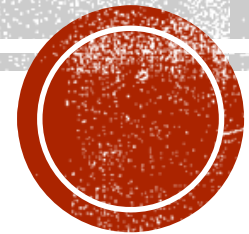


# INTRODUCING FLASK

Assumes you know python



# YOU NEED TO KNOW PYTHON

Crash course in Python:

`http://diveintopython3.problemsolving.io/index.html`



# WHAT IS FLASK?

- Created by Armin Ronacher
- A micro-framework in Python
  - No database layer or ORM
  - No built-in authentication
  - No admin interface



# INSTALL A VENV (PYTHON 3.7+)

```
microblog>python -m venv flaskenv
```

- Results in

```
flaskenv>ls  
bin  include  lib  lib64  pyvenv.cfg  share
```

```
flaskenv>cd ..  
microblog>source flaskenv/bin/activate  
(flaskenv) microblog>
```



# INSTALL FLASK

```
(flaskenv) microblog>pip install flask
Collecting flask
  Using cached https://files.pythonhosted.org/packages/9b/93/628509b8d5dc749656a9641f4caf13540e2cdec85276964ff8f43bbb1d3b/Flask-1.1.1-py2.py3-no
ne-any.whl
Collecting itsdangerous>=0.24 (from flask)
  Using cached https://files.pythonhosted.org/packages/76/ae/44b03b253d6fade317f32c24d100b3b35c2239807046a4c953c7b89fa49e/itsdangerous-1.1.0-py2
.py3-none-any.whl
Collecting Jinja2>=2.10.1 (from flask)
  Using cached https://files.pythonhosted.org/packages/65/e0/eb35e762802015cab1ccee04e8a277b03f1d8e53da3ec3106882ec42558b/Jinja2-2.10.3-py2.py3-
none-any.whl
Collecting click>=5.1 (from flask)
  Using cached https://files.pythonhosted.org/packages/fa/37/45185cb5abbc30d7257104c434fe0b07e5a195a6847506c074527aa599ec/Click-7.0-py2.py3-none
-any.whl
Collecting Werkzeug>=0.15 (from flask)
  Using cached https://files.pythonhosted.org/packages/ce/42/3aeda98f96e85fd26180534d36570e4d18108d62ae36f87694b476b83d6f/Werkzeug-0.16.0-py2.py
3-none-any.whl
Collecting MarkupSafe>=0.23 (from Jinja2>=2.10.1->flask)
  Downloading https://files.pythonhosted.org/packages/98/7b/ff284bd8c80654e471b769062a9b43cc5d03e7a615048d96f4619df8d420/MarkupSafe-1.1.1-cp37-c
p37m-manylinux1_x86_64.whl
Installing collected packages: itsdangerous, MarkupSafe, Jinja2, click, Werkzeug, flask
Successfully installed Jinja2-2.10.3 MarkupSafe-1.1.1 Werkzeug-0.16.0 click-7.0 flask-1.1.1 itsdangerous-1.1.0
(flaskenv) microblog>
```

# VERIFY FLASK

```
(flaskenv) microblog>python
Python 3.7.0b3 (default, Mar 30 2018, 04:35:22)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import flask
>>> _
```



# ON THE FLASK WEBSITE

```
from flask import Flask, escape, request

app = Flask(__name__)

@app.route('/')
def hello():
    name = request.args.get("name", "World")
    return f'Hello, {escape(name)}!'
```

```
$ env FLASK_APP=hello.py flask run
* Serving Flask app "hello"
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```



# WE WILL TAKE MORE SYSTEMATIC APPROACH

- First, we create a package in which the web application lives
- In python, a directory that includes the file `__init__.py` is considered a package
- When you import the package, `__init__.py` executes and defines what symbols are exposed to the outside world

```
(flaskenv) microblog>mkdir blogapp
```

Inside the `blogapp` directory, create a file called `__init__.py`, and add the following code

```
from flask import Flask
```

```
app = Flask(__name__)
```

```
from blogapp import routes
```





# EXPLANATION

- `__name__` is a python predefined variable which is set to the module in which it is used.
- Flask uses this a starting point to load additional resources, such as templates or other additional resources
- The routes module is imported at the bottom, and not at the top (like usual Python import statements).
- This is to prevent circular imports [When we create 'routes' we shall see that it needs to use 'app' defined in this script]
- In routes.py, we will implement the *view functions*, also called application routes. These are the urls that the application will respond to



```
CREATE routes.py inside  
blogapp  
from blogapp import app  
  
@app.route('/')  
@app.route('/index')  
def index():  
    return "Hello, World!"
```

@app.route are decorators. A decorator creates an association between the URL given as argument and the function.



# CREATE THE TOP-LEVEL SCRIPT

- This script defines the Flask application instance.
- Create a file called `microblog.py` and insert one line of code

```
from blogapp import app
```

- The project should now look like

```
(flaskenv) microblog>ls
blogapp  flaskenv  microblog.py  __pycache__
```

Top-level

```
(flaskenv) microblog>cd blogapp
(flaskenv) blogapp>ls
__init__.py  __pycache__  routes.py
(flaskenv) blogapp>_
```

Inside blogapp directory



# ALL DONE!

- Set the environment variable `FLASK_APP`, so that flask knows what your application is

```
microblog>export FLASK_APP=microblog.py
```

- Then run flask!

```
(flaskenv) microblog>flask run
* Serving Flask app "microblog.py"
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

# OPTIONAL STEP

- Register environment variables that you want automatically, whenever you run the `flask` command

```
(flaskenv) microblog>pip install python-dotenv
Collecting python-dotenv
  Downloading https://files.pythonhosted.org/packages/57/c8/5b14d5cffe7bb06bedf9d66c4562bf90330d3d35e7f0266928c370d9dd6d/python_dotenv-0.10.3-py2.py3-none-any.whl
Installing collected packages: python-dotenv
Successfully installed python-dotenv-0.10.3
```

- Now, create a file called `.flaskenv` and add one line

```
FLASK_APP=microblog.py
```



# MORE COMPLEX OUTPUT

- What if we wanted to return a more complex web-page?

```
from blogapp import app

@app.route('/')
@app.route('/index')
def index():
    user = {'username': 'Vivek'}
    return '''
    <html>
    <head>
    |     <title>Home Page - Microblog</title>
    </head>
    <body>
    |     <h1>Hello, ''' + user['username'] + '''!</h1>
    </body>
    </html>'''
```

# TEMPLATES

- Returning HTML strings is ugly!!
- If we wanted to change the style of the page, it would be extremely time-consuming and error-prone
- Flask comes with a templating engine called *Jinja*
- Templates allow us to separate business logic and presentation
- In flask, templates are written as separate files, stored in a *templates* directory inside the main app directory (in our example, `blogapp`)



# CREATING A TEMPLATE

- First, create the *templates* directory

```
(flaskenv) microblog>ls
blogapp flaskenv microblog.py pycache_
(flaskenv) microblog>mkdir blogapp/templates
(flaskenv) microblog>_
```





# CREATING A TEMPLATE – II

Inside the *templates* directory, create an `index.html` file

```
<html>
  <head>
    <title>{{ title }} - Microblog</title>
  </head>
  <body>
    <h1>Hello, {{ user.username }}!</h1>
  </body>
</html>
```



# CREATING A TEMPLATE – III

- Modify the `routes.py` to use the template

```
from blogapp import app

@app.route('/')
@app.route('/index')
def index():
    user = {'username': 'Vivek'}
    return render_template('index.html', title='Home', user=user)
```



# CONDITIONAL STATEMENTS

```
<html>
  <head>
    {% if title %}
    <title>{{ title }} - Microblog</title>
    {% else %}
    <title>No title was set! Stop being lazy!</title>
    {% endif %}
  </head>
  <body>
    <h1>Hello, {{ user.username }}!</h1>
  </body>
</html>
```

# CONDITIONS – MODIFIED routes.py

```
@app.route('/if')  
def conditional():  
    user = {'username': 'Vivek'}  
    return render_template('conditional.html', user=user)
```



# LOOPS

```
<html>
  <head>
    {% if title %}
    <title>{{ title }} - Microblog</title>
    {% else %}
    <title>Welcome to Microblog</title>
    {% endif %}
  </head>
  <body>
    <h1>Hi, {{ user.username }}!</h1>
    {% for post in posts %}
    <div><p>{{ post.author.username }} says: <b>{{ post.body }}</b></p></div>
    {% endfor %}
  </body>
</html>
```

# LOOPS – MODIFIED routes.py

```
@app.route('/loop')
def posts():
    user = {'username': 'Vivek'}
    posts = [
        {
            'author': {'username': 'Abey'},
            'body': 'The Avengers movie was so cool!'
        },
        {
            'author': {'username': 'Seán'},
            'body': 'The All-Blacks win again!'
        }
    ]
    return render_template('loops.html', title='Home', user=user, posts=posts)
```

# TEMPLATES CAN BE INHERITED

- Can move parts of the template that are common to multiple pages into a base template
- Allows for consistency across pages
- Less code to write



# CREATE A base.html

```
<html>
  <head>
    {% if title %}
    <title>{{ title }} - Microblog</title>
    {% else %}
    <title>Don't be lazy! Set a title.</title>
    {% endif %}
  </head>
  <body>
    <div>Microblog: <a href="/index">Home</a></div>
    <hr>
    {% block content %}{% endblock %}
  </body>
</html>
```



# EXTEND THE TEMPLATE

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

{% block content %}
    <h1>Hi, {{ user.username }}!</h1>
    {% for post in posts %}
        <div><p>{{ post.author.username }} says: <b>{{ post.body }}</b></p></div>
    {% endfor %}
{% endblock %}
```



# TO-DO — I

- Create a venv;
- Activate the venv;
- Install Flask
- Create routes ('index' and 'sayings')
- Create a base template (base.html)
- Create two templates that inherit from base called 'index.html' and 'sayings.html'
- The base.html should look like the code shown in class
- The index.html should have a welcome message, customized to the user [as shown in class]



# TO-DO – II

- The `sayings.html` should iterate through *two* lists of strings and join them with the phrase “said:”.

E.g., `list = ["John", "Mary", "Abey"]`

`list2 = ["Good Morning", "Good Riddance", "I hate python"]`

**The output should be an HTML list like:**

1. John said Good Morning
2. Mary said Good Riddance
3. Abey said I hate Python

