**BUILD YOUR FIRST FLASK APP**

**Introduction**

[Flask](https://flask.palletsprojects.com/en/1.1.x/) is a popular Python framework for developing web applications. Classified as a *microframework*, it comes with minimal built-in components and requirements, making it easy to get started and flexible to use. At the same time, Flask is by no means limited in its ability to produce a fully featured app. Rather, it is designed to be easily extensible, and the developer has the liberty to choose which tools and libraries they want to utilize. As such, Flask is capable of creating both simple static websites as well as more complex apps that involve database integration, accounts and authentication, and more!

In this lesson, we’ll start by looking at an example of a minimal Flask application. It will display the text, Hello, World! on the webpage. You’ll learn how to create this and build on top of it in the following exercises.

Let’s get started!

**Instructions**

Click Run to start the app. Feel free to take a look at the code in **app.py** and move on when you’re ready!

Notice that the app is being run in [**http://localhost:5000/**](http://localhost:5000/) on the embedded browser. For now, this app is just running locally and can only be accessed there.

**app.py**

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def home():

    return 'Hello, World!'

****

**Instantiate Flask Class**

We’ll now break down each step in creating a minimal Flask app. The Python module that contains all the classes and functions needed for building a Flask app is called flask.

We can begin building our app by importing the Flask class, which is needed to create the main application object, from the flask module:

from flask import Flask

Now, we can create an instance of the Flask class. Let’s save the application object in a variable called app:

app = Flask(\_\_name\_\_)

When creating a Flask object, we need to pass in the name of the application. In this case, because we are working with a single module, we can use the special Python variable, \_\_name\_\_.

The value of \_\_name\_\_ depends on how the Python script is executed. If we run a Python script directly, such as with python app.py in the terminal, then \_\_name\_\_ is equal to the string '\_\_main\_\_'. On the other hand, if the script is being imported as a module into another Python script, then \_\_name\_\_ would be equal to its filename.

As we’ll see in the next exercise, this distinction can be useful when we have code that we want to be run only if the script is executed a particular way.

**Instructions**

**1.**

Import the Flask class from the flask module at the top of **app.py**.

Checkpoint 2 Passed

Hint

The syntax for importing is:

from module\_name import ClassName

**2.**

Create an instance of the Flask class, passing in \_\_name\_\_, and save the object to a variable called app.

Checkpoint 3 Passed

Hint

The syntax for instantiating a class is:

object\_name = ClassName(...)

**3.**

At the bottom of the script, try printing \_\_name\_\_. Then, run **app.py**.

What is the value of \_\_name\_\_?

Checkpoint 4 Passed

Hint

You should see '\_\_main\_\_' being printed as the value of \_\_name\_\_ because you’ve run the script **app.py** directly.

**app.py**

from flask import Flask

app = Flask(\_\_name\_\_)

print(\_\_name\_\_)

**Routing**

Each time we visit a URL in a browser, it makes a request to the web server, which processes the request and returns a response back to the browser. In our Flask app, we can create *endpoints* to handle the various requests. Requests from different URLs can be directed to different endpoints in a process called *routing*.

To build a route, we need to first define a function, known as a *view function*, that contains the code for processing the request and generating a response. The response could be something as simple as a string of text. Then, we can use the route() decorator to bind a URL to the view function such that the function will be triggered when the URL is visited:

@app.route('/')  
def home():  
    return 'Hello, World!'

The route() decorator takes the URL path as parameter, or the part of the URL that follows the domain name. All URL paths must start with a leading slash. In the above example, if we visit <http://localhost:5000/> in the browser, Hello, World! will be displayed on the webpage.

Multiple URLs can also be bound to the same view function:

@app.route('/')  
@app.route('/home')  
def home():  
    return 'Hello, World!'

Now, both <http://localhost:5000/> and <http://localhost:5000/home> will display Hello, World!.

**Instructions**

**1.**

Define a function called home() that returns 'Hello, World!'

Checkpoint 2 Passed

Hint

The syntax for defining a function is:

def function\_name():  
    # code goes here

**2.**

Use the route() decorator to bind the URL path '/' to the view function.

Run **app.py** and view your page at [http://localhost:5000/](http://localhost:5000/" \t "_blank) in the browser. What do you see?

Checkpoint 3 Passed

Hint

You should see Hello, World! displayed when you visit [http://localhost:5000/](http://localhost:5000/" \t "_blank).

**3.**

Bind a second URL path '/home' to the home() function.

Navigate to [http://localhost:5000/home](http://localhost:5000/home" \t "_blank) in the browser.

Checkpoint 4 Passed

Hint

You should see Hello, World! displayed when you visit [http://127.0.0.1:5000/home](http://127.0.0.1:5000/home" \t "_blank).

**4.**

Let’s create another route! Define a function called reporter() that returns 'Reporter Bio' and is bound to the path '/reporter.

Navigate to [http://localhost:5000/reporter](http://localhost:5000/reporter" \t "_blank) in the browser.

Checkpoint 5 Passed

Hint

You should see Reporter Bio displayed when you visit [http://localhost:5000/reporter](http://localhost:5000/reporter" \t "_blank).

**app.py**

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

@app.route('/home')

def home():

  return 'Hello, World!'

@app.route('/reporter')

def reporter():

  return 'Reporter Bio'





