

# BUILD YOUR FIRST FLASK APP

## Introduction

[Flask](#) 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/> 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!'
```

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**.

---

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`.

---

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__`?

---

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!'`

---

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/> in the browser. What do you see?

---

Hint

You should see `Hello, World!` displayed when you visit <http://localhost:5000/>.

### 3.

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

Navigate to <http://localhost:5000/home> in the browser.

---

Hint

You should see `Hello, World!` displayed when you visit <http://127.0.0.1:5000/home>.

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> in the browser.

---

Hint

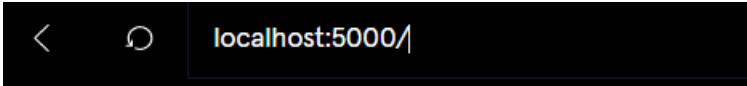
You should see `Reporter Bio` displayed when you visit <http://localhost:5000/reporter>.

**app.py**

```
from flask import Flask
app = Flask(__name__)

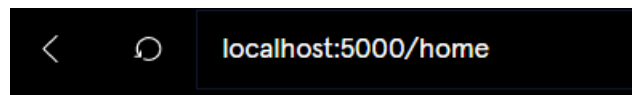
@app.route('/')
@app.route('/home')
def home():
    return 'Hello, World!'

@app.route('/reporter')
def reporter():
    return 'Reporter Bio'
```

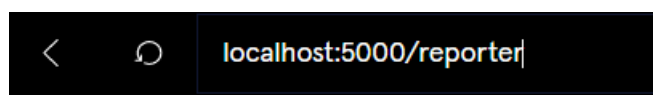


< ↻ localhost:5000/

Hello, World!



Hello, World!



Reporter Bio



## Render HTML

The response we return from a view function is not limited to plain text or data. It can also return HTML to be rendered on a webpage:

```
@app.route('/')
def home():
    return '<h1>Hello, World!</h1>'
```

We can use triple quotes to contain multi-line code:

```
@app.route('/')
@app.route('/home')
def home():
    return '''
    <h1>Hello, World!</h1>
    <p>My first paragraph.</p>
    <a href="https://www.codecademy.com">CODECADEMY</a>
    '''
```

## Instructions

### 1.

Update your `home()` function to return `Hello, World!` as a `<h1>` heading.

Run **app.py** and view your page at <http://localhost:5000/> in the browser.

---

Hint

You can return HTML as a string from the view function.

### 2.

Now, update `reporter()` to return `Reporter Bio` as a `<h2>` heading.

Navigate to <http://localhost:5000/reporter> in the browser.

---

Hint

You can return HTML as a string from the view function.

### 3.

Let's add another tag to the HTML in `reporter()`! After the `<h2>` heading, add an `<a>` tag with the text `Return to home page` that links back to <http://localhost:5000/>. To have the code run properly in the learning environment, use just the relative path `"/` as the `href` attribute.

Run the app and try clicking on the link you created!

---

Hint

The syntax for creating an `<a>` element is:

```
<a href="/path/to/page">Link Text</a>
```

**app.py**

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
@app.route('/home')
def home():
    return '<h1>Hello, World!</h1>'

@app.route('/reporter')
def reporter():
    return '''
        <h2>Reporter Bio</h2>
        <a href="/">Return to home page</a>
    '''
```

---

## Variable Rules

We've seen how the `route()` decorator can be used to bind one or more static URLs to a view function. But what if we want to handle a set of URLs that may be constantly changing? Let's take a look at how we can use variable rules to allow for dynamic URLs.

When specifying the URL to bind to a view function, we have the option of making any section of the path between the slashes (/) variable by indicating `<variable_name>`. These variable parts will then be passed to the view function as arguments. For example:

```
@app.route('/orders/<user_name>/<int:order_id>')
def orders(user_name, order_id):
    return f'<p>Fetching order #{order_id} for {user_name}.</p>'
```

Now, URLs like `'/orders/john/1'` and `'/orders/jane/8'` can all be handled by the `orders()` function.



Note that we can also optionally enforce the type of the variable being accepted using the syntax: `<converter:variable_name>`. The possible converter types are:

**string**      accepts any text without a slash (default)

**int**      accepts positive integers

**float**      accepts positive floating point values

**path**      like string but also accepts slashes

**uuid**      accepts UUID strings

## Instructions

### 1.

Update the `reporter` endpoint to handle requests from individual reporter pages whose URL path follows the pattern `/reporter/#`.

Add a variable section called `reporter_id` to the URL passed to the `route()` decorator. Use a converter to specify that the variable part must be a positive integer.

---

Hint

The syntax for specifying a variable section of the URL with a converter is `<converter:variable_name>`.

### 2.

Next, update the `reporter()` function to take a parameter called `reporter_id`. In the function body, modify the `<h2>` heading to read `Reporter # Bio`, where `#` is the `reporter_id`.

---

Hint

You can use Python f-strings to help format the HTML to return, as shown below:

```
return f'''
    <h2>Reporter {my_variable_here} Bio</h2>
    <a href="/">Return to home page</a>
'''
```

where `my_variable_here` is the `reporter_id`.

For more information on f-strings, check out the [Python documentation on them here](#).

### 3.

Run **app.py**. Try visiting various reporter pages such as <https://localhost/reporter/1> or <https://localhost/reporter/1000!>

**app.py**

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
@app.route('/home')
def home():
    return '<h1>Hello, World!</h1>'

@app.route('/reporter/<int:reporter_id>')
def reporter(reporter_id):
    return f'''
    <h2>Reporter {reporter_id} Bio</h2>
    <a href="/">Return to home page</a>
    '''
```

---

## Review

Congratulations on building your first Flask app!

You've learned to:

- Import the `Flask` class and create an application object

```
from flask import Flask
app = Flask(__name__)
```

- Define routes for handling requests sent from various URLs

```
@app.route('/')
def home():
    return '<h1>Hello, World!</h1>'
```

- Create variable rules to handle dynamic URLs

```
@app.route('/orders/<user_name>/<int:order_id>')
def orders(user_name, order_id):
    return f'<p>Fetching order #{order_id} for {user_name}</p>'
```

Time to put what you've learned to the test!

## Instructions

### 1.

Define a third view function called `article()` that is bound to the URL path `/article`. The function should return an `<a>` tag with the text `Return back to home page` that links to `/`.

Run **app.py** in the terminal and navigate to <http://localhost:5000/article> in the browser.

---

Hint

You can create a view function in Flask with a bound URL as shown below:

```
@app.route("url_to_bind")
def my_view_function():
    return f'''
    <a href="/">Return back to home page</a>
    '''
```

- Replace `"url_to_bind"` with the URL you wish to bind to the view function
- Replace `my_view_function` with the name of the route

### 2.

Now, add a variable rule such that a URL whose path follows the pattern `/article/X` will trigger the `article()` function. Name the variable part `article_name`.

---

Hint

You can create a variable rule in a Flask route as follows:

```
@app.route("url_to_bind/<variable>")
def my_view_function():
    return f'''
    <a href="/">Return to home page</a>
    '''
```

Replace `variable` with the variable part of the URL.

### 3.

Update the `article()` function to take a parameter called `article_name`. Since this value is the URL slug, assume `article_name` will be all lower-cased with hyphens separating each word. For example, `article_name` could look like this:

```
ten-ducks-enter-local-pond
```

In the function body, replace the hyphens with spaces and turn the text to title-case. Then, before the `<a>` element in the returned HTML, add a `<h2>` heading containing the formatted title. Check the hint for guidance on how to make these formatting changes.

Try visiting various article pages such as <http://localhost:5000/article/my-first-flask-app!>

---

### Hint

You can use Python's built-in `.format()` and `.title()` function to help format the article name in a `<h2>` tag as follows:

```
<h2>{article_name.replace('-', ' ').title()}</h2>
```

Paste the above line of code within the `article()` view function before the `<a>` element.

### app.py

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
@app.route('/home')
def home():
    return '<h1>Hello, World!</h1>'

@app.route('/reporter/<int:reporter_id>')
def reporter(reporter_id):
    return f'''
    <h2>Reporter {reporter_id} Bio</h2>
    <a href="/">Return to home page</a>
    '''

@app.route('/article/<article_name>')
def article(article_name):
```

```
return f'''
<h2>{article_name.replace('-', ' ').title()}</h2>
<a href='/>Return back to home page</a>
'''
```