Building Web Applications

A.Y 2020/2021

Goal

- Create simple web applications in Python
  - For interactive interfaces
  - For server-side components

- Learn a simple framework – Start simple
  - Extensible with modules

Summary

- Flask

- Case studies
  - First Flask basic application
  - Jinja2 Templates
  - User interaction

Flask - what it is

- Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around Werkzeug [1] and Jinja2 [2] and has become one of the most popular Python web application frameworks.

    - WSGI (pronounced *whiskey*) - The Web Server Gateway Interface is a simple calling convention for web servers to forward requests to web applications or frameworks written in the Python programming language

- Flask offers suggestions, but doesn't enforce any dependencies or project layout. It is up to the developer to choose the tools and libraries they want to use. There are many extensions provided by the community that make adding new functionality easy.

1. https://pypi.org/project/Werkzeug/
2. https://www.palletsprojects.com/p/jinja/

Werkzeug

- Werkzeug is a comprehensive WSGI web application library. It began as a simple collection of various utilities for WSGI applications and has become one of the most advanced WSGI utility libraries.
    - Flask wraps Werkzeug, using it to handle the details of WSGI while providing more structure and patterns for defining powerful applications.

Jinja2

- Jinja2 is a full-featured template engine for Python.
  - Flask wraps Jinja2, using it to handle templates.

```
{% extends "layout.html" %}
{% block body %}
  <ul>
  {% for user in users %}
    <li><a href="{{ user.url }}">{{ user.username }}</a></li>
  {% endfor %}
  </ul>
{% endblock %}
```

Flask installation

- Install Flask, Werkzeug and Jinja2 in a single step (system-wide installation)

```
$> sudo pip install Flask
$> …
$> …
$> Successfully installed Flask-1.1.0
```

Flask applications

- One ‘Flask’ object represents the whole application

```
from flask import Flask
app = Flask(__name__)
## __name__ is the application name
```

- Running the application starts the web server (running until you kill it)

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

The web server

- By default, Flask runs a web server on:
  - http://127.0.0.1:5000/
  - Accessible by localhost, only
  - Running on port 5000

- Can be customized with parameters to the .run method

```
# syntax: app.run(host=None, port=None, debug=None,
**options)
app.run(host='0.0.0.0', port=80) # public
app.run(debug=True) # for development
```

Web pages

- Each page is implemented by a method:

```
@app.route('/')
def index():
    return "Hello, web world!"
```

- Must specify
  - The (local) URL at which the page will be visible: '/' in the @app.route decorator
  - The name of the page: index
  - The (HTML) content of the page: return statement

Case study 1: First Flask Application

Route: **/**

Softare Applications 2020/2021

This is the web page of the course

The slides are available here.

Route: **/slides**

Slides

- Introduction
- What is Software?

Back to home.

Solution: HTML

Route: */*

```
<html>

  <h1>Software Applications 2020/2021</h1>
  <p>This is the web page of the course</p>
  <p>The slides are available <a href="slides">here</a>.</
p>

</html>
```

Route: **/slides**

```
<html>
  <head>
    <title>SA 2020/2021</title>
  </head>
  <body>
    <h1>Slides</h1>
    <ul>
      <li>Introduction</li>
      <li>What is Software?</li>
    </ul>
    <p>Back to <a href="/">home</a>.</p>
  </body>
</html>
```

Solution: Flask

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
 return """<html><head><title>SA 2020/2021</title></head><body><h1>Software Applications
2019/2020
   </h1><p>This is the web page
   of the course</p><p>The slides are available <a href="slides">here</a>.</p></body></html>
   """

@app.route('/slides')
def slides():
  return """<html><head><title>SA 2020/2021</title></head><body><h1>Slides</h1><ul>
   <li>Introduction</li><li>What is Software?</li></ul><p>Back to <a href="/">home</a>.</
   p></body>
   </html>
   """
if __name__ == '__main__':
  app.run()
```

https://github.com/anuzzolese/genomics-unibo/blob/master/2020-2021/exercises/flask/basic_app/basic_app.py

HTML templating with Jinja2

- Embedding HTML in Python strings is
  - Ugly
  - Error prone
  - Complex (i.e., must follow HTML escaping rules and Python quoting rules)
  - Did I say Ugly?

- Templating: separating the (fixed) structure of the HTML text (template) from the variable parts (interpolated variables)
- Flask supports the Jinja2 templating engine

Jinja2 basics

- Templates should be in the ./templates subfolder
- Templates are HTML files, with .html extension
- Templates can interpolate passed-by values:
  - {{ parameter }}
  - {{ expression }}
- Templates can include programming statements:
  - {% statement %}
- Templates can access some implicit objects
  - request, session
- Templates are processed when requested by the Flask page

```
return render_template('slides.html', name=name)
```

Case study 2: revised solution with templating

```
from flask import Flask
from flask import render_template

app = Flask(__name__)

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

@app.route('/slides')
def slides():
  return render_template('slides.html')

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

https://github.com/anuzzolese/genomics-unibo/blob/master/2020-2021/exercises/flask/basic_templating/basic_templating.py

Revised solution with templating: HTML

https://github.com/anuzzolese/genomics-unibo/blob/master/2020-2021/exercises/flask/basic_templating/templates/index.html

Route: **/**

Located in: templates/index.html

```
<html>
  <head>
    <title>SA 2020/2021</title>
  </head>
  <body>
    <h1>Software Applications 2020/2021</h1>
    <p>This is the web page of the course</p>
    <p>The slides are available <a href="slides">here</a>.</p>
  </body>
</html>
```

Route: **/slides**

Located in: templates/slides.html

```
<html>
  <head>
    <title>SA 2020/2021</title>
  </head>
  <body>
    <h1>Slides</h1>
    <ul>
      <li>Introduction</li>
      <li>What is Software?</li>
    </ul>
    <p>Back to <a href="/">home</a>.</p>
  </body>
</html>
```

https://github.com/anuzzolese/genomics-unibo/blob/master/2020-2021/exercises/flask/basic_templating/templates/slides.html

Main Jinja2 {% statements %}

- {% for var in list %} ... {% endfor %}

- {% if condition %} ... {% elif cond %} ...
  {% else %} ... {% endif %}

Statements Vs. Expressions

- A {% statement %} controls the flow of execution in a template

  http://jinja.pocoo.org/docs/dev/templates/#list-of-control- structures

- An {{ expression }} evalutates the variable (or the expression) and «prints» the results in the HTML file

  http://jinja.pocoo.org/docs/dev/templates/#expressions

Case study 3

Route: /

Softare Applications 2020/2021

This is the web page of the course

Enter username [_____] [Submit]

Route: /login

Softare Applications 2020/2021

Your nane: name

Continue

Route: /

Softare Applications 2020/2021

This is the web page of the course

Welcome name!

The slides are available.

Logout

Route: /slides

Slides

- Introduction
- What is Software?

Back to home.

Querying request parameters

- All FORM variable are sent with the HTTP request
- Flask packs all FORM variables in the ‘request.form’ object (a dictionary)
- ‘request’ is a global implicit object, and must be imported

```
from flask import request

user = request.form['user']
```

Using parameters in templates

- Specify name=value of all needed parameters in the render_template call
- Within the template, use the {{ name }} syntax
- Template parameters need not be the same as FORM parameters (they are independent concepts, independent values)

Flask

```
myuser = 'Andrea'
return render_template('index.html', user=myuser)
```

Template

```
<p>Welcome {{ user }}.</p>
```

Remembering values

- Values in request.form expire immediately
- We may «remember» values for a longer time
- By storing them in «session» containers – Based on HTTP cookies
  – Kept in memory in the web server
  – Valid until browser disconnection or timeout, only
  – http://flask.pocoo.org/docs/0.10/quickstart/#sessions

Implementing sessions in Flask

- Sessions are automatically initialised and managed by Flask
- Session data is encrypted. For that we must define a secret key
  - app.secret_key = 'softwareapplicationssecret'
- The ‘session’ object is a global shared dictionary that stores attribute-value pairs

Flask

```
myuser = 'Andrea'
session['user'] = myuser
```

Template

```
<p>Welcome {{ session['user'] }}.</p>
```

Automatic redirects

- In some cases, a user action doesn’t need to generate a response page
  – E.g., the Logout action needs to destroy the session, but will just bring you to the normal ‘index’ page

- You may use a ‘redirect’ method to instruct the browser that the current response is empty, and it must load the new page (HTTP 302)

```
return redirect('/index'))
```

Exercise solution

```
from flask import Flask, render_template, request, session, redirect

app = Flask(__name__)
app.secret_key = 'softwareapplicationssecret'

@app.route('/')
def index():
      return render_template('index2.html')

@app.route('/slides')
def slides():
      user = session['user']
      topics = ["Introduction", "What is Software?"]
      return render_template('slides2.html', user=user, slides=topics)

@app.route('/login', methods=['POST'])
def login():
      user = request.form['user']
      session['user'] = user
      return render_template('welcome.html', user=user)

@app.route('/logout')
def logout():
      del session['user']
      return redirect('/')

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

https://github.com/anuzzolese/genomics-unibo/blob/master/2020-2021/exercises/flask/advanced_templating/advanced_templating.py

HTML solution: index2.html

Route: /

Located in: templates/index2.html

```
<html>
  <head>
    <title>SA 2020/2021</title>
  </head>
  <body>
    <h1>Software Applications 2020/2021</h1>

    <p>This is the web page of the course</p>

    {% if session.user %}
    <p>Welcome {{ session['user'] }}!</p>
    <p><img src="static/{{ session['user'] }}.png" /></p>
    <p><a href="logout">Logout</a></p>

    {% else %}
    <p>
      <form action="login" method='POST'>
        Enter name: <input type='text' name='user'>
          <input type='submit' value='Submit'>
      </form>
    </p>
    {% endif %}

  </body>
</html>
```

https://github.com/anuzzolese/genomics-unibo/blob/master/2020-2021/exercises/flask/advanced_templating/advanced_templating.py

HTML solution: welcome.html

Route: **/login**

Located in: templates/welcome.html

```
<html>
  <head>
    <title>SA 2020/2021</title>
  </head>
  <body>
    <h1>Welcome</h1>
    <p>Welcome {{ user }}.</p>
    <p><a href="/">Continue</a> </p>
  </body>
</html>
```

https://github.com/anuzzolese/genomics-unibo/blob/master/2020-2021/exercises/flask/advanced_templating/advanced_templating.py

HTML solution: slides2.html

Route: /

Located in: templates/slides2.html

```
<html>
  <head>
    <title>SA 2020/2021</title>
  </head>
  <body>
    <h1>Slides</h1>
    <ul>

      {% for topic in slides %}
      <li> {{ topic }} </li>
      {% endfor %}

    </ul>
    <p>Back to <a href="/">home</a>.</p>
  </body>
</html>
```

https://github.com/anuzzolese/genomics-unibo/blob/master/2020-2021/exercises/flask/advanced_templating/advanced_templating.py

Exercise

- Process with Pandas the Auto.csv dataset contained into the GitHub repo of the course then:
  1. compute the average number of cylinders into the dataset
  2. compute the maximum number of cylinders into the dataset
  3. compute the minimum number of cylinders into the dataset
  4. present the information about average, maximum, and minimum number of cylinders into a Web page published by a Flask application you implement
  5. The Flask application uses the templating mechanism in order to keep separate the HTML presentation from the Python logics.