Python virtual environment

Python virtual environment → isolation mechanism for python packages

- https://docs.python.org/3/tutorial/venv.html
- https://docs.python.org/3/library/venv.html
- https://docs.python.org/2.7/installing/index.html?highlight=virtualenv

In older verions of Python the package is the **`virtualenv`** external library (e.g., `python[3]-virtualenv` library in Ubuntu and Fedora)

```
python[2] -m virtualenv <directory>
```

From Python 3.4 the package is **`venv`**, included in the standard library

```
python3 -m venv <directory>
```

We set the due local environment by using the activate script

```
source <directory>/bin/activate
```

Dependencies

- The base tool for managing Python dependencies is **`pip`**

  `pip install <package-name>`

  - If executed in the ***virtual environment*** installs packages in the **`venv`** directory (does not affect system libraries)
- Accept a list of dependencies from a file

  `pip install -r <file>`
- <file> is a list of packages with optional constraints on versions

  Example:

  flask==0.12.2

  requests<=2.18.4,>2.10

Distributing Python applications

- Python integrates the **`distutils`** library for easier packaging
  - https://docs.python.org/3/library/distutils.html
  - https://docs.python.org/2/distutils/introduction.html
- Directory structure:

  `<project-directory>`
  - `<package-directory>`
  - `requirements.txt`
  - `[venv]`
  - `[setup.py]`
  - `[MANIFEST.in, REAME.txt, … ]`

  } we do not discuss these here

Flask

- Lightweight Python library for Web applications
- Integrate with additional libraries for complex behaviors

  Examples:

  - **wtforms** → forms management
  - **flask-restful** → RESTful APIs

Hello World Flask application

Create a Python module in your library with the following content

```
# -*- encoding: utf8 -*-
from flask import Flask

app = Flask(__name__)
app.config.from_object(__name__)
DEBUG = True

@app.route('/', methods=['GET'])
def helloworld():
   return 'Hello!'
```

Run the flask development server as

```
export FLASK_APP=<python-module>
flask run
```

- **Note:** app is a **WSGI-compliant** Python object

Templates and Jinja2

- How to efficiently build HTML Web pages?
  - using Python strings facilities (e.g. *format)* would be a huge pain
- Web templating libraries → **Jinja2**
  - Python-like language to write templates
    - http://jinja.pocoo.org/docs/2.10/

```
hello_template = '''
<html>
    <body>
            <h1>Hello {{ recipient }}!</h1>
    <body>
</html>
'''
```

Template string

- Flask code 1: write the template in the application code

```
# -*- encoding: utf8 -*-
from flask import Flask, render_template_string

hello_template = '''
<html>
   <body>
      <h1>Hello {{ recipient }}!</h1>
   <body>
</html>
'''

app = Flask(__name__)
app.config.from_object(__name__)
DEBUG = True

@app.route('/', methods=['GET'])
def helloworld():
   return render_template_string(hello_template, recipient='World')
```

Template file

- **Flask code 2**: write a .html page in the “templates” directory placed in the root of the application directory with the template string

```
+ <project-directory>
    + <flask-directory>
        - <flask_app>.py
        + templates
            - greeting.html
    - requirements.txt
```

```
# -*- encoding: utf8 -*-
from flask import Flask, render_template

app = Flask(__name__)
app.config.from_object(__name__)
DEBUG = True

@app.route('/', methods=['GET'])
def helloworld():
   return render_template(‘greeting.html’, recipient='World')
```

Managing input data

```
from flask import app, request
…
@app.route('/greeting/<recipient>', methods=['GET'])
def helloworld(recipient):
…
    request.args
    request.data
```

- **Args** allows to access the **query** parameter of the request
- **Data** allows to access the payload of the request
  - Not really useful for GET requests (why?)
  - ***Encoding is important!***
    - *Examples: Raw data, Json, Xml, Form data*

Homeworks

- Create a Google Cloud account and try executing the quick start application
  - https://cloud.google.com/appengine/docs/standard/python/quickstart