APIs and Requests in Flask

Learning Objectives

After this lesson, you will be able to:

- Create an API that makes a `GET` request with Flask.
- Create an API that makes a `POST` request with Flask.

Discussion: Remember APIs?

- We can call them.
- But who publishes them?
- Do you think we could make one?

APIs

In your browser, head to https://swapi.co/api/people/13/?format=json.

- That’s a collection of data about Chewbacca.

What would it look like in Chewbacca’s language?

Head to https://swapi.co/api/people/13/?format=wookiee.

- This is the same data written in Wookiee!

Web API Recap

- A list of function calls that are made to remote servers.
    - Sent by encoding a URL (an HTTP request).
    - We could **call** the OMDb API to get a movie's information.
- Now, we're going to **create** an API using Flask.

Discussion: The Sides of an API

What’s the difference between calling and creating an API?

HTTP

- Stands for Hypertext Transfer Protocol.
- A system of rules (protocol) that determines how webpages (hypertext) get sent from one place to another (transfer).

Recap: Clients and Servers

With HTTP, there are two sides:

- Clients
  - Make the requests.
- Servers
  - Receive those requests.

CRUD

These four functions are everywhere in programming:

- **C**reate
- **R**ead
- **U**pdate
- **D**elete

CRUD Mapped to HTTP Requests

What do we do when calling the OMDb API?

- `GET`:
  - *Read*.
  - “Tell me all values in `instrument_list`.”
- `POST`:
  - Usually *Create*, sometimes *Update*.
  - “Add `cello` to `instrument_list`.”
- `PUT`:
  - Similar to `POST`.
  - *Create* or *Update* an entity.
- `PATCH`:
  - *Update* only a specified field.
  - “In `instrument_list`, change `guitar_type` to `bass`.”
- `DELETE`:
  - *Delete!*
  - “Delete `instrument_list`.”
  - Doesn’t necessarily happen immediately.

Knowledge Check:

What does CRUD stand for?

Knowledge Check: `POST` and `GET`

What's the difference between a `POST` and `GET` request?

Creating an API With Flask

We’re going to create an example of an API that:

- Takes in a list of dictionaries.
- Parses that list based on what we pass into the API.
- Returns a JSON with the appropriate data.

Remember JSON?

- Both dictionaries and JSONs have key-value pairs.
- Both dictionaries and JSONs are wrapped in curly brackets (`{}`).

```
heroes_dictionary = {'person': 'Peter_Norvig', 'person': 'Gilbert_Strang', '
heroes_json = [{'person': 'Peter_Norvig'}, {'person': 'Gilbert_Strang'}, {'p
```

We Do: New Functions

- Import two new functions: `jsonify` and `request`.

```
from flask import Flask, jsonify, request
```

We Do: First API Route

- Add a new route under our `hello` home page.

```
@app.route('/api', methods=['GET'])
def returnJsonTest():
    return jsonify({'What happened?': 'It worked!'})
```

- Test both routes:
  - `localhost:5000`
  - `localhost:5000/api`

Knowledge Check: Discussion

What two new functions did we add into our import?

What do they do?

We Do: Altering Data With APIs

- Cool!
- What if we want the data to change?
- Add a list under the `app` instantiation, above the routes.

```
heroes = [{'person': 'Peter_Norvig'}, {'person': 'Gilbert_Strang'}, {'person':
```

What can we do with that?

We Do: APIs to Return All Data

- We have a list.
- We need to get data from it.
- Make a new route:

```
@app.route('/heroes', methods=['GET'])

def gimmeAllHeroes():

    return jsonify({'heroes': heroes})
```

We Do: APIs to Return Only SOME Data

- At this route, loop over the heroes.
- Try to find the one we want!

```python
@app.route('/heroes/<string:name>', methods=['GET'])
def gimmeOneHero(name):
        names = [hero for hero in heroes if hero['person'] == name]
    return jsonify({'hero': names[0]})
```

We Do Aside — Always Error-Check

What happens when you input something that's inaccurate?

This is a good time for error-checking!

```
def gimmeOneHero(name):
    names = [hero for hero in heroes if hero['person'] == name]
    if names:
        return jsonify({'hero': names[0]})
    else:
        return "Hero not found"
```

Create a POST Request With Flask

- What if we want more heroes?
- Let’s add data to our list of heroes with a `POST` request.
    - `POST` was “Create” (and, very rarely, “Update”).

Adding Our New `POST` Function

- We can use the same route — with a different method.

```
@app.route('/heroes', methods=['POST'])
def addMyHero():
    newhero = {"person": request.get_json()["person"]}


    heroes.append(newhero)
    return jsonify({"heroes": heroes})
```

Knowledge Check

Assuming our code doesn’t have any errors, what should happen when our `POST` request takes place?

Profit

Now we'll check to see if our `POST` request works.

- Open a new terminal window, and `python hello_api.py`.
  - Launch the app!
- Going to `/heroes` gives us the heroes list.

- How do we `POST`?

- We'll use `curl`:
  - A command line tool for getting or sending files with URL syntax.
  - Not necessary to memorize!

Trying Out `POST` and `cURL`

- With the app running, open a new tab in the command prompt.
- Replace the name, then copy this right over:

```
curl -X POST -H "Content-Type: application/json" -d '{"person":"<<INSERT A N
```

- Check the command line output!
- Try going to `http://localhost:5000/heroes` — your hero is listed!

Quiz

Which of these is the right code for a POST request?

- Option A

```
@app.route('/myapiroute', methods=['POST'])
def butAmIMakingARequest():
    type_of_request = {"requestType:" :" This is definitely a GET Request"}
    requestage.append(type_of_request)
    return jsonify({"theAnswer" :  requestage})
```

- Option B

```
type_of_request = [{"requestType:" :" This is definitely a POST Request"}]
@app.route('/myapiroute', methods=['GET'])
def butAmIMakingARequest():
    return jsonify({"theAnswer" :  type_of_request})
```

Summary

We covered APIs and requests in Flask:

- We made an API using JSON!
- We used `GET` to display it.
- We used `POST` to add to it.

Additional Reading

- Flask JSONify Documentation