Indice

Install

```
pip3 install Flask
```

Example

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello(): return 'Hello, world!'

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

main Get for get index resource:

```
@app.route('/')
    def hello(): ....
```

Development server options for app run

```
app.run('0.0.0.0')        or
app.run('0.0.0.0' , 8080) or $ python app.py runserver -t 0.0.0.0 -p 8080 or
app.run(debug=True)
```

From Flask to Manager

using: pip3 install Flask-Script

or: pip3 install -r requirements.txt

```
from flask.ext.script import Manager
from main             import app

manager = Manager(app)

app.config['DEBUG'] = True

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

Routes

i suppose is possible also to use regex searching on get string

```
@app.route('/appointments/<int:appointment_id>/')
def appointment_detail(appointment_id):
    return 'Detail of appointment {}.'.format(appointment_id)
```

another filtering for GET and POST

- @app.route('/appointments/..../', methods=['GET', 'POST'])
- @app.route('/appointments/..../', methods=['DELETE'])

Variables

```
from flask import url_for

@app.route('/appointments/<int:appointment_id>/')
def appointment_detail(appointment_id):
    edit_url = url_for('appointment_edit', appointment_id=appointment_id)
    # Return the URL string just for demonstration.
    return edit_url

@app.route( '/appointments/<int:appointment_id>/', endpoint='some_name')
def appointment_detail(appointment_id):
    # Use url_for('some_name', appointment_id=x)
    # to build a URL for this.
    return 'Just to demonstrate...
```

about int in appointment_id

- **int**: The value of this converter is an integer
- **float**: The value of this converter is a floating point number
- **path** :
  The value of this converter is a string such as the default,
  but also accepts slashes

Http request methods:

- **PUT**: This option is like POST, but repeat PUT calls on a resource should have no effect
- **DELETE**: Using this option removes the resource
- **HEAD**: This option is like GET, but replies only with HTTP headers and not content
- **OPTIONS**: This option is used to determine which methods are available for resource

---

Automation

- *Flask implements HEAD for you if GET is present.*
- *OPTIONS for you in all cases.*
- *app.add_url_rule* is +- like app.route but built as a **function**
- **Route collisions**:
  /path:foopath/ == /foo/bar/baz/
- when index (app.py) changes , flask make an automatic live update (on started server)
- All app.route with special arguments (like /path:foopath/) work only then the argument is captured by the next function (def function (...) :)
- Each requests can be captured only one time (this make matching similar to switch)

Requests and Responses

request

- files: This feature of the request object specifies the dict of file uploads from:
  POST or PUT requests, which go here instead of request.form, for example, {}.
  Each value in the dict is a FileStorage object which behaves like a Python file object,
  but also includes a save(filepath) method to store uploaded files (after you
  validate the destination path).
- is_xhr: True: This feature of the request object specifies when
  the incoming request is a JavaScript XMLHttpRequest, and False otherwise.
  This works with JavaScript libraries that provide the X-Requested-With HTTP header,
  set to XMLHttpRequest.

response

- string1: return 'Hello, world!', 200, {'Content-Type': 'text/plain'}
- string2: return make_response('Hello, world!', status=200, headers=headers)
- page_html1: send_from_directory ('path' , 'page.html')

interesting case page not found html (code):

```
@app.errorhandler(404)
def error_not_found(error):
    return send_from_directory ('error' , '404.html') , 404
```

- redirect: return redirect (url_for('...'))

Hosting

```
@app.route ( '/static/<path:filename>' )
def assets ( filename ):
    return send_from_directory ( 'static' , filename )
```

Plugins

- Python
  - Database (mongodb / sql / postgres)
  - File uploads (to study ftp and other protocols)
- Nodejs
  - Only remote shell => Commander
  - Cloud controll => Cloud Commander

ErrorsHandling

```
* class SchedulingException(Exception):
      """Some application-specific error has occurred."""
      pass

* @app.errorhandler(SchedulingException)
  def scheduling_exception_handler(error):
      return 'Just a demo of app.errorhandler...', 500

* @app.route('/test_error/')
  def test_error():
      raise SchedulingException('Use my custom error handler.')
```

- pass == (continue or return 0) ?
- raise: The raise keyword is used to raise an exception.

Sessions

- Flask provides a session object, which behaves like a Python dictionary, and persists automatically across requests. You can, in your Flask application code: from flask import session

```
from flask import session
# ... in a request ...
session['spam'] = 'eggs'
# ... in another request ...
spam = session.get('spam') # 'eggs'
```