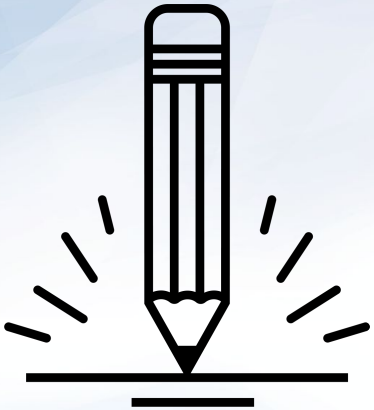
Rendering a List

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

```python
# import necessary libraries
from flask import Flask, render_template

# create instance of Flask app
app = Flask(__name__)


# create route that renders index.html template
@app.route("/")
def index():
    team_list = ["Jumpers", "Dunkers", "Dribblers", "Passers"]
    return render_template("index.html", list=team_list)


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

```html
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Teams!</title>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
</head>

<body>
  <div class="container text-center">
    <h1 class="jumbotron">Team Rosters</h1>
    <div>
      <ul style="list-style: none;">

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

      </ul>
    </div>
  </div>
</body>

</html>
```

```
{% 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.

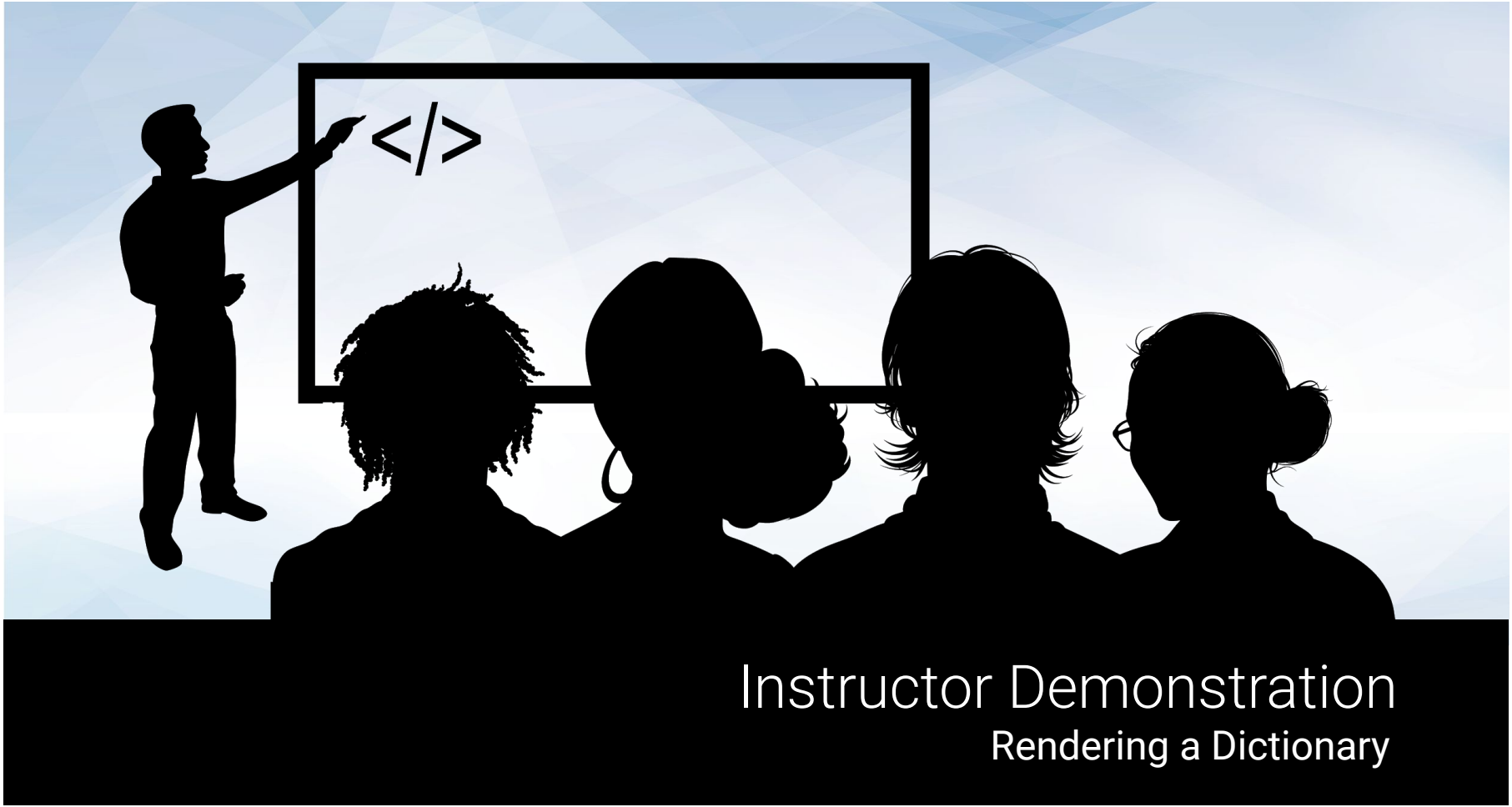
Suggested Time:
10 Minutes

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

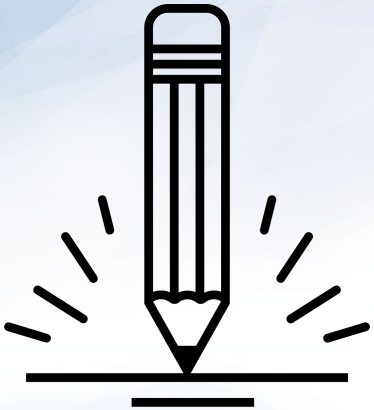
Rendering a Dictionary

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

```python
# import necessary libraries
from flask import Flask, render_template

# create instance of Flask app
app = Flask(__name__)


# create route that renders index.html template
@app.route("/")
def index():
    player_dictionary = {"player_1": "Jessica",
                         "player_2": "Mark"}
    return render_template("index.html", dict=player_dictionary)


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

```html
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Sports!</title>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
</head>

<body>
  <div class="container text-center">
    <h1 class="jumbotron">Player Roster</h1>
    <div>
      <ul style="list-style: none;">
        <li>{{ dict.player_1 }}</li>
        <li>{{ dict.player_2 }}</li>
      </ul>
    </div>
  </div>
</body>

</html>
```

```html
<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

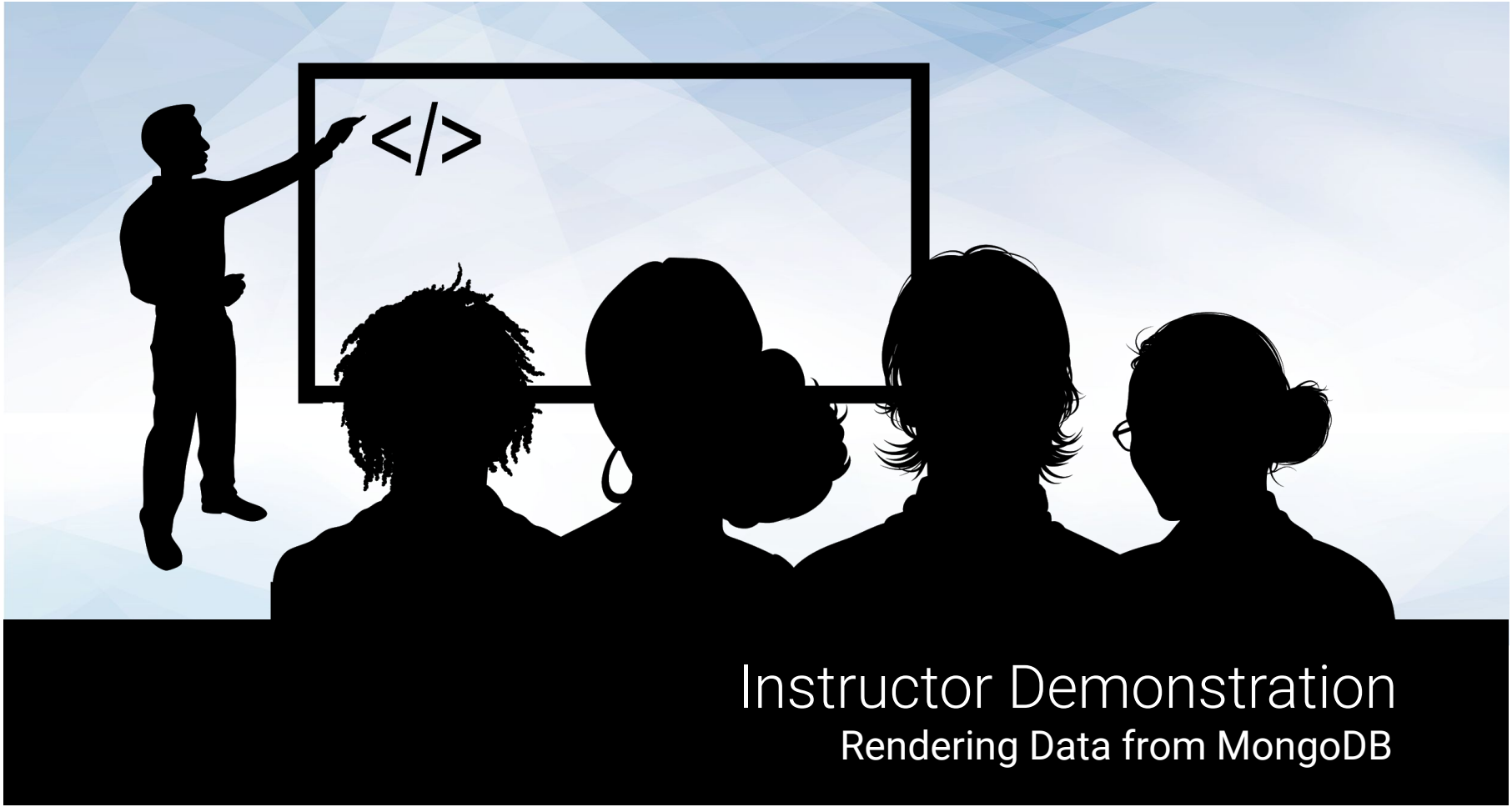
- Create a list of dictionaries that include the name and type of animal.
- Loop through the list and display an unordered list on the webpage.
- Each line should include the name of the animal and type.
- Add some CSS styling to each list item.

Time’s Up! Let’s Review.

Countdown timer

40:00

(with alarm)



Instructor Demonstration

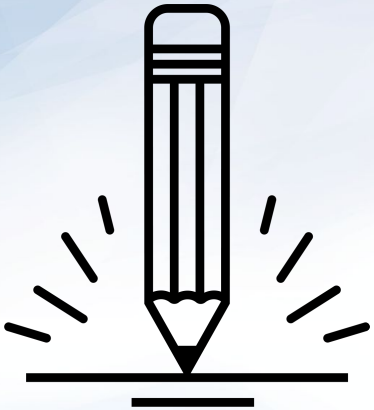
Rendering Data from MongoDB

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.

```python
from flask import Flask, render_template

# Import our pymongo library, which lets us connect our Flask app to our Mongo database.
import pymongo

# Create an instance of our Flask app.
app = Flask(__name__)

# Create connection variable
conn = 'mongodb://localhost:27017'

# Pass connection to the pymongo instance.
client = pymongo.MongoClient(conn)

# Connect to a database. Will create one if not already available.
db = client.team_db

# Drops collection if available to remove duplicates
db.team.drop()

# Creates a collection in the database and inserts two documents
db.team.insert_many(
    [
        {
            'player': 'Jessica',
            'position': 'Point Guard'
        },
        {
            'player': 'Mark',
            'position': 'Center'
        }
    ]
)


# Set route
@app.route('/')
def index():
    # Store the entire team collection in a list
    teams = list(db.team.find())
    print(teams)

    # Return the template with the teams list passed in
    return render_template('index.html', teams=teams)


if __name__ == "__main__":
    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

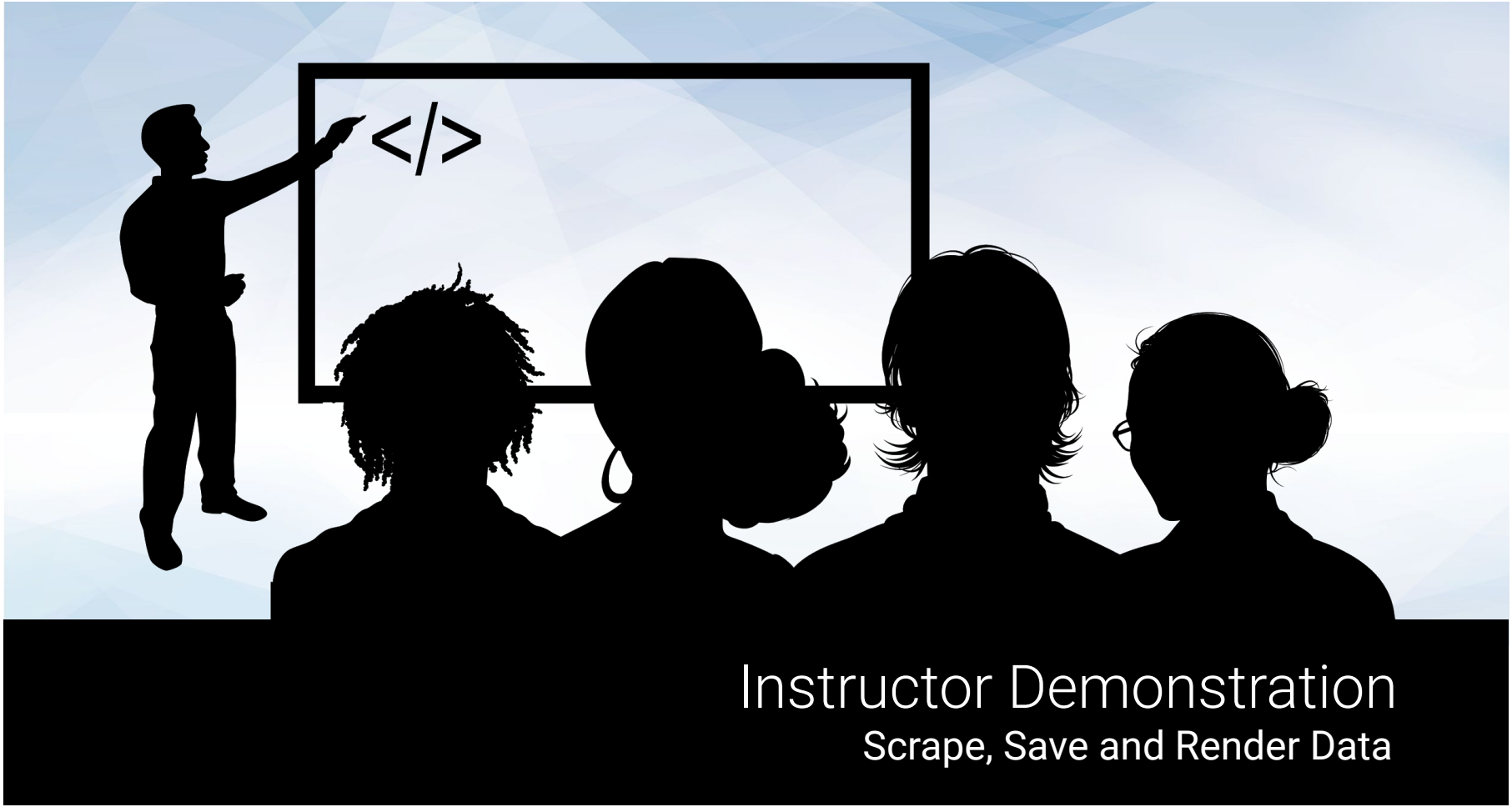
Instructions:

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 (cost * stock).
- **Hints:**
  - Use bootstrap cards to clean up the look.

Time’s Up! Let’s Review.



Instructor Demonstration

Scrape, Save and Render Data

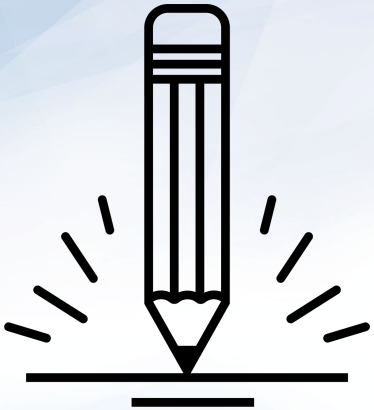
Scrape, Save and Render Data

/app.py
/scrape_craigslist.py
/templates
    /index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Hot Finds</title>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
</head>

<body>
  <div class="container">

    <div class="jumbotron text-center">
      <h1>Hot Finds On Craigslist</h1>
      <p><a class="btn btn-primary btn-lg" href="/scrape" role="button">Find An Awesome Deal!</a></p>
    </div>

    <!-- Craigslist Listings -->
    <div class="row" id="craiglist-listings">
      <div class="col-md-12">
        <h4 class="heading">{{listings.price}} {{listings.headline}}</h4>
        <small>{{listings.hood}}</small>
      </div>
    </div>

  </div>
</body>

</html>
```

```
from splinter import Browser
from bs4 import BeautifulSoup


def init_browser():
    # @NOTE: Replace the path with your actual path to the chromedriver
    executable_path = {"executable_path": "/usr/local/bin/chromedriver"}
    return Browser("chrome", **executable_path, headless=False)


def scrape():
    browser = init_browser()
    listings = {}

    url = "https://raleigh.craigslist.org/search/hhh?max_price=1500&availabilityMode=0"
    browser.visit(url)

    html = browser.html
    soup = BeautifulSoup(html, "html.parser")

    listings["headline"] = soup.find("a", class_="result-title").get_text()
    listings["price"] = soup.find("span", class_="result-price").get_text()
    listings["hood"] = soup.find("span", class_="result-hood").get_text()

    return listings
```

```
from flask import Flask, render_template, redirect
from flask_pymongo import PyMongo
import scrape_craigslist

app = Flask(__name__)

# Use flask_pymongo to set up mongo connection
app.config["MONGO_URI"] = "mongodb://localhost:27017/craigslist_app"
mongo = PyMongo(app)

# Or set inline
# mongo = PyMongo(app, uri="mongodb://localhost:27017/craigslist_app")


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


@app.route("/scrape")
def scraper():
    listings = mongo.db.listings
    listings_data = scrape_craigslist.scrape()
    listings.update({}, listings_data, upsert=True)
    return redirect("/", code=302)


if __name__ == "__main__":
    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