Flask for Newbies

Greg Horie

… for Newbies

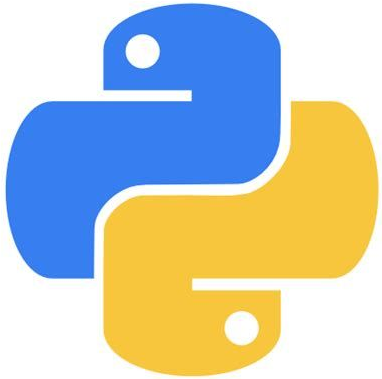
Because I'm new too !

Not Covered

- CSS
- Javascript

Will Cover

- Flask
- Python
- Some HTML
- And a bit of VS Code



What is Flask?

- Micro-framework for python web development.
- What are the benefits of Flask?
  - Flask is simple and extensible.
  - It leaves most choices to the web developer.
  - Flask has many extensions that add functionality when required.
  - Example extensions:
    - Form handling.
    - Database integrations.
    - Authentication.
    - Email.
    - Etc.

Configurations and Conventions

- Flask has sensible defaults.
  - Few config adjustments are required.
  - Changes are possible, but not required very often.
- Flask has a few conventions:
  - HTML templates - Stored in `templates/` directory of your project.
  - Static files - Stored in `static/` directory of your project.
- Starting point is code (rather than configuration).

Why Choose Flask?

- Well-suited for people starting their python web development journey.
- Focus is on the code, not the configuration.
  - One less concern in the "full stack"
- Great for simple apps.
  - e.g. small web sites, IoT integration apps, REST APIs
- Still possible to scale up to production-worthy applications.

Flask

web development,
one drop at a time

Flask Prep

```
# requirements.txt file with the following python packages:
flask
flask-wtf
flask-sqlalchemy
flask-migrate

# activate a virtual environment for your project
$ python -m venv venv
$ source venv/bin/activate

# install required packages
(venv) $ pip install -r requirements.txt
```

Hello World!

hello.py:

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello():
     return "<h1>Hello World!</h1>"

if __name__ == "__main__":
     app.run()
```

```
# Let's try it ...

(venv) $ python hello.py
```

What did we learn?

- 7 lines of code gives you a very basic Flask app.
- `app = Flask(__name__)` required to set the application context.
- `@app.route("/")` decorator maps url to function.
- `app.run()` starts the Flask webserver.
- IPv6 supported.

Jinja2 Templating + Debug

crazy_cats.py:

```
...
@app.route('/')
def index():
    cat = get_random_cat_pic()
    return render_template('cats.html',
                           gif=cat)

def get_random_cat_pic():
    return random.choice(os.listdir('static'))

if __name__ == '__main__':
    app.run(host="::", debug=True)
```

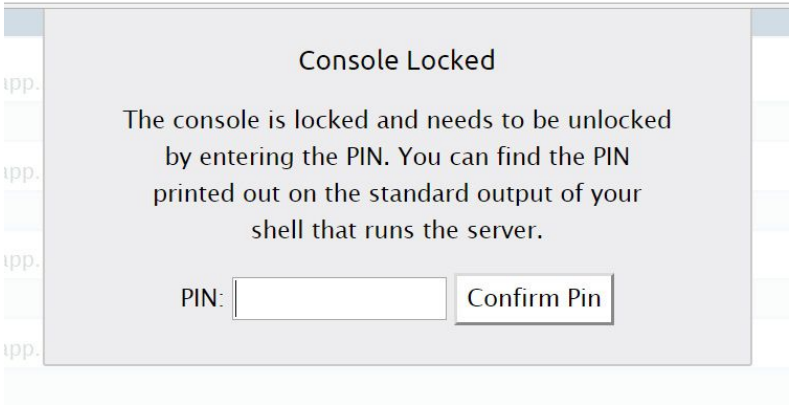
templates/cats.html:

```
<!DOCTYPE html>
<html>
      <head>
      <meta charset="utf-8">
      <title>Cat Page</title>
      </head>
      <body>
      <h1>Welcome to my Cat Page!</h1>
      <img src="{{ url_for('static',
                filename=gif) }}",
                width="400" height="400">
      </body>
</html>
```

Jinja2 Templating + Debug

```
# Let's try it ...
(venv) $ python crazy_cats.py
...
 * Debug mode: on
 * Running on http://[::]:5000/ (Press
CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 110-429-491
```

Debugger ...



What did we learn?

- `render_template()` method used to generate an http response.
  - Arguments passed can be used within the html template.
- Jinja2 templating system is used by Flask (similar to DTL).
- `url_for()` helper method available to avoid hard-coding URLs.
- Flask live debug mode improves visibility when coding.
  - **Note** - Poor practice to hard-code debug mode into your code.
  - Consider using environment variables with python-dotenv or envdir.

Template Inheritance, Macros & Filters

base.html:

```
<!DOCTYPE html>
<html>
      <head>
      <meta charset="utf-8">
      <title>Cat Page</title>
      </head>
      <body>
{% block content %}

{% endblock %}
      </body>
</html>
```

cats.html:

```
{% extends "base.html" %}
{% block content -%}

{% macro render_cat_page() %}
<h1>Welcome to my Cat Page!</h1>
<img src="{{ url_for('static',
filename=gif) }}", width="400"
height="400">
{% endmacro %}

{{- render_cat_page() | indent(8) }}

{%- endblock %}
```

Template Inheritance, Macros & Filters

```
# Let's try it ...

(venv) $ python crazy_cats.py
```

What did we learn?

- Flask supports DRY principles (don't repeat yourself).
  - Template inheritance allows for code reuse.
  - Macros also support code reuse.
- Many filters are available for data transformations in jinja2
  - If its a common transformation, then a filter is most likely already available (e.g. to_json, to_yaml, etc.)

WTForms

crazy_cats.py:

```
from flask import Flask, render_template
from flask_wtf import FlaskForm
from wtforms import SubmitField
import os
import random

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

class RandomCatForm(FlaskForm):
     submit = SubmitField('More Cats!')
```

```
@app.route('/', methods=['GET', 'POST'])
def index():
     form = RandomCatForm()
     cat = None
     if form.validate_on_submit():
          cat = get_random_cat_pic()
     return render_template('cats.html',
                            form=form, gif=cat)

...
```

WTForms

cats.html:

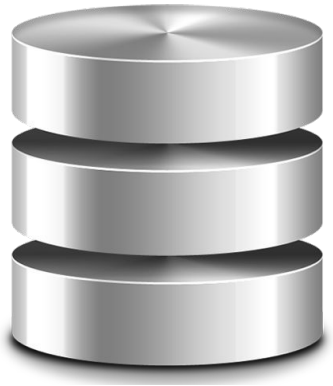
```
{% extends "base.html" %}
{% block content -%}

{% macro render_cat_page() %}
<h1>Welcome to my Cat Page!</h1>
<form method="POST">
     <!-- CSRF protection -->
     {{ form.hidden_tag() }}
     {{ form.submit() }}
</form>
<br>
```

```
{% if gif != None %}
<img src="{{ url_for('static',
filename=gif) }}", width="400"
height="400">
{% endif %}
{% endmacro %}

{{- render_cat_page() | indent(8) }}

{%- endblock %}
```

What did we learn?

- Flask implements HTML forms by inheriting from the FlaskForm class.
  - Extend this class with the elements required in your form.
  - The form is also passed as an argument to **`render_template()`**.
- The WTForms includes CSRF security.
  - **Note** - Secrets should normally not be hard-coded.
- Conditional logic can also be implemented in the Jinja2 templating.

Setting up a Database for Flask



- We'll use sqlite3.
- High-level steps for database:
  - Create a database instance for our flask app.
  - Create a model that maps to a database table.
    - No SQL knowledge required.
    - Typical project will add all CRUD operations to the application.
    - For this demo we'll stick to Read.
  - "Migrate" model (table) into the database.
  - Insert any initial data into the database.
- Database now ready for flask integration.

Flask SQLAlchemy & Flask Migrate

crazy_cats.py:

```
import os
from flask import Flask, render_template
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate

app = Flask(__name__)

basedir = os.path.abspath(os.path.dirname(__file__))
dbfile = 'data.sqlite'
app.config['SQLALCHEMY_DATABASE_URI'] = f"sqlite:///{basedir}/{dbfile}"
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
```

Flask SQLAlchemy & Flask Migrate

```
db = SQLAlchemy(app)  # bind database instance with the flask app
Migrate(app, db)  # initializes the extension with the flask cli interface

class Cats(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.Text)
    pic = db.Column(db.Text)

    def __init__(self, name, pic):
    self.name = name
    self.pic = pic

...
```

Database Cheatsheet

```
# indicate .py application to flask
$ export FLASK_APP=crazy_cats.py

# create sqlite3 migrations directory
# and files
$ flask db init

# create migration scripts and db file
$ flask db migrate -m "first migration"

# create/update db schema for app
$ flask db upgrade
```

```
# sqlite3 cheat sheet

# access sqlite3 database
$ sqlite3 data.sqlite

# show tables
sqlite> .tables

# show table schema
sqlite> .schema

# show records in the cats table
sqlite> select * from cats;

# log out
sqlite> .exit
```

Load SQL Data

init_data.py:

```
# simple script to load cat records
from crazy_cats import db, Cats

cat_records = [
     ("Boogie", "boogie.gif"),
     ("Hugs", "hug_me.gif"),
     ...
]
for rec in cat_records:
     cat = Cats(rec[0], rec[1])
     db.session.add(cat)
db.session.commit()
```

```
(venv) $ python init_data.py

(venv) $ sqlite3 data.sqlite "select *
from cats;"
```

What did we learn?

- Flask abstracts database interactions with the SQLAlchemy.
  - SQLAlchemy is an ORM framework.
- Flask Migrate can be used to manage database creation and updates.
  - Further abstraction so the developer can focus on python, not SQL.
- Data loading can be done in many ways including a simple init script.
- Sqlite3 is a nice starting point if you are not familiar with SQL databases.
  - Note - Other options (e.g. PostgreSQL) should be considered for scale.

Integrating App and Database

crazy_cats.py:

```
...
# functions to handle http requests
@app.route('/')
def index():
    return render_template('home.html')

@app.route('/list')
def list_cats():
    # Grab a list of cats from database.
    cats = Cats.query.all()
    return render_template('list.html', cats=cats)
```

Integrating App and Database

list.html:

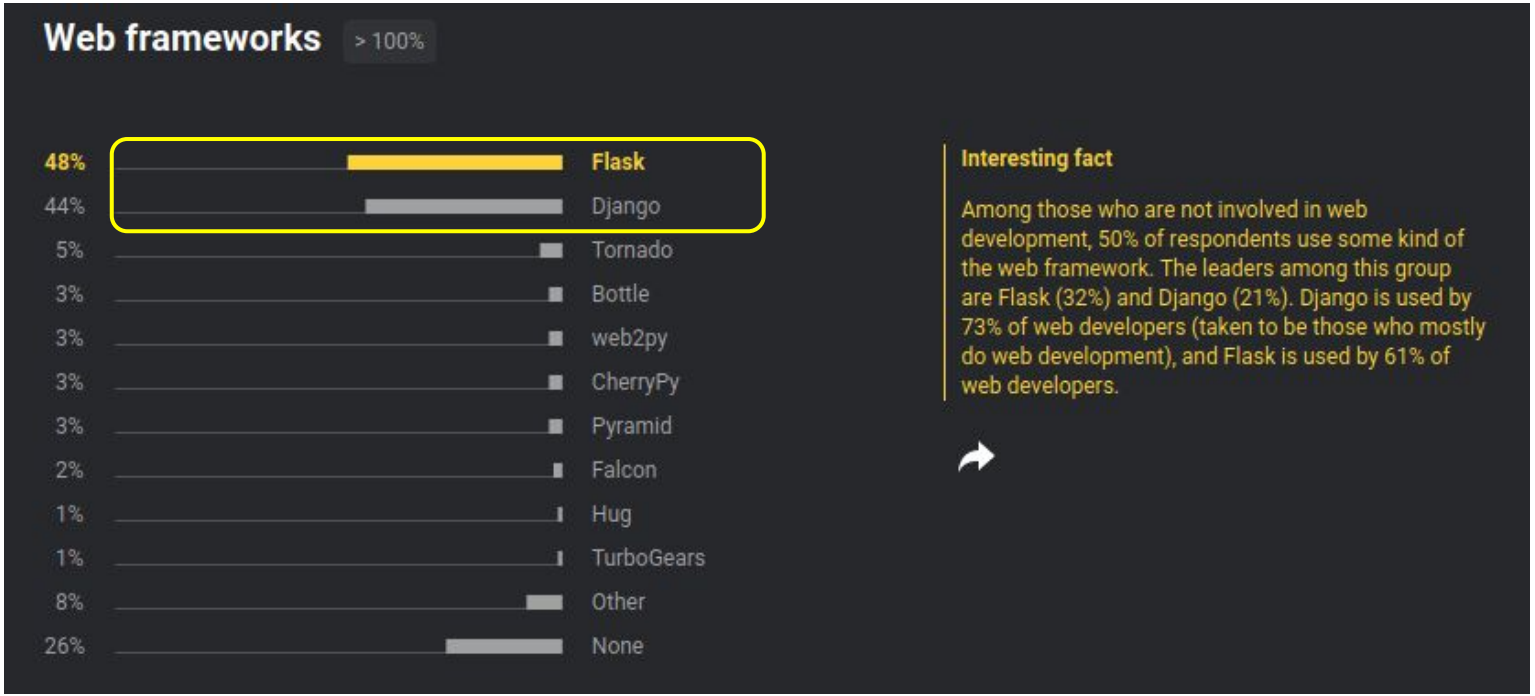
```
{% extends "base.html" %}
{% block content %}
  <h1>Here is a list of all available cats.</h1>
  <ul>
     {% for cat in cats  %}
     <li><a href="{{ url_for('static', filename=cat.pic) }}">{{ cat.name }}</a>
     {% endfor %}
  </ul>

</div>
{% endblock %}
```

```
(venv) $ python crazy_cats.py
```

What did we learn?

- Flask can be easily integrated with an SQL database.
- `SQLAlchemy.db.Model.query.all()` method used to list an entire table.
  - SQLAlchemy has a rich set of methods that allow you to query SQL databases.
- SQLAlchemy abstracts the database implementation specifics.
  - Allow you to focus on your app rather than the database.

Flask or Django?



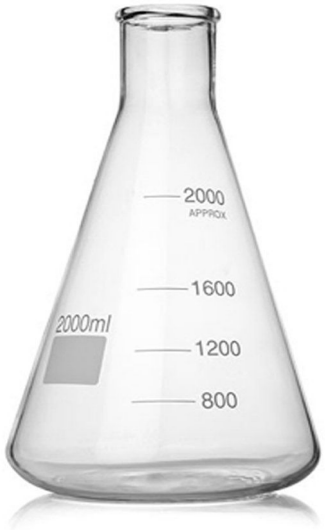
https://www.jetbrains.com/lp/python-developers-survey-2019/

Flask or Django

- Both are excellent, well-supported options
- Flask may be easier for new developers
  - Challenge - Provides few "guard rails"
  - Extensions required to flesh out capabilities.
- Django is great for experienced web developers
  - "Batteries included" - database integration, users, admin console, etc.
  - Opinionated about project structure.
  - Time-investment required to understand.

Summary

- Flask is a python web development micro-framework.
- Flask is well-suited to new python developers.
- Give it a try - It's fun!

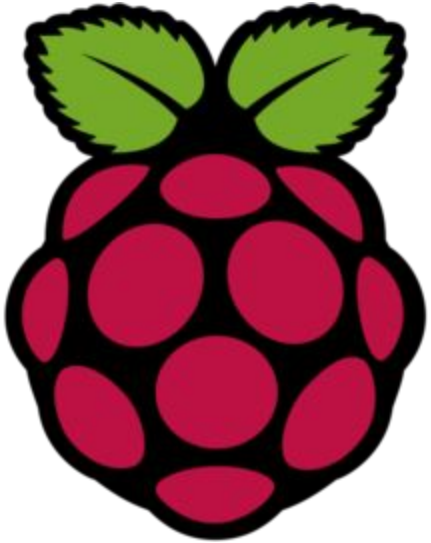


Possible Future Discussions

- More Flask
  - Full CRUD app
  - REST API
  - Auth / OAuth
  - Elasticsearch and/or Prometheus integration
- Bootstrap
  - Or other CSS / Javascript magic
- Python Streamlits
- Click for CLI tools
- Ansible for automation
- Containers / Docker / Kubernetes
- Idiomatic Python

VicPiMakers and Others Slack

- Please let us know if you want an invite to this Slack group



Backup Slides