INTRODUCING FLASK

Dhruv Baldawa (@dhruvbaldawa)

WHAT IS FLASK ?

In layman terms, something which helps you make web sites/services/applications (*think Django, but simpler and smaller*)

- “micro”framework based on **Werkzeug** and **Jinja2**
- simple and extensible core
- gives you the bare minimum to get started with development

SNEAK PEEK

```python
# hello.py - Your first flask app !
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World!'

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

Yes, thats it !!

```
$ python hello.py
* Running on http://127.0.0.1:5000/
```

WARNING !

CODE EXAMPLES AHEAD

FEEL FREE TO INTERRUPT

Tip: look for the ***emphasized part*** in the code

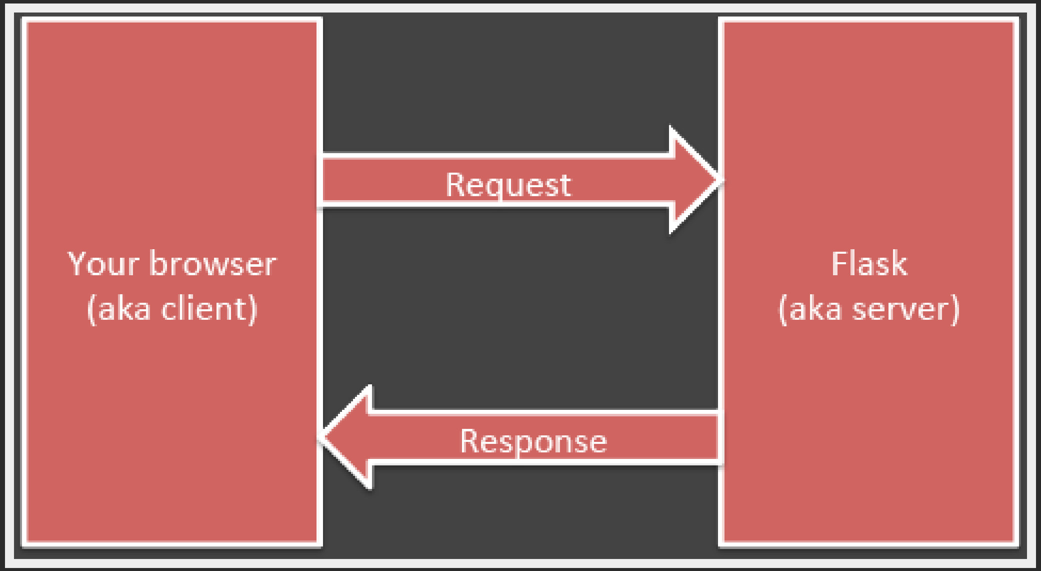
Press ***Space*** for next and ***Backspace*** for previous slide.

Pressing ***ESC*** does some magic. :-)

Just before we start, lets go through some

HTTP BASICS

WORKING OF HTTP



Client sends a **REQUEST** to the server
Servers sends a **RESPONSE** back to the client

HTTP METHODS

1. GET -> retrieve (HTTP 1.0)
2. POST -> submit (HTTP 1.0)
3. PUT -> update (HTTP 1.1)
4. DELETE -> remove (HTTP 1.1)
5. and some more..

HTTP STATUSES

1. 2xx – Success
    1. 200 – OK
    2. 201 – Object created
2. 3xx – Redirection
3. 4xx – Client Error
    1. 400 – Bad request
    2. 401 – Unauthorized
    3. 403 – Forbidden
    4. 404 – Not found
4. 5xx – Server Error
    1. 500 – Internal Server Error

WHY SHOULD I KNOW THIS ?

```
$ python app.py
127.0.0.1 - - GET / HTTP/1.1 200 -
127.0.0.1 - - GET /favicon.ico HTTP/1.1 404 -
127.0.0.1 - - GET / HTTP/1.1" 404 -
127.0.0.1 - - POST / HTTP/1.1 400 -
```

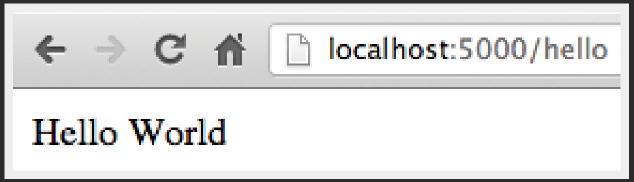
WHAT WE WILL COVER ?

- Routing
- Requests
- Response
- Templates

ROUTING

```
@app.route('/hello') # <- This is routing !
def hello(): # <- so, this will be called a view
    ''' execute the following code when the user visits /hello '''
    return 'Hello World'
```

- The `route()` decorator "binds" a function to a particular URL.



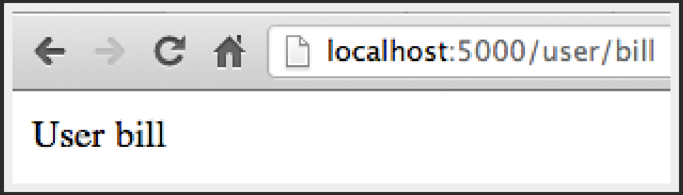
Hello World

This is what that does !

VARIABLE RULES

```
@app.route('/user/<username>')
def show_user_profile(username):
    # show the user profile for that user
    return 'User %s' % username

@app.route('/post/<int:post_id>')
def show_post(post_id):
    # show the post with the given id, the id is an integer
    return 'Post %d' % post_id
```



User bill

VARIABLE RULES

CONVERTERS

default	string with no slashes allowed
int	accepts integers
float	like int but accepts floating point numbers
path	like the default but also accepts slashes

VARIABLE RULES

TO SUMMARIZE:

- To add variable parts to a URL you can mark these special sections as `<variable_name>`
- Such a part is then passed as **keyword argument** to your function
- Optionally a converter can be specified by specifying a rule with `<converter:variable_name>`

USING HTTP METHODS

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    ''' This function only works for the GET and POST methods '''
    if request.method == 'POST':
        # this is executed in case of POST request
        do_the_login()
    else:
        # this is executed in case of GET request
        show_the_login_form()
```

- The supported methods are *GET, POST, PUT, DELETE, HEAD, OPTIONS.*
- If GET is present, HEAD will automatically be added.
- OPTIONS is used to check what methods are available for a particular URL.

URL BUILDING

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

@app.route('/')
def index(): pass

@app.route('/login')
def login(): pass

@app.route('/user/<username>)
def profile(username): pass

with app.test_request_context(): # lets ignore this for time being
    print url_for('index')                    # prints /
    print url_for('login')                    # prints /login
    print url_for('login', next='/')          # prints /login?next=
/
    print url_for('profile', username='Jack')# prints /user/Jack
```

URL BUILDING

TO SUMMARIZE:

- `url_for()` accepts the function name and some keyword arguments, corresponding to the "variable" part of the URL rule.
- Unknown variable parts are appended to the URL as query parameters.
- URL building will handle escaping of special characters and Unicode data transparently for you.

SURPRISE !

THIS IS NOT THE ONLY WAY TO CREATE VIEWS AND ROUTES !

REQUESTS

```
from flask import request # The request object
```

- As far as the request objects are considered, "its complicated".
- Its a global object, which helps you access the request data.
- Lets you access the URL, POST/PUT, cookies, file upload data.

REQUESTS

ACCESSING REQUEST DATA

Method	Attribute
GET	`request.args`
POST/PUT	`request.form`
Cookie	`request.cookies`
File	`request.files`
GET + POST/PUT	`request.values`

REQUESTS

ACCESSING GET DATA

```
with app.test_request_context('/login?a=1&a=2&b=3'):
    print request.args.get('a')      # prints '1'
    print request.args['a']          # prints '1'
    print request.args.getlist('a') # prints ['1', '2']
```

ACCESSING POST DATA

```
@app.route('/login', methods=['POST', 'GET'])
def login():
    error = None
    if request.method == 'POST':
        if valid_login(request.form['username'],
                       request.form['password']):
            return log_the_user_in(request.form['username'])
        else:
            return 'Invalid username/password'
```

REQUESTS

ACCESSING FILE DATA

```
@app.route('/upload', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        f = request.files['the_file']
        f.save('/var/www/uploads/uploaded_file.txt')
...
```

USING COOKIES

```
# Accessing cookies
def index():
    username = request.cookies.get('username')

# Setting cookies
def index():
    resp = make_response(render_template(...))
    resp.set_cookie('username', 'the username')
    return resp
```

REQUESTS

SESSIONS

```python
from flask import session, request
def fresh_login():
    if 'username' in session:
        # removing session data
        session.pop('username', None)
    else:
        # set session data
        session['username'] = request.form['username']

# IMPORTANT! set the secret key. keep this really secret:
app.secret_key = 'ASDSgjhJhHuoOuY7786689'
```

RESPONSE

- The view function returns a response object.
- Remember this tuple, `(response, status, headers)`
- `make_response()` can be used to create response objects
- `make_response()` can be used as `make_response(body, status, headers)`
- Use `render_template()` to return HTML templates

RESPONSE

RETURN VALUES TO RESPONSE OBJECTS

Return Type	Conversion
response	returned as is
string	returned with status 200, and mimetype text/html
tuple	taken as form (body, status, headers) with atleast one element
none	assume a valid WSGI app, and convert it to a response object

RESPONSE

```python
def not_found(error):
    return render_template('error.html'), 404 # The template data
with status 404 is returned.

def not_found(error):
    ''' not_found using make_response() '''
    resp = make_response(render_template('error.html'), 404)
    resp.headers['X-Something'] = 'A value' # lets not worry about
 headers
    return resp
```

RESPONSE

SPECIAL G OBJECT

- This object stores information for one request only and is available from within each function.
- Never store such things on other objects because this would not work with threaded environments.
- That special g object does some magic behind the scenes to ensure it does the right thing.

RESPONSE

SPECIAL G OBJECT

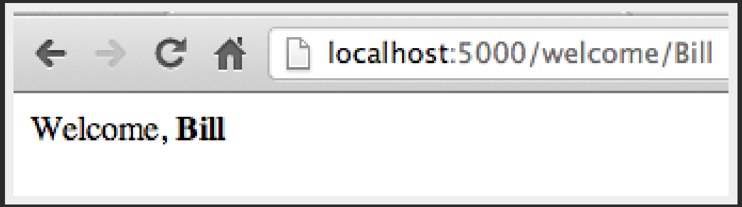
```
from flask import g
@app.route("/beer"):
def beer():
    get_beer()
    return g.beer

def get_beer():
    if session['user_age'] <= 21:
        g.beer = 'Fruit Beer'
    else:
        g.beer = get_user_ordered_beer()
```

TEMPLATES

```
@app.route("/welcome/<user>")
def welcome_user(user):
    return render_template('welcome.html', user=user)
```

```
# welcome.html
Welcome, <strong>{{ user }}</strong>
```


localhost:5000/welcome/Bill

Welcome, **Bill**

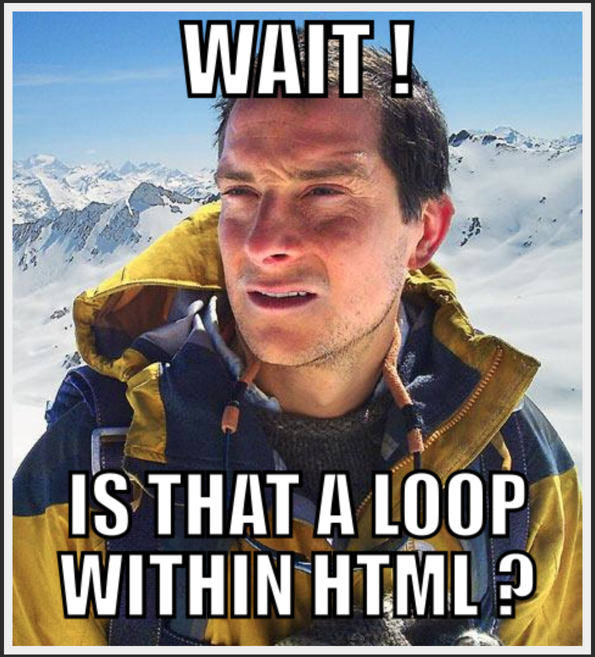
TEMPLATES

- Flask uses Jinja2 for templating.
- You have to use `render_template('template')` for using templates
- Templates are stored in templates/ directory
- You can pass data variables as keyword arguments

JINJA

SNEAK PEEK

```
<ul id="navigation">
{% for item in navigation %}
    <li><a href="{{ item.href }}">{{ item['caption'] }}</a></li>
{% endfor %}
</ul>
```



JINJA

WHAT HAPPENS WHEN YOU DO `foo.bar`?

- check if there is an *attribute* called `bar` on `foo`.
- if there is not, check if there is an *item* `'bar'` in `foo`.
- if there is not, return an undefined object.

WHAT HAPPENS WHEN YOU DO `foo['bar']`?

- if there is not, check if there is an *item* `'bar'` in `foo`.
- check if there is an *attribute* called `bar` on `foo`.
- if there is not, return an undefined object.

JINJA

SOME EXAMPLES

```
{% if kenny.sick %}
    Kenny is sick.
{% elif kenny.dead %}
    You killed Kenny!  You bastard!!!
{% else %}
    Kenny looks okay --- so far
{% endif %}
```

```
{% for item in iterable|sort %}
    ...
{% endfor %}
```

JINJA

GLOBAL VARIABLES

config	g
request	session
url_for()	get_flashed_messages()

WE HAVE COVERED FLASK AND JINJA BASICS TO GET YOU STARTED.

BUT, THERE IS MORE TO IT

WHICH CAN BE A PART OF SOME OTHER DISCUSSION.

THANK YOU

REFERENCES

- Flask documentation
- Jinja2 documentation
- HTTP – Wikipedia
- HTTP Status Codes – Wikipedia