An Introduction to Web Development with Flask

Prudhvi Boyapalli

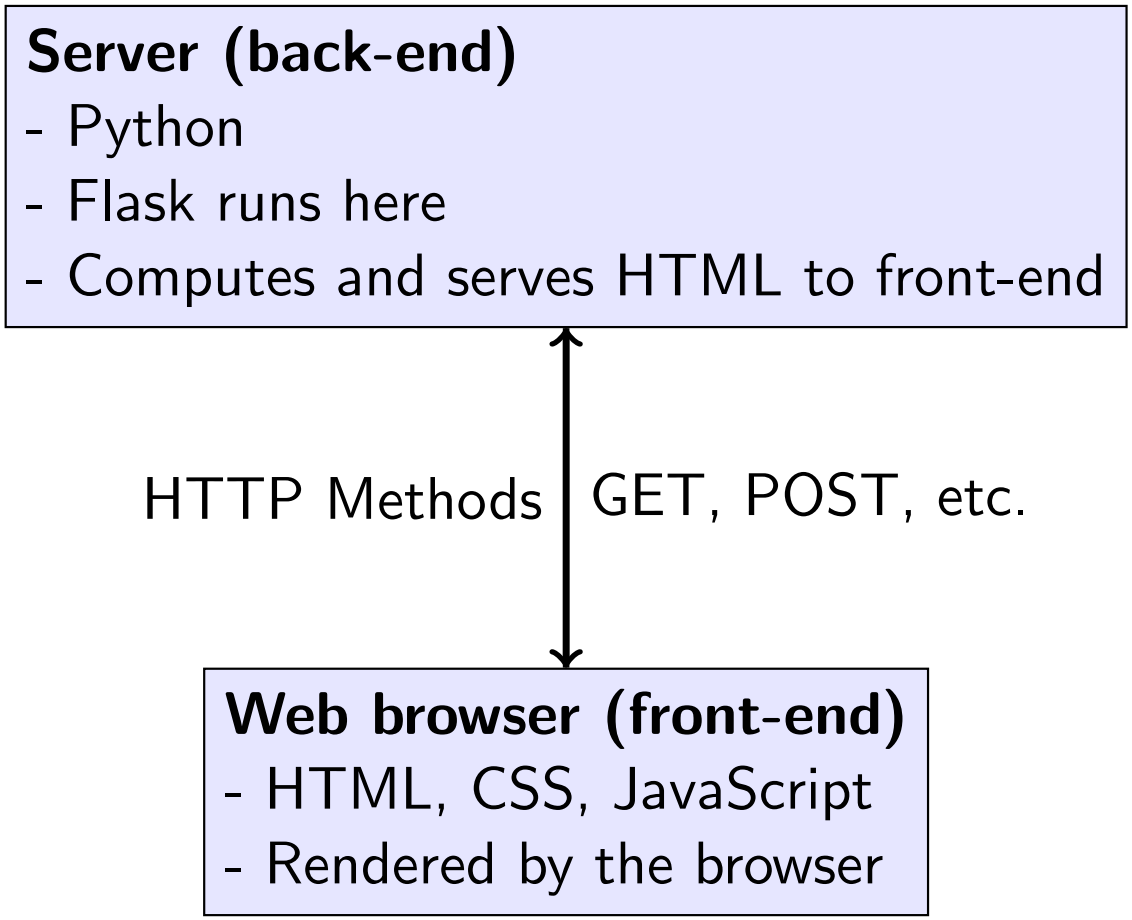
Rice Computer Science Club
Hack & Learn Workshop
January 13, 2017

Introduction
Flask
Pre-reqs
Setup
Hello World
Basic Example
Closer Look
Variable Routing
List Sort
Demo
Structure
Communication
Putting It All Together
Deep Dive
Conclusion
Summary
Resources
Contact

Introduction

How does a web app work? What does Flask do?

Flask is a Python microframework for building web applications.



Necessary prior knowledge

- Basic familiarity with Python
  - If you've taken COMP140 you're in good shape
- Basic familiarity with HTML and JavaScript, CSS
  - Be able to read HTML and JavaScript syntax
  - CSS is optional

Necessary prior knowledge

- Basic familiarity with Python
  - If you've taken COMP140 you're in good shape
- Basic familiarity with HTML and JavaScript, CSS
  - Be able to read HTML and JavaScript syntax
  - CSS is optional

Install Python & Flask

- Install Python if you haven't already: `python.org`
  - Either Python 2.7.9+ or 3.3+
- Install Flask using PIP
  - **P**ip **I**nstalls **P**ackages from the **Py**thon **P**ackage **I**ndex a.k.a. PyPI
  - `pip` is included in the Python install

Flask Installation

```
> sudo pip install Flask
```

Editing

An IDE is overkill for this workshop. Use a programmer's editor that you're comfortable with:

- Vim (All platforms)
- Atom (All platforms)
- Visual Studio Code (All platforms)
- Sublime Text (All platforms)
- Notepad++ (Windows)
- TextMate (OSX)
- Whatever floats your boat

Debugging

You'll be testing and debugging your code from the browser, so use one with good developer tools:

- Chrome
- Firefox

Other browsers can be used, but Chrome or Firefox will minimize the chance of running into browser specific issues.

Hello World
(Open up your editor and follow along!)

Super basic hello world example

This is the bare minimum code needed on the back-end to serve a web page

./code/hello_world_example/hello_world.py

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

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

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

Running the web app

Once we've written the flask code, we can run the server locally and view the results in our web browser

Execute these commands to run the code:

```
> cd /directory/with/project
> export FLASK_APP=hello_world.py
> flask run
```

Then you'll see something like:

```
* Serving Flask app "hello_world"
* Running on http://127.0.0.1:5000/
(Press CTRL+C to quit)
```

Running the web app

Once we've written the flask code, we can run the server locally and view the results in our web browser

Execute these commands to run the code:

```
> cd /directory/with/project
> export FLASK_APP=hello_world.py
> flask run
```

Then you'll see something like:

```
* Serving Flask app "hello_world"
* Running on http://127.0.0.1:5000/
(Press CTRL+C to quit)
```

Running the web app

Once we've written the flask code, we can run the server locally and view the results in our web browser

Execute these commands to run the code:

```
> cd /directory/with/project
> export FLASK_APP=hello_world.py
> flask run
```

Then you'll see something like:

```
* Serving Flask app "hello_world"
* Running on http://127.0.0.1:5000/
(Press CTRL+C to quit)
```

How did that work?

1. We started up a local flask server which is listening at IP address 127.0.0.1, port 5000

How did that work?

1. We started up a local flask server which is listening at IP address 127.0.0.1, port 5000
2. We visited this page and the browser sent a request to our flask server

How did that work?

1. We started up a local flask server which is listening at IP address 127.0.0.1, port 5000
2. We visited this page and the browser sent a request to our flask server
3. Because we visited `127.0.0.1:5000/` (emphasis on the `"/"`), the method bound to `"/"` was invoked

How did that work?

1. We started up a local flask server which is listening at IP address 127.0.0.1, port 5000
2. We visited this page and the browser sent a request to our flask server
3. Because we visited `127.0.0.1:5000/` (emphasis on the `"/"`), the method bound to `"/"` was invoked
4. The `hello()` method in `hello_world.py` was executed and the return value is sent to the browser as HTML

How did that work?

1. We started up a local flask server which is listening at IP address 127.0.0.1, port 5000
2. We visited this page and the browser sent a request to our flask server
3. Because we visited `127.0.0.1:5000/` (emphasis on the `"/"`), the method bound to `"/"` was invoked
4. The `hello()` method in `hello_world.py` was executed and the return value is sent to the browser as HTML
5. The browser received the HTML and rendered the webpage

Interactive hello world with variable routing

Now that we have a basic idea of how flask works, we can do some cool stuff with routing:

./code/hello_world_example/hello_user.py

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

@app.route("/world")
def hello():
    return "Hello World!"

@app.route("/user/<name>")
def hello_user(name):
    return "Hello %s, how are you today?" % name

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

Running the new and improved web app

As before:

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

Hello World

`http://127.0.0.1:5000/world`

Hello <user>

`http://127.0.0.1:5000/user/Foo`
`http://127.0.0.1:5000/user/Bar`

Running the new and improved web app

As before:

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

Hello World

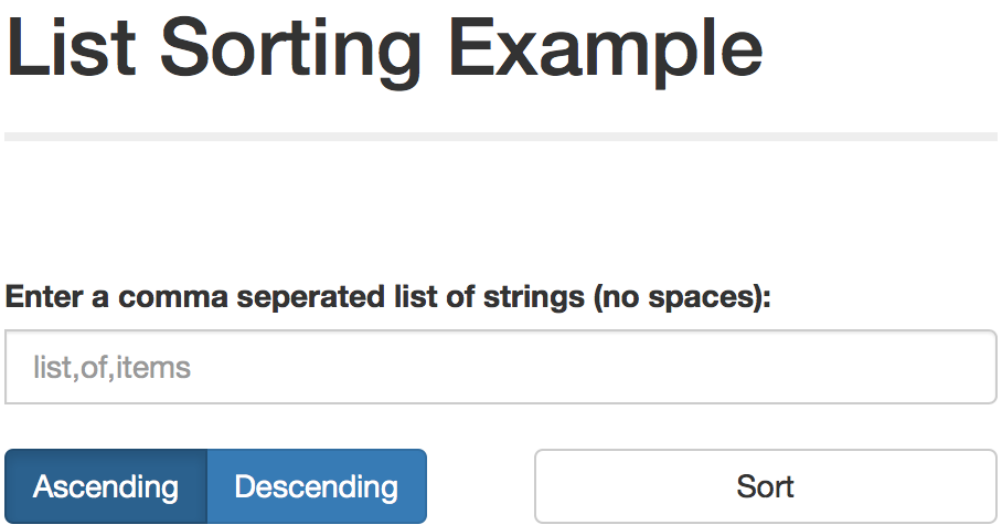
```
http://127.0.0.1:5000/world
```

Hello <user>

```
http://127.0.0.1:5000/user/Foo
http://127.0.0.1:5000/user/Bar
```

List Sort
(Don't try to code this up right now.)

Quick Demo



Project Structure

As we build more complex web applications, we need to organize our files in a sensible way and maintain compatibility with flask

List Sort Project Structure

Directories indicated in bold.

- **list_sort_example**
  - **static**
    - **scripts** (JavaScript files)
    - **styles** (CSS files)
  - **templates**
    - index.html
  - list_sort.py

Project Structure

As we build more complex web applications, we need to organize our files in a sensible way and maintain compatibility with flask

List Sort Project Structure

Directories indicated in bold.

- **list_sort_example**
  - **static**
    - **scripts** (JavaScript files)
    - **styles** (CSS files)
  - **templates**
    - index.html
  - list_sort.py

Project Structure

As we build more complex web applications, we need to organize our files in a sensible way and maintain compatibility with flask

List Sort Project Structure

Directories indicated in bold.

- **list_sort_example**
  - **static**
    - **scripts** (JavaScript files)
    - **styles** (CSS files)
  - **templates**
    - index.html
  - list_sort.py

Hypertext Transfer Protocol

The data transfer protocol used on the World Wide Web

GET The most common HTTP method, used to get files from a server. Multiple GET requests are made by your browser every time you visit a webpage to retrieve all the necessary resources needed to render that page, such as HTML files, CSS and JavaScript files, images, etc.

POST This method is used to send data to a server. Commonly used to submit form data or any other user input.

These are the most common HTTP request methods and the ones used by the demo, others exist:

`developer.mozilla.org/en-US/docs/Web/HTTP/Methods`

Hypertext Transfer Protocol

The data transfer protocol used on the World Wide Web

GET The most common HTTP method, used to get files from a server. Multiple GET requests are made by your browser every time you visit a webpage to retrieve all the necessary resources needed to render that page, such as HTML files, CSS and JavaScript files, images, etc.

POST This method is used to send data to a server. Commonly used to submit form data or any other user input.

These are the most common HTTP request methods and the ones used by the demo, others exist:

`developer.mozilla.org/en-US/docs/Web/HTTP/Methods`

Hypertext Transfer Protocol

The data transfer protocol used on the World Wide Web

GET The most common HTTP method, used to get files from a server. Multiple GET requests are made by your browser every time you visit a webpage to retrieve all the necessary resources needed to render that page, such as HTML files, CSS and JavaScript files, images, etc.

POST This method is used to send data to a server. Commonly used to submit form data or any other user input.

These are the most common HTTP request methods and the ones used by the demo, others exist:

`developer.mozilla.org/en-US/docs/Web/HTTP/Methods`

Putting It All Together (part 1)

Retrieving resources and rendering the webpage

1. User enters the address into their browser

Putting It All Together (part 1)

Retrieving resources and rendering the webpage

1. User enters the address into their browser
2. Browser sends a `GET` request to the address, in this case `127.0.0.1:5000/`

Putting It All Together (part 1)

Retrieving resources and rendering the webpage

1. User enters the address into their browser
2. Browser sends a `GET` request to the address, in this case `127.0.0.1:5000/`
3. The flask method bound to `"/"` is invoked and returns `index.html`

Putting It All Together (part 1)

Retrieving resources and rendering the webpage

1. User enters the address into their browser
2. Browser sends a `GET` request to the address, in this case `127.0.0.1:5000/`
3. The flask method bound to `"/"` is invoked and returns `index.html`
4. Browser processes the HTML file, which instructs it to load CSS and JavaScript files

Putting It All Together (part 1)

Retrieving resources and rendering the webpage

1. User enters the address into their browser
2. Browser sends a `GET` request to the address, in this case `127.0.0.1:5000/`
3. The flask method bound to `"/"` is invoked and returns `index.html`
4. Browser processes the HTML file, which instructs it to load CSS and JavaScript files
5. Browser sends additional `GET` requests to the location indicated within the HTML files

Putting It All Together (part 1)

Retrieving resources and rendering the webpage

1. User enters the address into their browser
2. Browser sends a `GET` request to the address, in this case `127.0.0.1:5000/`
3. The flask method bound to `"/"` is invoked and returns `index.html`
4. Browser processes the HTML file, which instructs it to load CSS and JavaScript files
5. Browser sends additional `GET` requests to the location indicated within the HTML files
6. Flask serves the static CSS and JavaScript files

Putting It All Together (part 1)

Retrieving resources and rendering the webpage

1. User enters the address into their browser
2. Browser sends a `GET` request to the address, in this case `127.0.0.1:5000/`
3. The flask method bound to `"/"` is invoked and returns `index.html`
4. Browser processes the HTML file, which instructs it to load CSS and JavaScript files
5. Browser sends additional `GET` requests to the location indicated within the HTML files
6. Flask serves the static CSS and JavaScript files
7. Browser now has all necessary files and renders the webpage

Putting It All Together (part 1)

Retrieving resources and rendering the webpage

1. User enters the address into their browser
2. Browser sends a `GET` request to the address, in this case `127.0.0.1:5000/`
3. The flask method bound to `"/"` is invoked and returns `index.html`
4. Browser processes the HTML file, which instructs it to load CSS and JavaScript files
5. Browser sends additional `GET` requests to the location indicated within the HTML files
6. Flask serves the static CSS and JavaScript files
7. Browser now has all necessary files and renders the webpage

This all happens within milliseconds!

Putting It All Together (part 2)

Send data to the server, back-end computation, return results

1. User enters information into the input fields and clicks the `Sort` button

Putting It All Together (part 2)

Send data to the server, back-end computation, return results

1. User enters information into the input fields and clicks the `Sort` button
2. The JavaScript function bound to that button is invoked

Putting It All Together (part 2)

Send data to the server, back-end computation, return results

1. User enters information into the input fields and clicks the `Sort` button
2. The JavaScript function bound to that button is invoked
3. The input information is read from the input fields and stored in an object (**J**ava**S**cript **O**bject **N**otation)

Putting It All Together (part 2)

Send data to the server, back-end computation, return results

1. User enters information into the input fields and clicks the `Sort` button
2. The JavaScript function bound to that button is invoked
3. The input information is read from the input fields and stored in an object (**J**ava**S**cript **O**bject **N**otation)
4. A `PUT` request is created and sent to `"/sort"`

Putting It All Together (part 2)

Send data to the server, back-end computation, return results

1. User enters information into the input fields and clicks the `Sort` button
2. The JavaScript function bound to that button is invoked
3. The input information is read from the input fields and stored in an object (**J**ava**S**cript **O**bject **N**otation)
4. A `PUT` request is created and sent to `"/sort"`
5. Flask server receives the request
    1. The method bound to `"/sort"` is invoked

Putting It All Together (part 2)

Send data to the server, back-end computation, return results

1. User enters information into the input fields and clicks the `Sort` button
2. The JavaScript function bound to that button is invoked
3. The input information is read from the input fields and stored in an object (**J**ava**S**cript **O**bject **N**otation)
4. A `PUT` request is created and sent to `"/sort"`
5. Flask server receives the request
    1. The method bound to `"/sort"` is invoked
    2. The data from the request object is accessed

Putting It All Together (part 2)

Send data to the server, back-end computation, return results

1. User enters information into the input fields and clicks the `Sort` button
2. The JavaScript function bound to that button is invoked
3. The input information is read from the input fields and stored in an object (**J**ava**S**cript **O**bject **N**otation)
4. A `PUT` request is created and sent to `"/sort"`
5. Flask server receives the request
    1. The method bound to `"/sort"` is invoked
    2. The data from the request object is accessed
    3. The list is sorted

Putting It All Together (part 2)

Send data to the server, back-end computation, return results

1. User enters information into the input fields and clicks the `Sort` button
2. The JavaScript function bound to that button is invoked
3. The input information is read from the input fields and stored in an object (**J**ava**S**cript **O**bject **N**otation)
4. A `PUT` request is created and sent to `"/sort"`
5. Flask server receives the request
    1. The method bound to `"/sort"` is invoked
    2. The data from the request object is accessed
    3. The list is sorted
    4. A response object is created with the results in JSON format and returned

Putting It All Together (part 2)

Send data to the server, back-end computation, return results

1. User enters information into the input fields and clicks the `Sort` button
2. The JavaScript function bound to that button is invoked
3. The input information is read from the input fields and stored in an object (**J**ava**S**cript **O**bject **N**otation)
4. A `PUT` request is created and sent to `"/sort"`
5. Flask server receives the request
    1. The method bound to `"/sort"` is invoked
    2. The data from the request object is accessed
    3. The list is sorted
    4. A response object is created with the results in JSON format and returned
6. Back in the JavaScript function, the response is parsed

Putting It All Together (part 2)

Send data to the server, back-end computation, return results

1. User enters information into the input fields and clicks the `Sort` button
2. The JavaScript function bound to that button is invoked
3. The input information is read from the input fields and stored in an object (**J**ava**S**cript **O**bject **N**otation)
4. A `PUT` request is created and sent to `"/sort"`
5. Flask server receives the request
    1. The method bound to `"/sort"` is invoked
    2. The data from the request object is accessed
    3. The list is sorted
    4. A response object is created with the results in JSON format and returned
6. Back in the JavaScript function, the response is parsed
7. The necessary HTML to display the sorted list is created and inserted into page

Deep Dive

Conclusion

Topics Covered

- Setting up a development environment for Flask
- Basic & and variable routing
- `GET` and `POST` requests
- Accessing request data
- Overview of in-browser developer tools
- Building and serving a "complex" web app with HTML, CSS, JavaScript and Python

Next Steps

More Flask Features

- URL Building
- Additional HTTP Methods
- HTML templates with placeholders, complex templating language
- File uploads
- Reading and writing cookies
- Redirects and error handling
- Managing sessions
- And lots of other stuff

The Flask website and documentation is really good! Lots of great tutorials exist, just search around.

Deploying your web app

Once you've developed and tested you app locally, you can deploy your code and make it publicly accessible. There lots of tutorials on how to do this for various cloud providers (AWS, Azure, DigitalOcean, etc).

Resources

- **This presentation & and all of the code**
  `github.com/hack-rice/flask-tutorial`
- **Flask website & awesome documentation**
  `flask.pocoo.org`

Contact

Prudhvi Boyapalli

- prb2 AT rice.edu

HackRice

- Fall 2017
- Annual hackathon right here on campus
- Great way to hone your skills and make something cool

RiceApps

- Work on a team of students to build useful applications
- Great way to gain hands on development experience
- `http://riceapps.org`

Rice CS Facebook Group

- `https://facebook.com/groups/365985373415471`

CS Club Facebook Page

- `https://facebook.com/ricecsclub`

Contact

Prudhvi Boyapalli

- prb2 AT rice.edu

HackRice

- Fall 2017
- Annual hackathon right here on campus
- Great way to hone your skills and make something cool

RiceApps

- Work on a team of students to build useful applications
- Great way to gain hands on development experience
- `http://riceapps.org`

Rice CS Facebook Group

- `https://facebook.com/groups/365985373415471`

CS Club Facebook Page

- `https://facebook.com/ricecsclub`

Contact

Prudhvi Boyapalli

- prb2 AT rice.edu

HackRice

- Fall 2017
- Annual hackathon right here on campus
- Great way to hone your skills and make something cool

RiceApps

- Work on a team of students to build useful applications
- Great way to gain hands on development experience
- `http://riceapps.org`

Rice CS Facebook Group

- `https://facebook.com/groups/365985373415471`

CS Club Facebook Page

- `https://facebook.com/ricecsclub`

Contact

Prudhvi Boyapalli

- prb2 AT rice.edu

HackRice

- Fall 2017
- Annual hackathon right here on campus
- Great way to hone your skills and make something cool

RiceApps

- Work on a team of students to build useful applications
- Great way to gain hands on development experience
- `http://riceapps.org`

Rice CS Facebook Group

- `https://facebook.com/groups/365985373415471`

CS Club Facebook Page

- `https://facebook.com/ricecsclub`