Flask 101

An introduction for developers

What is Flask

A web application micro-framework written in Python. Simple.

Follow-along repo:

github.com/nationalarchives/flask-101

Micro-frameworks provide a solid core...

Three dependencies:

- **Werkzeug** provides routing, debugging and Web Server Gateway Interface (WSGI)
- **Jinja2** provides template support
- **Click** provides command-line integration

These are all authored by Armin Ronacher, the author of Flask

... you then select *extensions* to provide the rest

...but remember, Flask does not provide a lot of the things you might expect if coming from, say, Laravel

A quick look at WSGI

Web Server Gateway Interface (WSGI*)

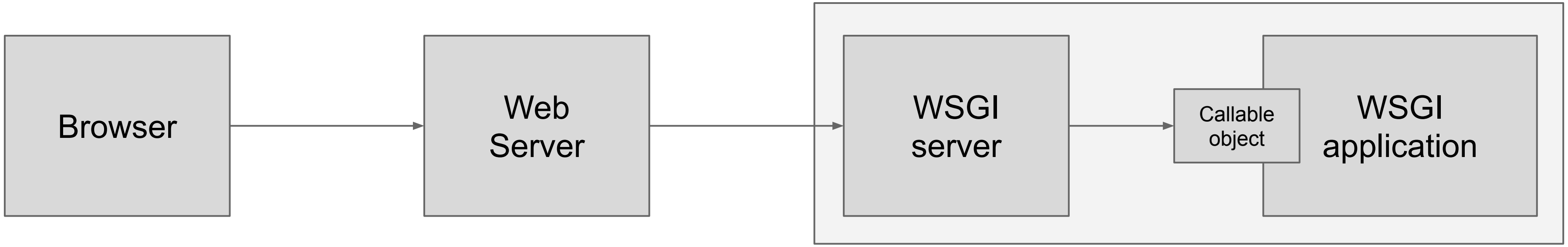
- A calling **convention for web servers to forward requests to applications** written in Python
- Specified (and standardised) in PEP 3333
- Has two sides:
  - A server/gateway side (often a full web server such as Apache)
  - The application/framework side (a Python callable)

* Pronounced *whiskey or 'Whiz Ghee',* apparently.

OK. So, what do I actually need to know about WSGI?

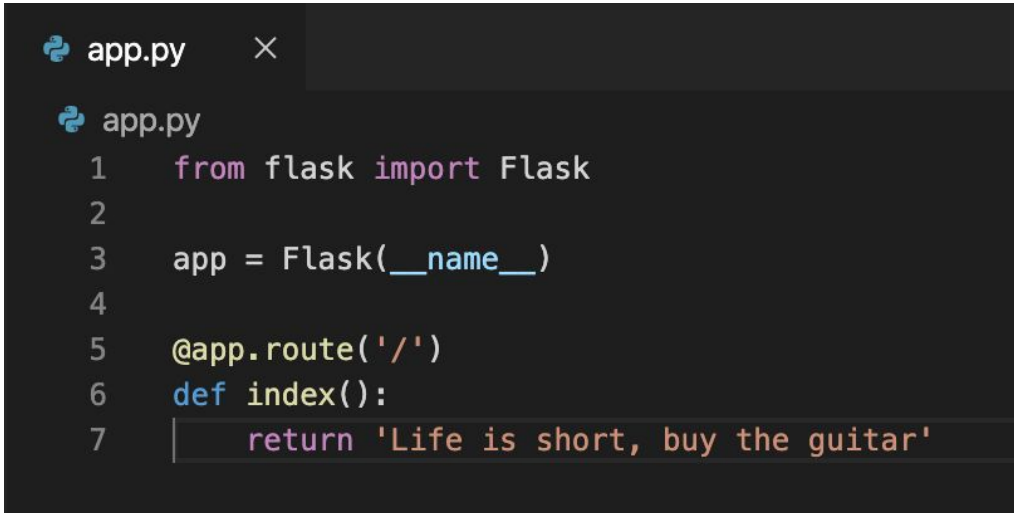
Initially, all you need to know is:

- a WSGI container is a separate process that runs on a different port to your web server
- Your web server is configured to pass requests (some, not all) to the WSGI container which runs your web application, then passes the response back to the requester



Routing, request and redirects

A tiny but complete Flask application



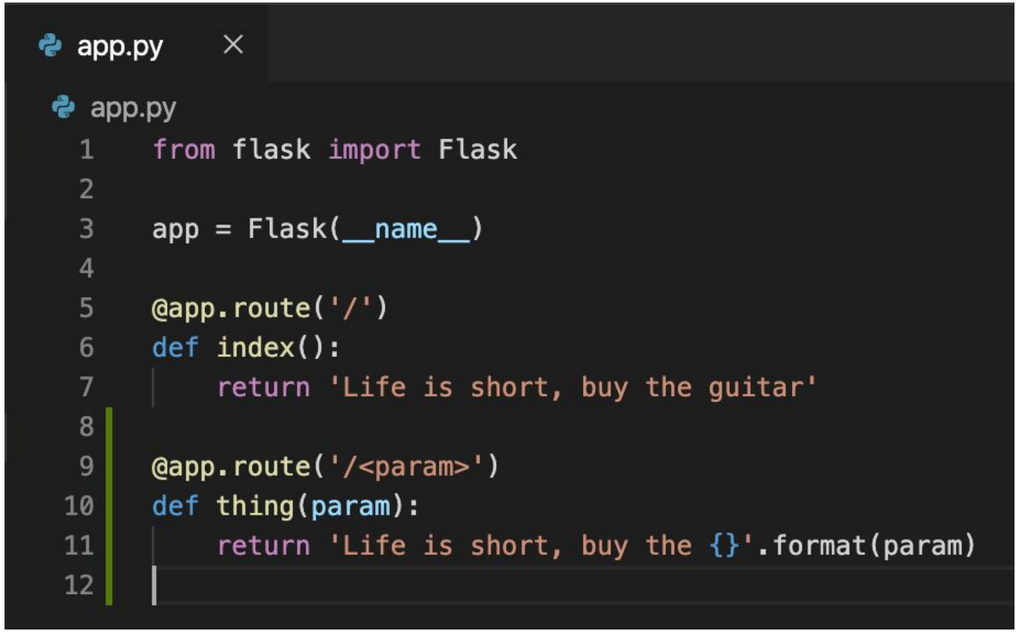
app.py

app.py

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    return 'Life is short, buy the guitar'
```

- Imports Flask
- Creates an application instance
- Decorates our **`index():`** method with the **`@app.route`** decorator. In doing so we create our first route function (with index() being run when '/' receives a HTTP request)

Dynamic routes



app.py ×

app.py

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    return 'Life is short, buy the guitar'

@app.route('/<param>')
def thing(param):
    return 'Life is short, buy the {}'.format(param)

```

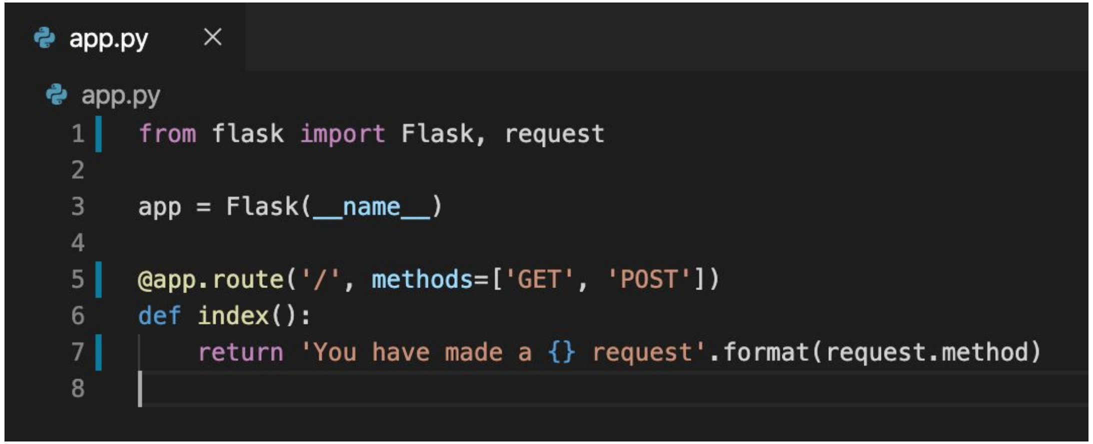
Here we add a dynamic route.

Flask supports `string`, `int`, `float`, and `path`* for routes.

* a special type of string that can include forward slashes.

Play with this using: **git checkout dynamic-routes**

Specifying accepted methods


app.py

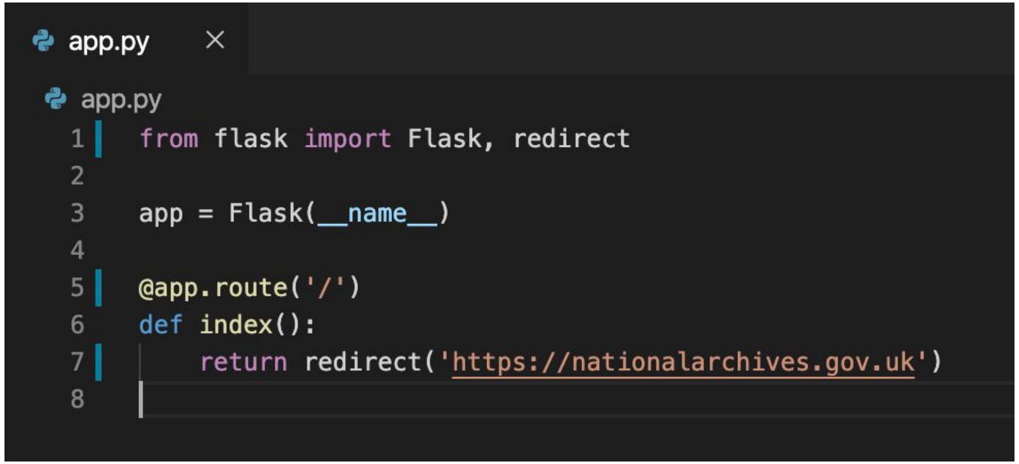
```python
from flask import Flask, request

app = Flask(__name__)

@app.route('/', methods=['GET', 'POST'])
def index():
    return 'You have made a {} request'.format(request.method)

```

Play with this using: **git checkout specify-http-methods**

By default, the route decorator allows any HTTP methods but you also have the ability to whitelist only those you want to permit.

Note also the import of `request` and how this allows us to get information about the request

Redirects

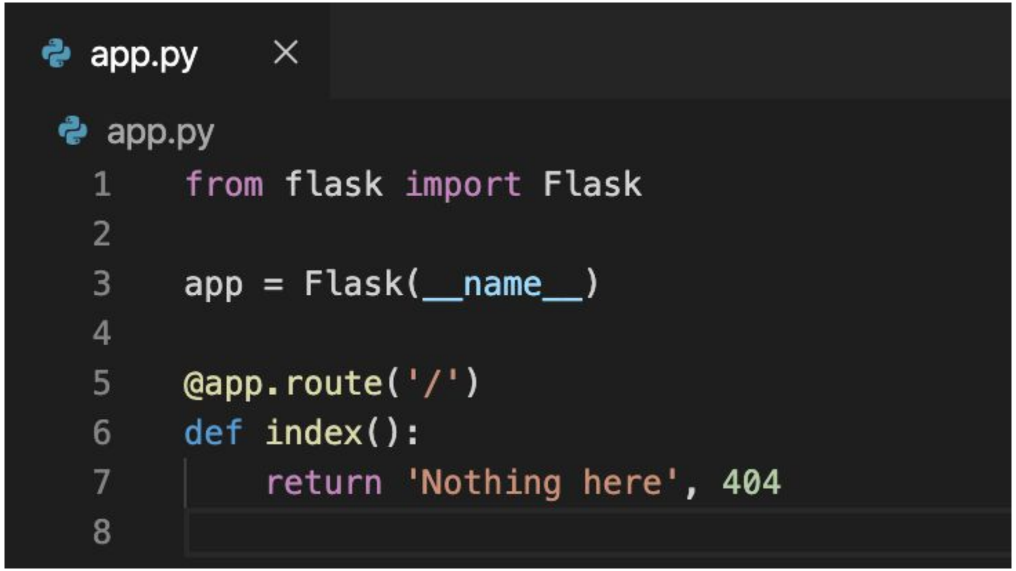

app.py

```
from flask import Flask, redirect

app = Flask(__name__)

@app.route('/')
def index():
    return redirect('https://nationalarchives.gov.uk')

```

To perform redirects we import `redirect` from `flask`

Play with this using: **git checkout redirects**

Specifying status codes



app.py

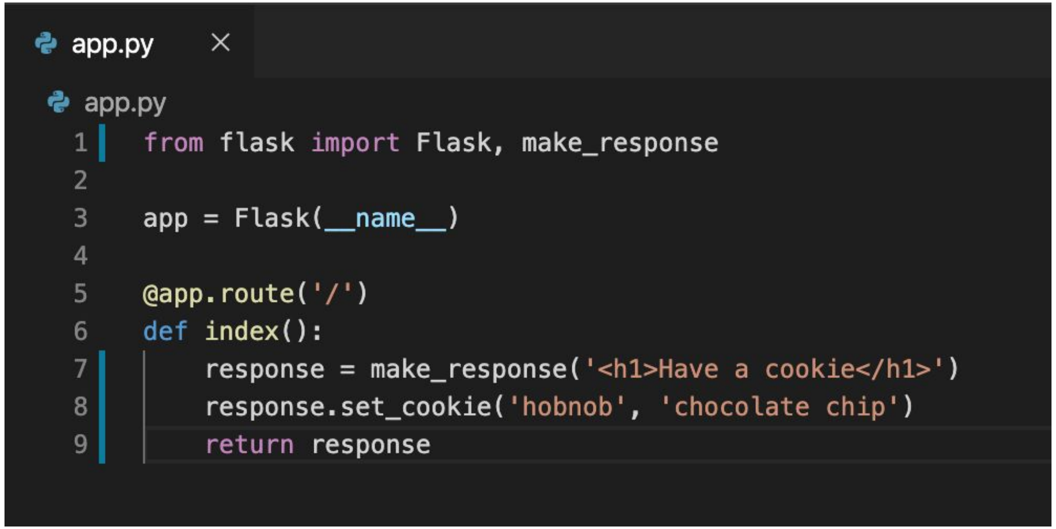
app.py

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    return 'Nothing here', 404

```

By returning a tuple from our view functions we can specify the response HTTP status code.

Play with this using: **git checkout specify-http-methods**

Returning a response object



app.py

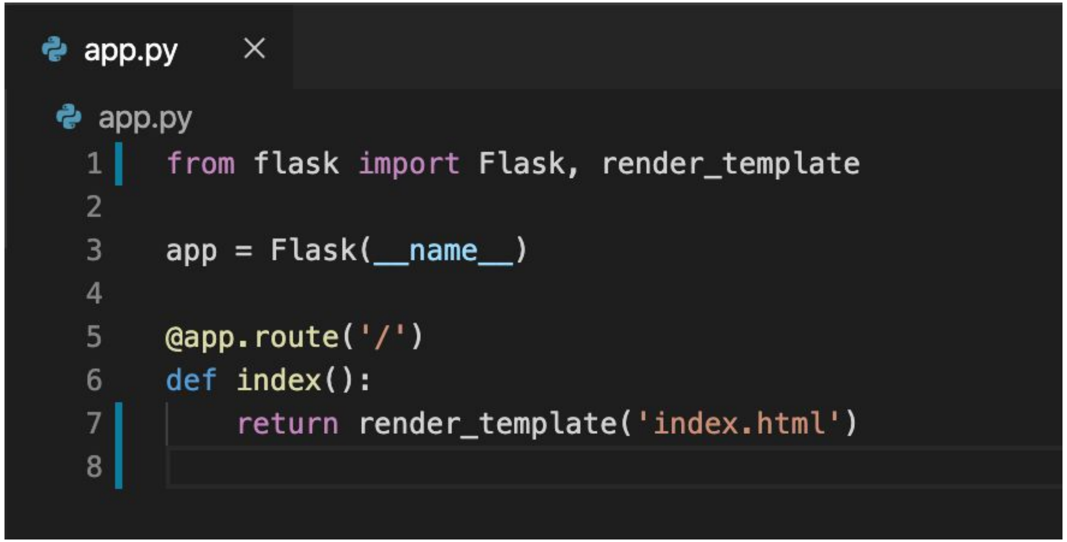
```python
from flask import Flask, make_response

app = Flask(__name__)

@app.route('/')
def index():
    response = make_response('<h1>Have a cookie</h1>')
    response.set_cookie('hobnob', 'chocolate chip')
    return response
```

Returning tuples obviously doesn't scale too well, so Flask provides `make_response()` to prepare a response object

Play with this using: **git checkout response-object**

Templates

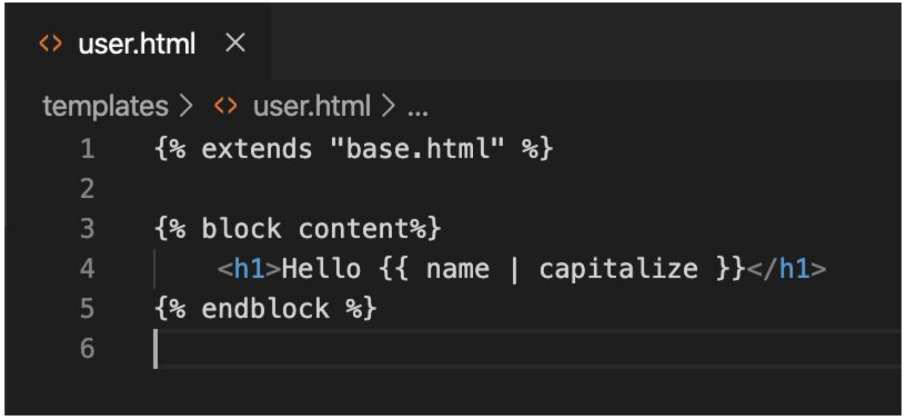
A simple template



```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')
```

Flask uses the Jinja2 Template engine

Jinja2 template engine



```
{% extends "base.html" %}

{% block content%}
    <h1>Hello {{ name | capitalize }}</h1>
{% endblock %}

```

We won’t dwell on the capabilities of Jinja2. It provides everything you’d expect, including:

- Template inheritance
- Includes
- Variables
- Control structures: conditionals, loops

It also provides:

- Macros (Python functions you define and import into the templates that need them)
- Predefined filters, including: trim, upper, lower, striptags and safe

The repository has an implementation of templates using several of these features. To explore use: **git checkout add-templates**

Command-line basics

Command-line options

Some useful command line options include:

- `flask run` to start a development server
- `flask shell` opens a Python shell in the context of the application
- `flask *[command]* --help` to see available options for the command

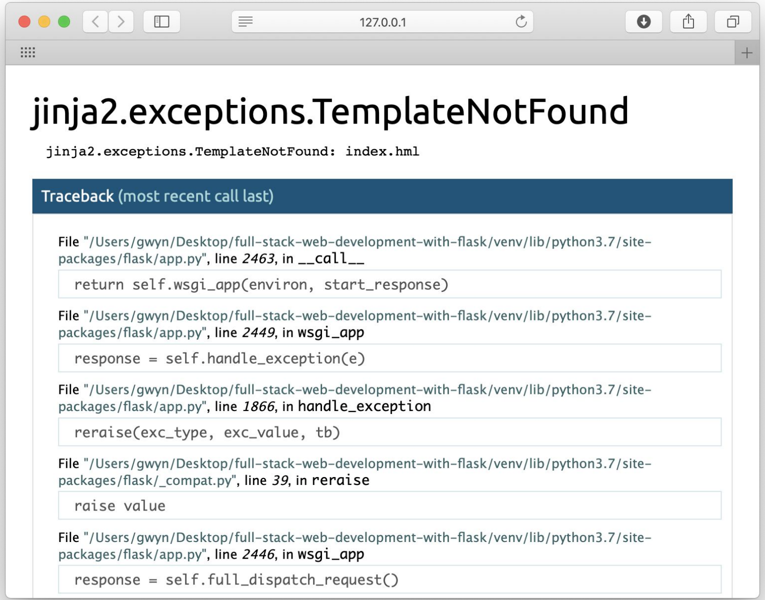
Note: you can also create your own command-line methods by importing Click and using it to decorate your methods

Other cool stuff…

Debug mode

Flask applications can optionally be executed in debug mode. This enables two modules:

- ***Reloader***: watches the source code and restarts the server when a change takes place (it doesn't refresh the browser)
- ***Debugger***: transforms the web browser into an **interactive stack trace** that allows you to:
    - Inspect source code
    - Evaluate expressions in any place in the call stack 👏



127.0.0.1

jinja2.exceptions.TemplateNotFound

```
jinja2.exceptions.TemplateNotFound: index.hml
```

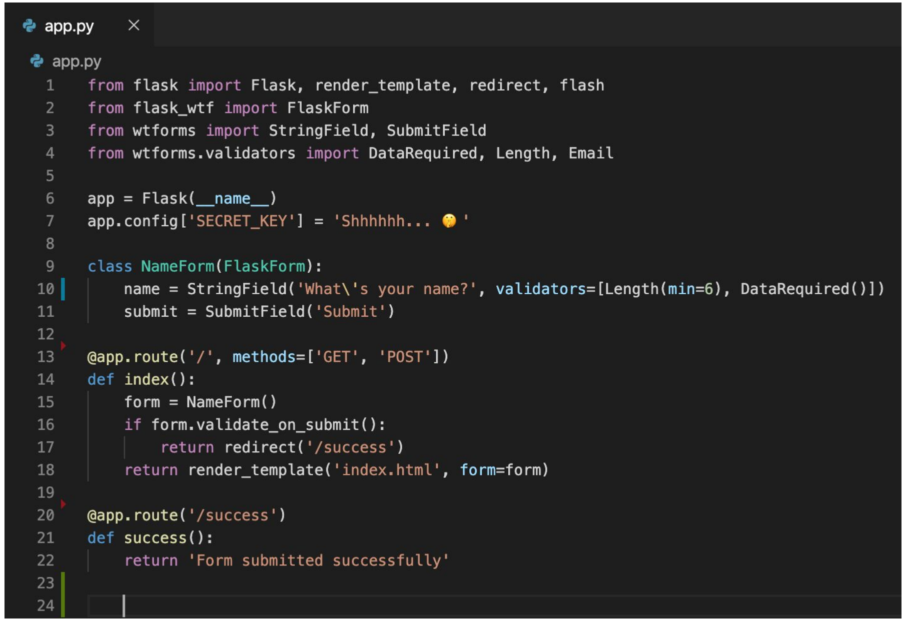
Traceback (most recent call last)

```
File "/Users/gwyn/Desktop/full-stack-web-development-with-flask/venv/lib/python3.7/site-packages/flask/app.py", line 2463, in __call__
    return self.wsgi_app(environ, start_response)
File "/Users/gwyn/Desktop/full-stack-web-development-with-flask/venv/lib/python3.7/site-packages/flask/app.py", line 2449, in wsgi_app
    response = self.handle_exception(e)
File "/Users/gwyn/Desktop/full-stack-web-development-with-flask/venv/lib/python3.7/site-packages/flask/app.py", line 1866, in handle_exception
    reraise(exc_type, exc_value, tb)
File "/Users/gwyn/Desktop/full-stack-web-development-with-flask/venv/lib/python3.7/site-packages/flask/_compat.py", line 39, in reraise
    raise value
File "/Users/gwyn/Desktop/full-stack-web-development-with-flask/venv/lib/python3.7/site-packages/flask/app.py", line 2446, in wsgi_app
    response = self.full_dispatch_request()
```

By default, debug mode is disabled. To enable it, set a FLASK_DEBUG=1 environment variable before invoking flask run:

Let's add an extension!

Flask-WTF



```python
from flask import Flask, render_template, redirect, flash
from flask_wtf import FlaskForm
from wtforms import StringField, SubmitField
from wtforms.validators import DataRequired, Length, Email

app = Flask(__name__)
app.config['SECRET_KEY'] = 'Shhhhhh... 🤫 '

class NameForm(FlaskForm):
    name = StringField('What\'s your name?', validators=[Length(min=6), DataRequired()])
    submit = SubmitField('Submit')

@app.route('/', methods=['GET', 'POST'])
def index():
    form = NameForm()
    if form.validate_on_submit():
        return redirect('/success')
    return render_template('index.html', form=form)

@app.route('/success')
def success():
    return 'Form submitted successfully'

```

The `request` object in Flask is capable of handling forms, but there are extensions that could make things easier. One such extension is **Flask-WTF**

If you'd like to follow along, do this:

- install it with **`pip install flask-wtf`**

Play with this using: **git checkout flask-wtf**

Flask extensions can provide…

Creating RESTful APIs, analytics generation, session management, security (many aspects of), authentication (including using OAuth and OpenID), working with databases (both SQL and document-based), database migrations, caching, data validation, email, internationalization, full-text search, route rate limiting, queueing, exception tracking, SDK integrations (Google maps, Gravatar, Pusher), CORS, debugging, documentation, testing

There is a curated ‘awesome list’: https://github.com/humiaozuzu/awesome-flask

…but, as always, be judicious when using other people’s code.