Flask cheat sheet v3.0

Got some of this from blog.miguelgrinberg.com/post/the-flask-mega-tutorial

---

Naming things is hard

First, come up with *two* names for your project: one which will be referred to as "the site name," and one as "the app name." The difference between these two names, and why there even *are* two names, is weird and confusing. Just do your best and come up with two names. (If your site has to do with basketball, for example, you could use "`hoopsSite`" for the site name and "`basketball`" for the app name.)

From this point forward in the instructions, whenever I say to type `mysite`, substitute your site name (e.g., `hoopsSite`). **Whenever I say to type `myapp`, substitute your app name** (e.g., `basketball`).

Install

Python virtual environment

It's just good practice to use virtual environments.

```
$ mkdir mysite
$ cd mysite
$ python -m venv venv
$ source venv/bin/activate
```

---

git repo

It's just good practice to use a git repo.

```
$ git init .
```

Also create a `.gitignore` file using vim, with these contents:

```
.gitignore
venv
__pycache__
```

Also create a `README.md` file, also using vim, with these contents:

give it these contents:

```
A sample Flask app. (...or whatever you want it to say...)
```

and commit it to your repo:

```
$ git add README.md
$ git commit -m "Initial commit."
```

Make sure you can run the "`git log`" and "`git status`" commands correctly before continuing.

Flask

```
$ pip install flask
```

Tip: when you create your alias to start up Flask, have it include "`source venv/vin/activate`" before "`export FLASK_APP=myapp`" and "`flask run`". The alias in my `.bashrc` for starting the Ben & Jerry's project looks like this:

```
alias run="cd /mydir ; source venv/bin/activate ; export FLASK_APP=bj ; pwd ; flask run"
```

After I've got all the stuff below setup, I will then be able to simply type "`run`" at the command line and it will start my Flask server at port 5000.

"Hello world"

Create scaffolding

Make a new directory `mysite` (if you didn't just do that), and create these files inside it:

- `__init__.py` with contents:

```
from flask import Flask
myapp = Flask(__name__)
from mysite import routes
```

- `myapp.py` with one line: "`from mysite import myapp`"
- `routes.py` with contents:

```
from mysite import myapp
@myapp.route("/")
def index():
    return "Hello world!"
```

Run "hello world"

Either execute your alias (in my case, "`run`"), if you made one as suggested above, or else type:

```
$ export FLASK_APP=myapp
$ flask run
```

Then go to http://localhost:5000.

(Sometimes you'll see people instead pip install `python-dotenv` and then add "`FLASK_APP=myapp.py`" in a new `.flaskenv` file in the `mysite` directory. This way, you can simply type "`flask run`" to start your app, without having to specify `FLASK_APP` each time. Either way works. I prefer my alias. If you do this `.flaskenv` thing, you'll probably want to include "`.flaskenv`" in your `.gitignore` file.)

Basic dynamic web page

Flask uses the Jinja template engine, which has lots of powerful and convenient features. Here's the very very basics:

1. Create a `templates` directory in `mysite`.
2. In `templates`, create a `basic.html` with double-curly ("durly") markup:

```
<HTML>
...
        I have {{ numPairs * 2 }} total shoes.
...
</HTML>
```

---

3. In `routes.py`:

```
from flask import render_template
...
def index():
    ...
    return render_template("basic.html", numPairs=17)
```

Now restart flask, refresh your browser, and see what that did.

---

4. Can iterate through collections in a template like this:

```
<HTML>
...
        {% for hobby in hobbies %}
            ...stuff in here with tables, lists, etc., using "hobby"...
        {% endfor %}
...
</HTML>
```

---

5. Can use `if`-statement-like syntax in a template like this:

```
<HTML>
...
        {% if hobby == "skydiving" %}
            ...stuff...
        {% elif hobby == "paperclip collecting" %}
            ...stuff...
        {% else %}
            ...stuff...
        {% endif %}
...
</HTML>
```

---

6. A very common need in a dynamic web page is to generate a URL (often to another dynamic web page). The basic HTML syntax for this uses the `<a>` (for "anchor") tag (`<a href="http://umw.edu">Go UMW!</a>`).

But in a Jinja template, it is convenient to call the Flask `url_for()` function to accomplish this. Its first argument is the *function* in your `routes.py` that you want to URL to go to. Other (keyword) arguments may follow, in which case `url_for()` will automatically escape any necessary characters and glom them on as HTTP parameters to the end of the URL.

For example, if you had this in your `routes.py`:

```
def show_ingr():
    the_recipe = request.args['recipe']
    ...
```

and this in a Flask template:

```
Love Mom's <a href="{{url_for(show_ingr,recipe='lemon pie')}}">lemon pie</a>!
```

then when rendered, the template will produce this output:

```
Love Mom's <a href="/show_ingr?recipe=lemon+pie">lemon pie</a>!
```

which is exactly what you want.

Beyond the basics

There's lots, lots more Jinja features to explore.

Using template inheritance

Coming soon!

HTML forms

Coming soon!

Session management

Client-side storage.

In encrypted cookie value, stored in each browser, dutifully returned to the server whenever another request is sent.

1. In order to enable this, you need to have this one line somewhere in your `__init__.py`:

```
myapp.secret_key = "Some random string better than this!"
```

2. Import `session` from `flask`, and then this variable is available to you as a dict-like object holding key/value pairs *only* for the current users. (All other users have their own key/value session pairs which do not interfere with each other. This magic is achievable through cookie passing.

Server-side storage.

Coming soon!

An important weirdness with sessions "knowing" they've been updated. The issue is this. If you change one of the key/value pairs in the session — and by that I mean you change **which object a key is referring to** — then Flask "knows" you changed the session and dutifully sends the updated version to the client for storage, as it should.

However, if you only modify what's **in** one of the object values, rather than changing which object that key points to, Flask plays dumb and thinks you haven't changed the session.

For this reason, you'll need to add this line of Python code (see p. line 25 of routes.py in the class github repo):

```
session.modified = True
```

on the line immediately after changing merely the *contents* of a collection, rather than substituting an entirely different collection object some session value. (And if that's confusing, realize that `session.modified = True` is always a safe operation.)

Running on the Cloud

Coming soon!