

Python for Practice



Web applications

Overview



- WSGI
 - Web Server Gateway Interface
 - a standard interface between web servers and Python web application frameworks
 - WSGI makes it possible to write portable Python web code that can be deployed in any WSGI-compliant web server
- Python web-frameworks
 - Django
 - a “batteries included” web application framework
 - for complex, database-backed web applications quickly
 - Flask
 - a “microframework” for Python
 - for building smaller applications, APIs, and web services
 - ...

Web applications (common structure)



- URL Routing
 - incoming request to a particular piece of code
- Request and Response Objects
 - Information from a user browser to object(s)
 - Object(s) to user browser
- Template Engine
 - separating logic (code) from presentation
- Web Server

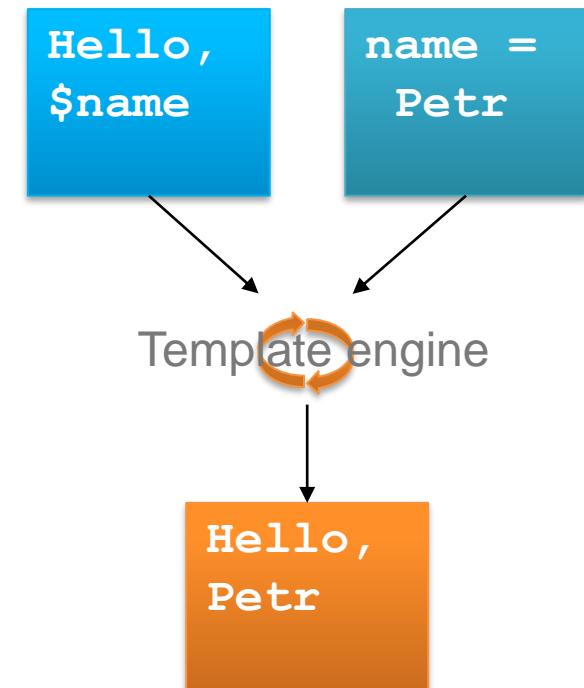
Template engines

Overview



- Separating logic from presentation
 - “user interface” can be developed separately
 - the same logic for different views
 - no need for “escaping” in code
 -

- General overview
 - template + data => final document
- Template engines in Python
 - Jinja2
 - Mako
 - Chameleon
 - ...



Jinja2



- <https://palletsprojects.com/p/jinja/>
- General usage

```
from jinja2 import Template
template = Template('Hello {{ name }}!')
template.render(name='John Doe')
```

See
e01_jinja.py

Jinja2



- Environment
 - configuration for the engine
 - if not created manually, it creates automatically

```
env = Environment(  
    loader=PackageLoader('yourapplication',  
                         'templates'),  
    autoescape=select_autoescape(['html',  
                                 'xml']))  
)
```

- methods
 - get_template(name)
 - list_templates()
 - ...

Jinja2



- Loaders
 - FileSystemLoader(searchpath)
 - PackageLoader(package_name, package_path)
 - DictLoader(mapping)
 - ChoiceLoader(loaders)
 - ...
 - BaseLoader

Jinja2



- Template
 - render(context)
 - generate(context)
 - render_async(context)
 - ...

Jinja2



- Template language

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>My Webpage</title>
</head>
<body>
    <ul id="navigation">
        {% for item in navigation %}
            <li><a href="{{ item.href }}">
                {{ item.caption }}</a></li>
        {% endfor %}
    </ul>

    <h1>My Webpage</h1>
    {{ a_variable }}

    #{ a comment #}
</body>
</html>
```

Jinja2



- Template language
 - `{% ... %}` statements
 - `{{ ... }}` expressions
 - `{# ... #}` comments
 - `# ...` line statements

Jinja2



- Template inheritance

```
<!DOCTYPE html>
<html lang="en">
<head>
    {% block head %}
    <link rel="stylesheet" href="style.css" />
    <title>{% block title %}{% endblock %} -
                                         My Webpage</title>
    {% endblock %}
</head>
<body>
    <div id="content">{% block content %}{% endblock %}</div>
    <div id="footer">
        {% block footer %}
        &copy; Copyright 2008 by <a
                                         href="http://domain.invalid/">you</a>.
        {% endblock %}
    </div>
</body>
</html>
```

Jinja2



- Template inheritance

```
{% extends "base.html" %}  
{% block title %}Index{% endblock %}  
{% block head %}  
    {{ super() }}  
    <style type="text/css">  
        .important { color: #336699; }  
    </style>  
{% endblock %}  
{% block content %}  
    <h1>Index</h1>  
    <p class="important">  
        Welcome to my awesome homepage.  
    </p>  
{% endblock %}
```

Jinja2



- Template language statements

```
{% for row in rows %}  
    <li class="{{ loop.cycle('odd', 'even') }}">  
        {{ row }}</li>  
{% endfor %}  
  
{% if kenny.sick %}  
    Kenny is sick.  
{% elif kenny.dead %}  
    You killed Kenny! You bastard!!!  
{% else %}  
    Kenny looks okay --- so far  
{% endif %}  
  
{% macro input(name, value='', type='text', size=20) -%}  
    <input type="{{ type }}" name="{{ name }}" value="{{  
        value|e }}" size="{{ size }}">  
{%- endmacro %}  
<p>{{ input('username') }}</p>
```



Flask

Overview

- <https://palletsprojects.com/p/flask/>

```
from flask import Flask, escape, request

app = Flask(__name__)

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

See
e02_hello.py

```
$ env FLASK_APP=e02_hello.py flask run
```

Deployment



- CGI

```
from wsgiref.handlers import CGIHandler  
from yourapplication import app  
  
CGIHandler().run(app)
```

- FastCGI

```
#!/usr/bin/python  
from flup.server.fcgi import WSGIServer  
from yourapplication import app  
  
if __name__ == '__main__':  
    WSGIServer(app).run()
```

- mod_wsgi (Apache)

- ...

Routing



```
@app.route('/')
def index():
    return 'Index Page'

@app.route('/hello')
def hello():
    return 'Hello, World'
```

Routing

```
@app.route('/user/<username>')
def show_user_profile(username):
    return 'User %s' % escape(username)

@app.route('/post/<int:post_id>')
def show_post(post_id):
    return 'Post %d' % post_id

@app.route('/path/<path:subpath>')
def show_subpath(subpath):
    return 'Subpath %s' % escape(subpath)
```

- string (default) accepts any text without a slash
- int accepts positive integers
- float accepts positive floating point values
- path like string but also accepts slashes
- uuid accepts UUID strings

Routing



- Building URLs

```
@app.route('/')
def index():
    return 'index'

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

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

with app.test_request_context():
    print(url_for('index'))
    print(url_for('login'))
    print(url_for('login', next='/'))
    print(url_for('profile', username='John Doe'))
```

HTTP method



```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        return do_the_login()
    else:
        return show_the_login_form()
```

Static files and templates



- Static files

```
url_for('static', filename='style.css')
```

- have to be stored in static/ subdirectory

- Templates

- Jinja2 preconfigured
- directory templates/

See
e03_template.py

```
@app.route('/hello/')
@app.route('/hello/<name>')
def hello(name=None):
    return render_template('hello.html', name=name)
```

Important objects

- **request**
 - represents request for a browser
 - fields
 - path
 - full_path
 - script_root
 - url
 - base_url
 - url_root
 - host
 - args
 - form
 - method
 - scheme
 - cookies
 - ...

See
e04_request.py

```
@app.route('/login')
def login():
    u = request.form['username'],
    p = request.form['password']):
    ...
    ...
```

File uploads



```
@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')
```

- HTML form must have the
enctype="multipart/form-data" attribute

Errors and redirects



```
@app.route('/')
def index():
    return redirect(url_for('login'))

@app.route('/login')
def login():
    abort(401)
    this_is_never_executed()

@app.errorhandler(404)
def page_not_found(error):
    return render_template(
        'page_not_found.html'), 404
```

HTTP error code
(default 200)

See
e05_redirect.py

Response



```
@app.route('/')
def index():
    resp = make_response(render_template(...))
    return resp
```

- returning
 - string -> default response
 - dict -> json

See
e06_response.py

Assignment



- Create a weather service
 - methods for obtaining
 - a list of cities
 - weather for a city
 - methods return json

Sessions



- session object
 - to store information specific to a user from one request to the next
 - implemented via cookies

See
e07_session.py

Extensions



- Huge amount of extensions
 - search on PyPI

Assignment



- Create a web app for managing users on a unix system (i.e., an app that manipulates /etc/passwd)
 - create own copy of the passwd file
 - operations for listing, adding and removing users
- Update the app to require a login
 - session needed

