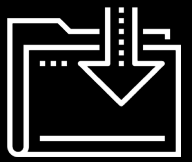
Introduction to Flask

Data Boot Camp



Class Objectives

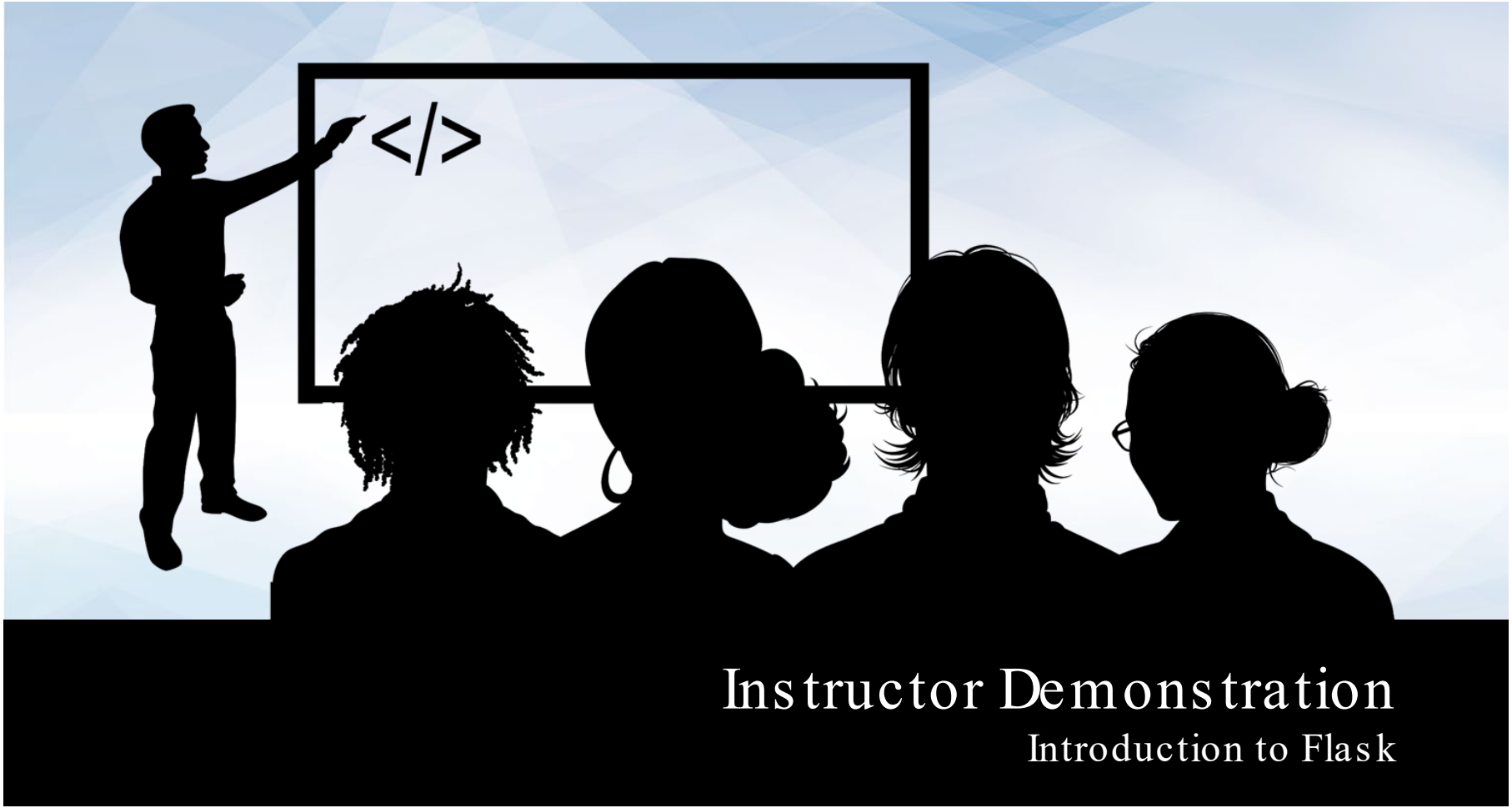
By the end of today's class you will be able to:

Create and run a Flask server.

Create static query endpoints in Flask.

Execute dynamic database queries with Flask.

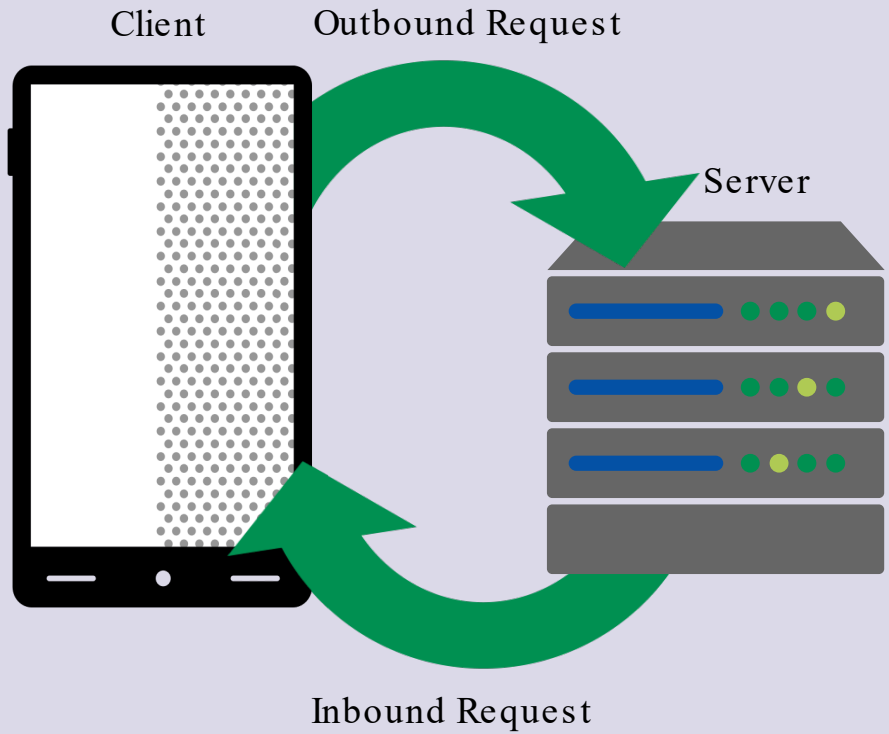
Return API query results in JSON.



Instructor Demonstration

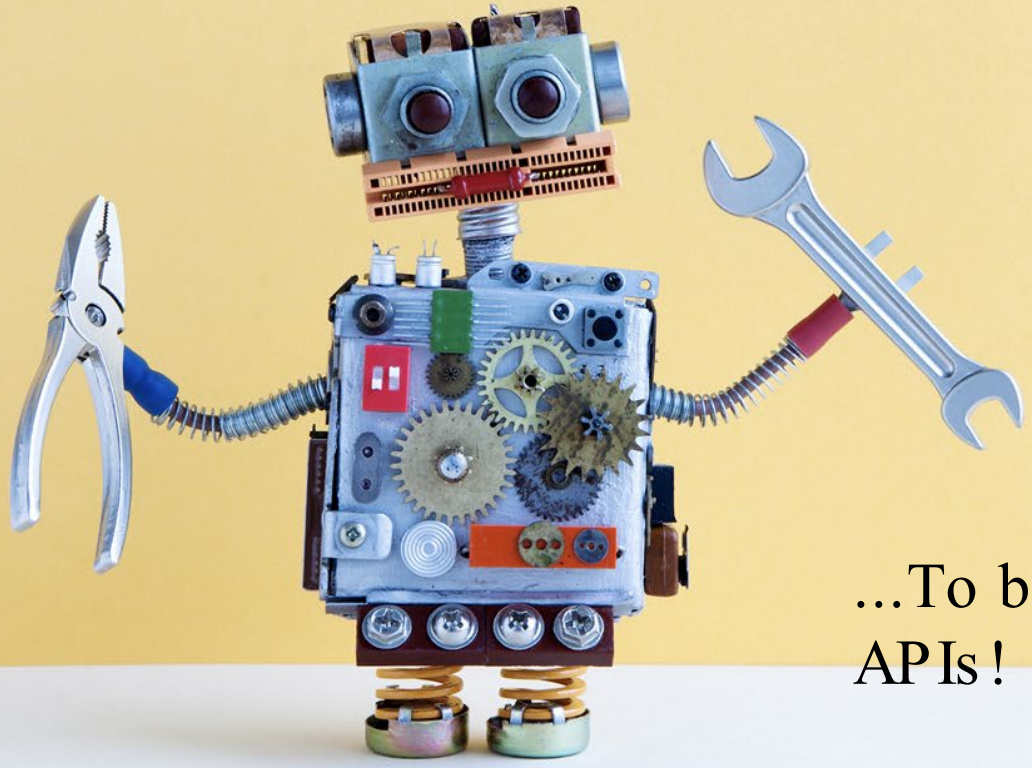
Introduction to Flask

Internet is Built from Clients and Servers

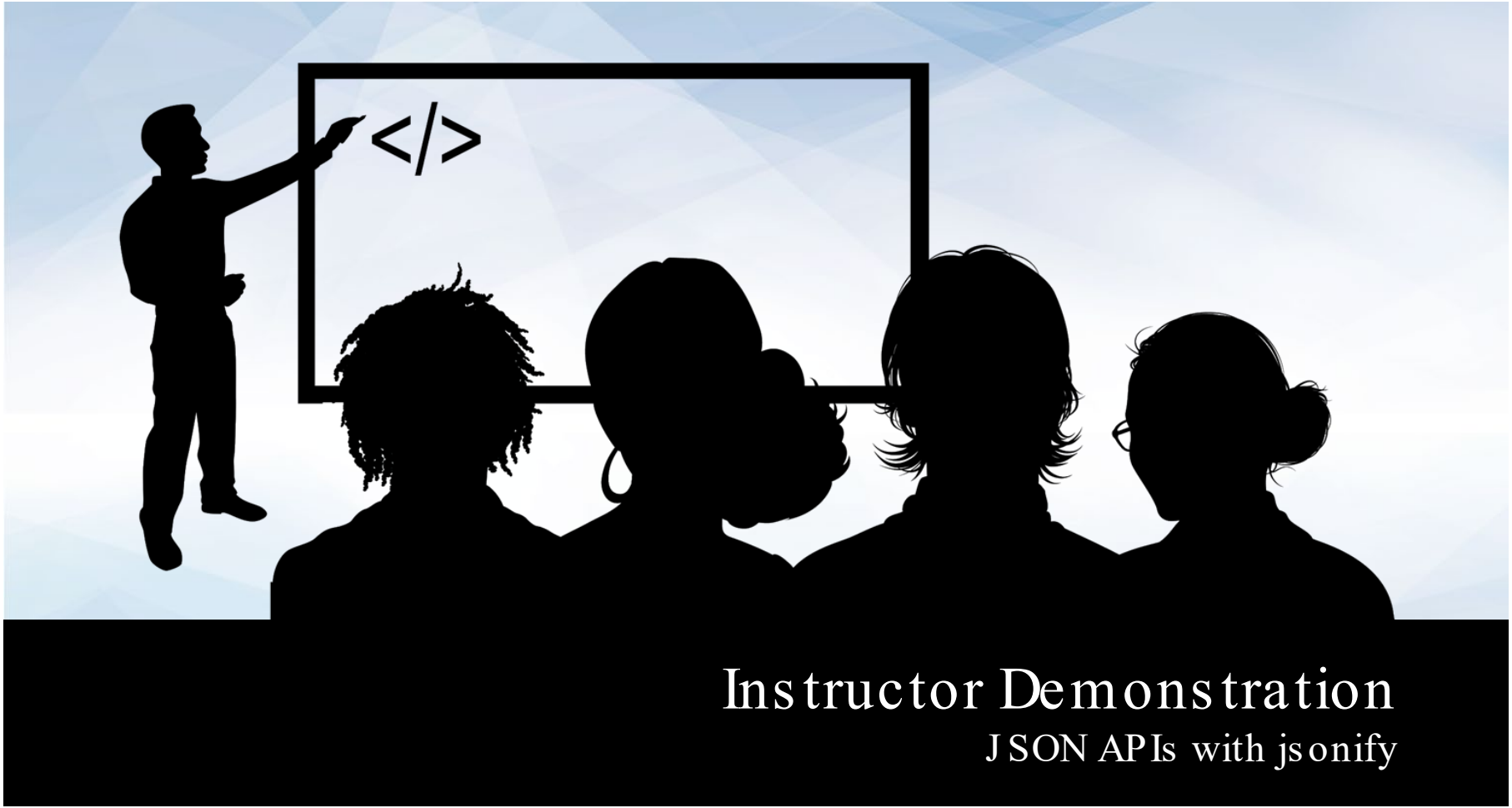


- Whatever application or device that is asking for information is called a “client”
  - A browser makes request on behalf of a user
- A “server” is a process running on a remote machine listening for requests
  - A server is essentially a *program*
- We can write the code that runs a server
  - We can determine what data is displayed
  - We can determine what data is shared

Flask is a micro web framework...



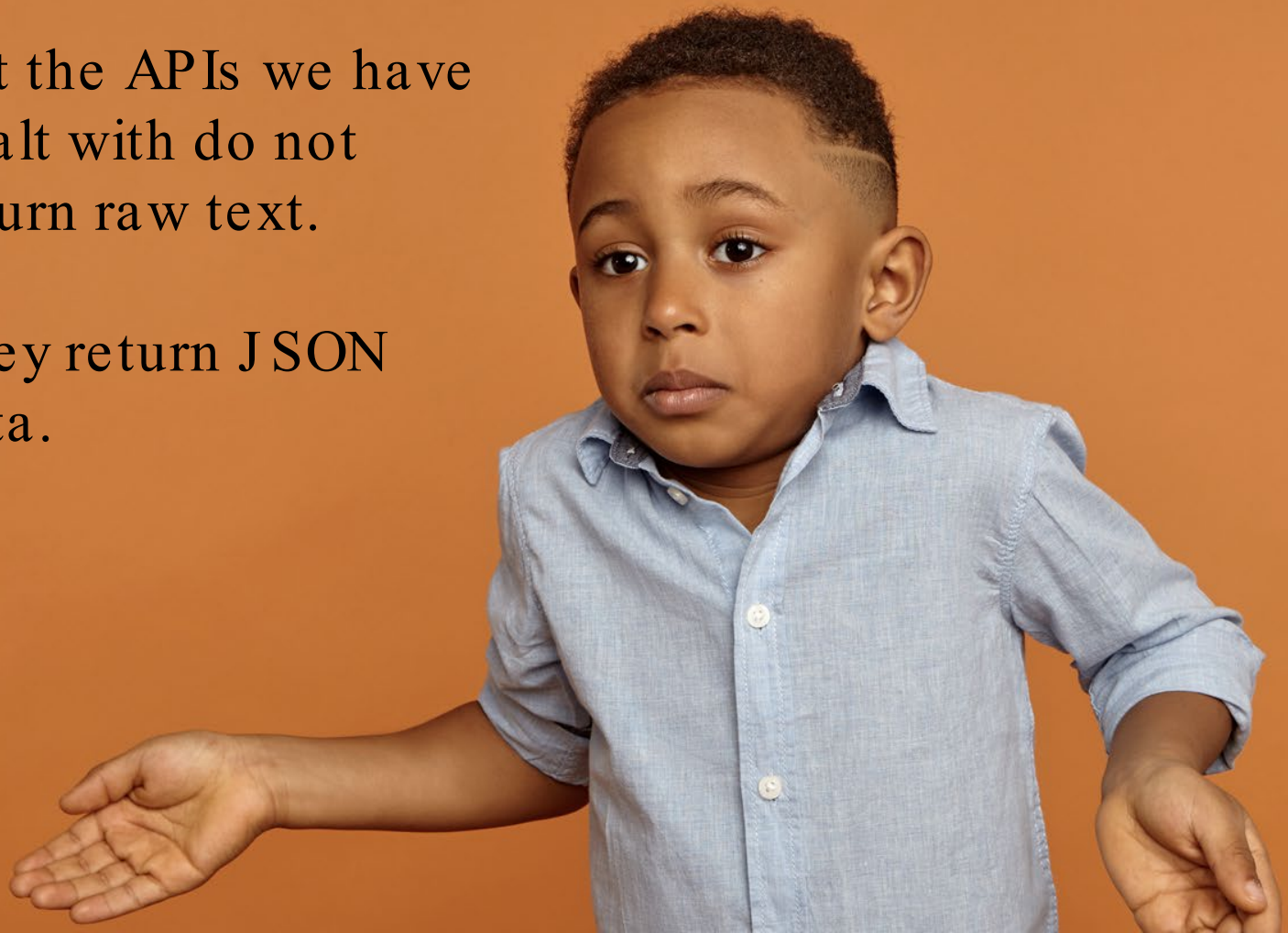
...To build **your** own APIs!


Instructor Demonstration
JSON APIs with jsonify

All routes so far have returned *string* responses.

But the APIs we have dealt with do not return raw text.

They return JSON data.



Flask has a function to create JSON responses

- We cannot simply return a dictionary response directly through Python
  - Routes must return HTTP responses
- `jsonify` automatically converts Python dictionaries into JSON responses
  - The converted JSON responses are wrapped in HTTP to send back to the client

```python
from flask import Flask, jsonify


app = Flask(__name__)

hello_dict = {"Hello": "World!"}


@app.route("/")
def home():
    return "Hi"


@app.route("/normal")
def normal():
    return hello_dict


@app.route("/jsonified")
def jsonified():
    return jsonify(hello_dict)
```

Activity: Justice League

In this activity, you will create a server that sends welcome text at one endpoint, and JSON data at another endpoint.

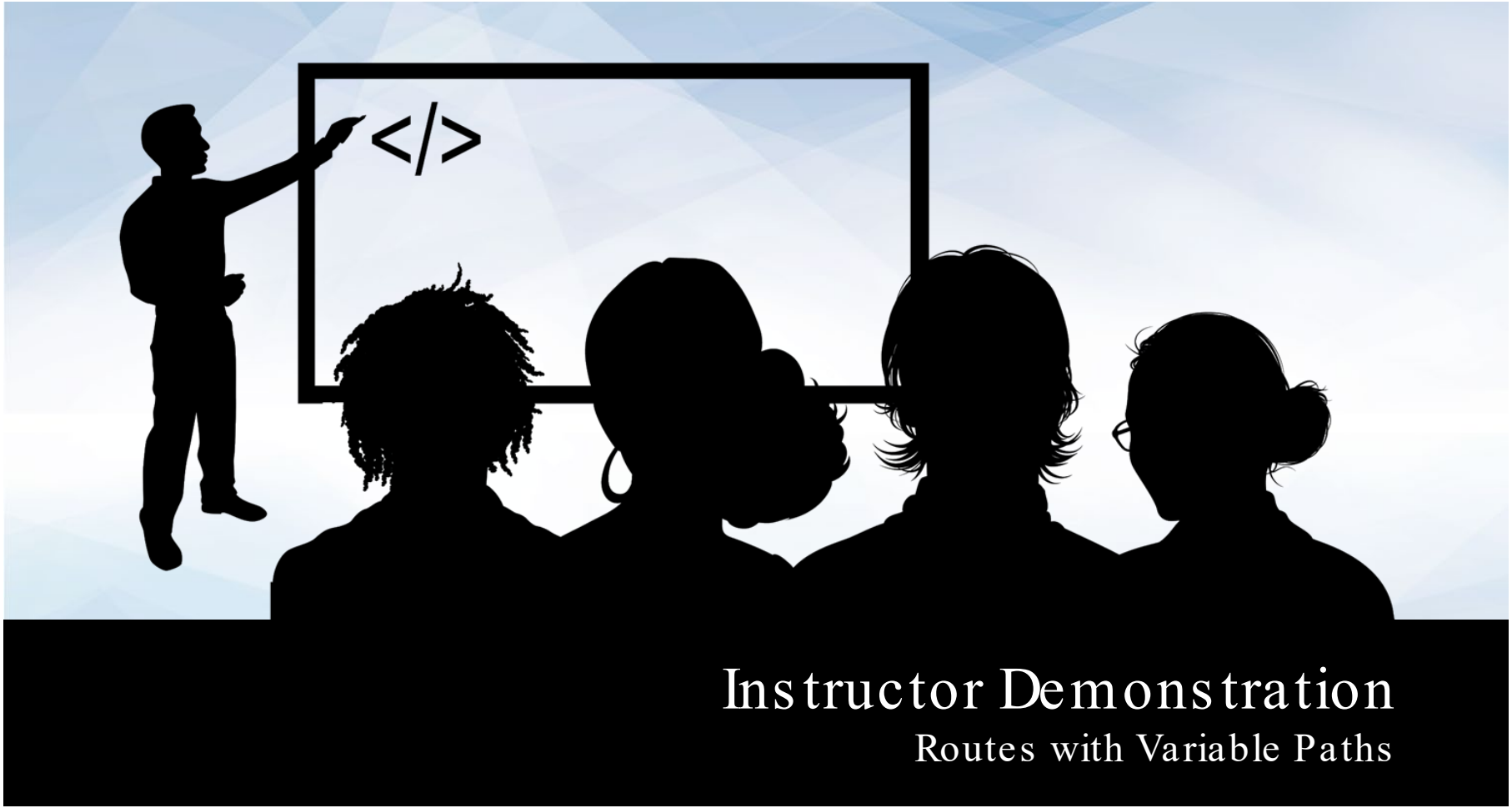
(Instructions sent via Slack.)

Suggested Time:
20 Minutes

Justice League Instructions

- Create a file called app.py for your Flask app.
- Define a Python dictionary containing the superhero name and real name for each member of the DC Comics Justice League
- Create a **GET** route called /api/v1.0/justice-league.
- Define a root route / that will return the usage statement for your API.

Time's Up! Let's Review.

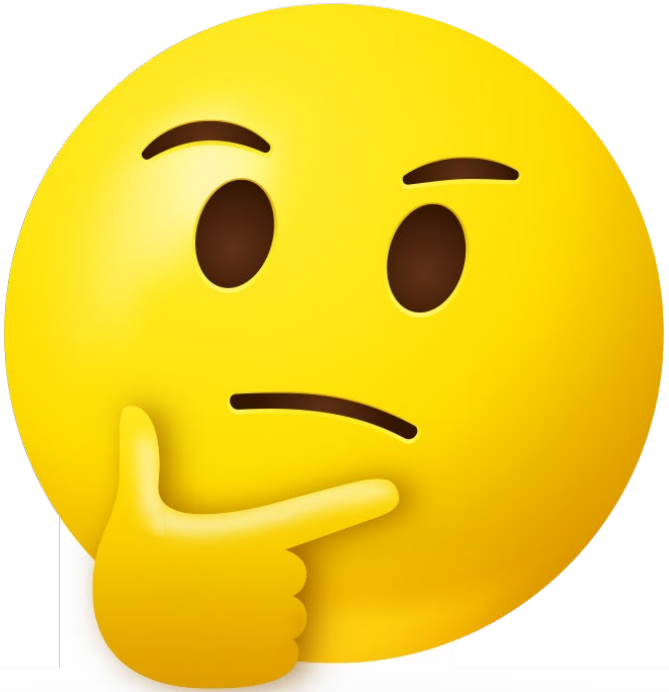


Instructor Demonstration

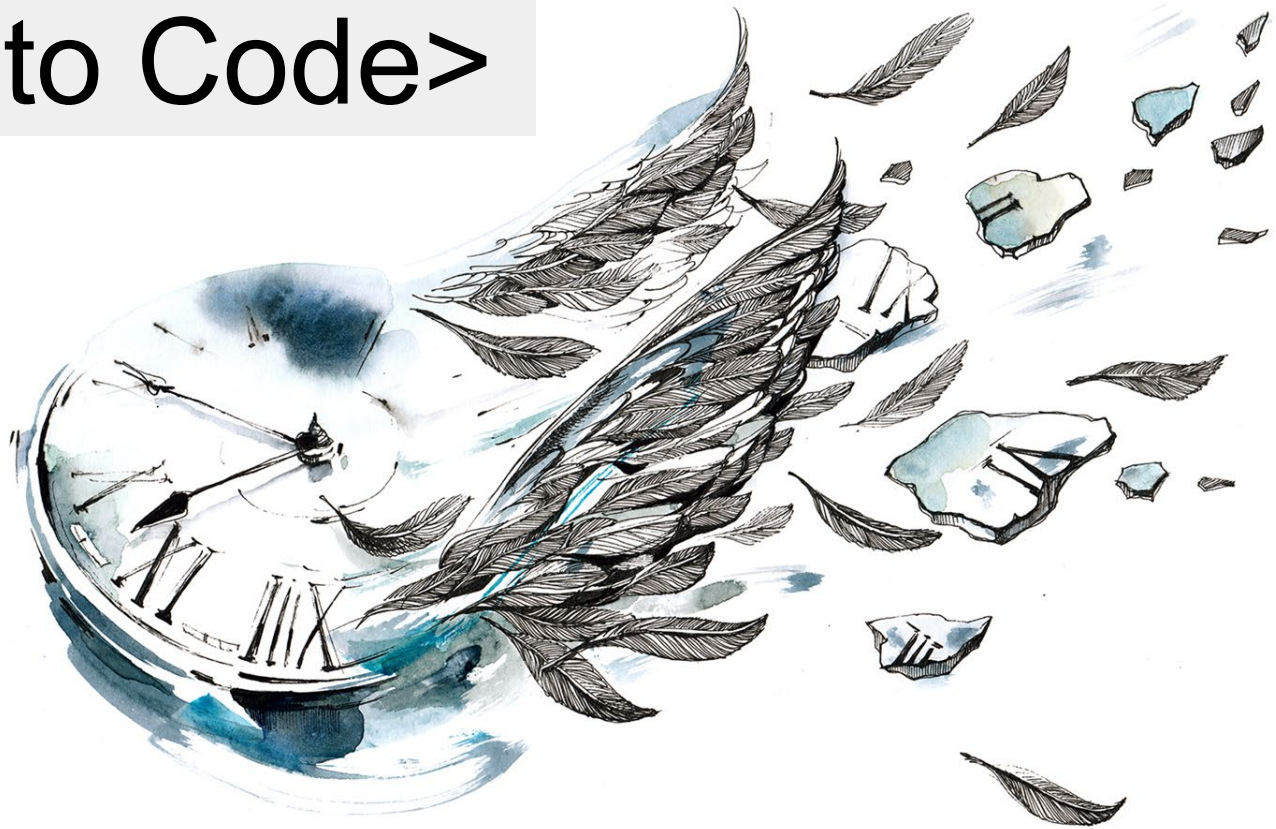
Routes with Variable Paths

Our current API is one-dimensional

- Our current API can only return the *entire* Justice League dataset
- Ideally clients can send a request for a character and expect
  - A JSON response with only specific character information
  - A detailed error response



<Time to Code>



Activity: Routes with Variable Rules

In this activity, you will add an additional API route that returns a JSON containing an individual superheroes information.

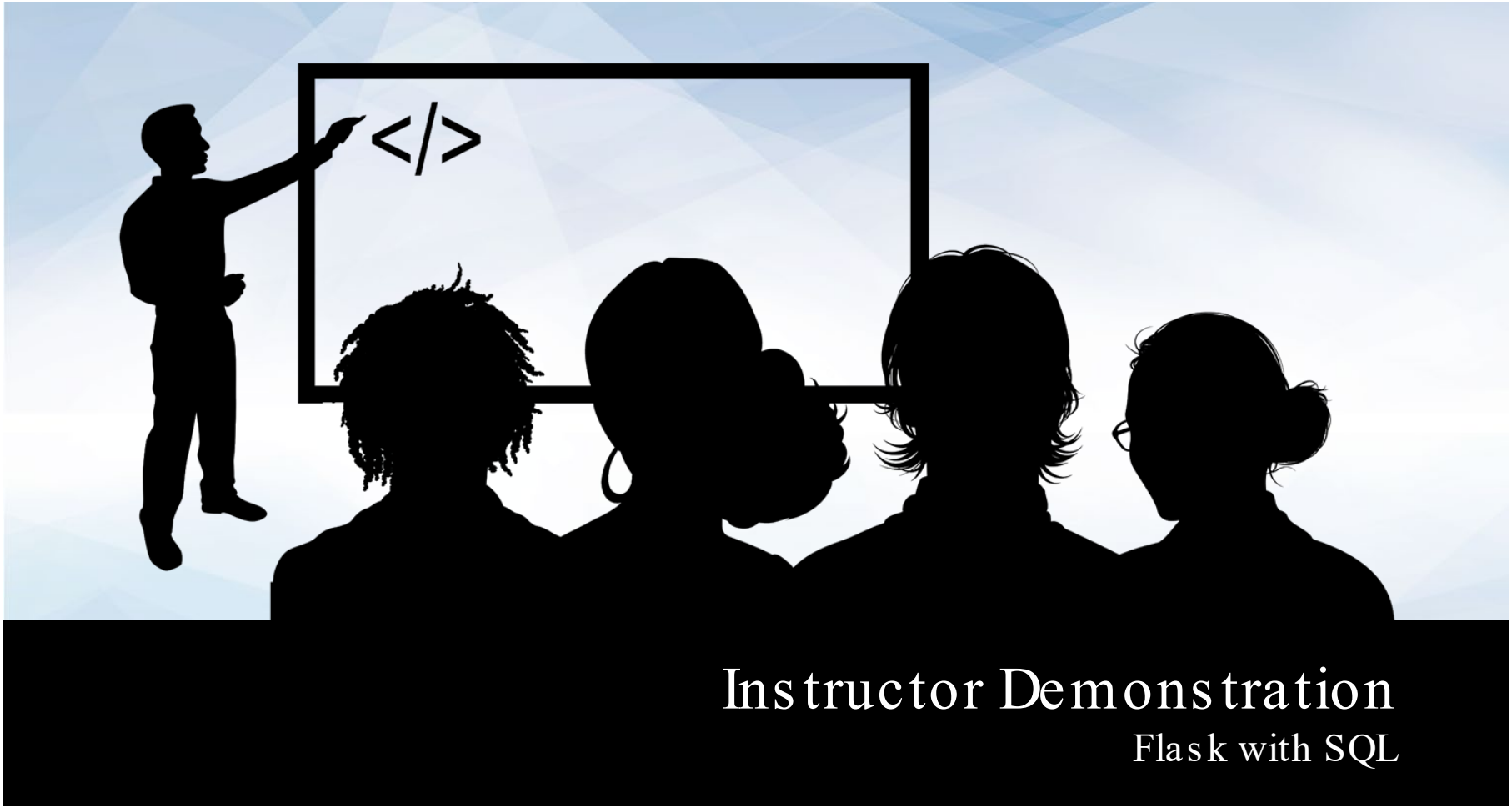
(Instructions sent via Slack.)

Suggested Time:
20 Minutes

Routes with Variable Rules

- Using the last activity as a starting point, add code to allow for getting a specific hero's information based on their superhero name.

Time's Up! Let's Review.



Instructor Demonstration

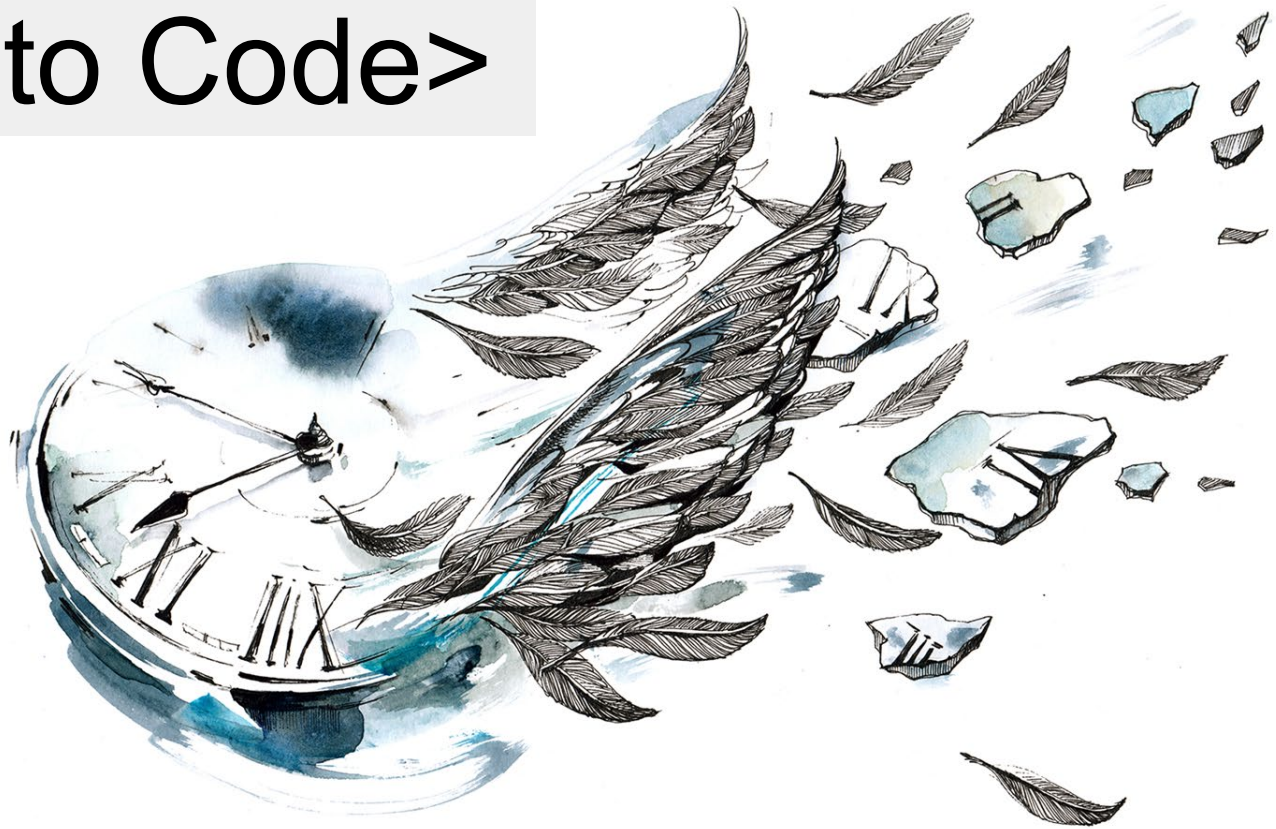
Flask with SQL

It is time to put all of the pieces together!

Flask and SQLAlchemy

- A useful API will enable the client to make requests and queries on *massive* datasets
  - Potentially too large to load into memory
- SQLAlchemy can be used to perform queries based on a flask route
- Convert the query into a dictionary, then into a JSON with jsonify
- Return the JSON query to the endpoint

<Time to Code>



Activity: Chinook Database Analysis

In this activity, you will practice analyzing databases using the SQLAlchemy ORM.

(Instructions sent via Slack.)

Suggested Time:
20 Minutes

Chinook Database Analysis Instructions

- Create a SQLAlchemy engine to the database chinook.sqlite.
- Design a query that lists all of the billing countries found in the invoices table.
- Design a query that lists the invoices totals for each billing country and sort the output in descending order.
- Design a query that lists all of the Billing Postal Codes for the USA.
- Calculate the invoice items totals sum(UnitPrice * Quantity) for each Billing Postal Code for the USA.

Time's Up! Let's Review.