PyLadies

Vienna 11.7.2020

Who?

International mentorship group with a focus on helping more women become active participants and leaders in the Python open-source community.

Our mission is to promote, educate and advance a diverse Python community through outreach, education, conferences, events and social gatherings.

Agenda for today

---

1. Introduction to web frameworks
2. Flask
3. Create a web application

Goals

- Learn basics of flask
- Create small static website
- Build weather app as a project

Useful tutorial

- https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world

Web frameworks

- support development of web applications
- Web frameworks aim to automate the overhead associated with common activities performed in web development. For example, many web frameworks provide libraries for **database access**, **templating frameworks**, and **session management**, and they often promote code reuse.

Other web frameworks in Python

- Django - more complex than flask
- Bottle - smaller than Flask, single file module, no dependencies
- Falcon - micro-framework, minimal dependencies, nice for simple REST API

Flask

- Simple and fast to start with
- Built in development server and debugger
- Integrated unit testing support
- Uses Jinja2 template system
- RESTful request dispatching

Flask

- first install flask -> *pip3 install flask*
- create file for your first app
- when you run it you can access your app on

http://localhost:5000/

or

http://127.0.0.1:5000/

Flask minimal app

```
from flask import Flask
app = Flask(__name__)
@app.route("/")
def hello():
    return "Hello World!"
if __name__ == "__main__":
    app.run()
```

URL routes

```
@app.route("/PyLadies")
def PyLadies():
    return "PyLadies Flask"
```

http://127.0.0.1:5000/PyLadies

Styles

In Flask, styling of your page is done using templates

- create directory called /templates in your working directory
- create flask01.html file with *<h1>Hello name</h1>*

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

@app.route("/hello/Tyna/")

def hello():

    return render_template(

    'flask01.html')
```

Port number

- You can change default port number with following code

```
if __name__ == "__main__":
    app.run(host='127.0.0.1', port=8000)
```

- Careful about privileged port numbers (5432 postgresql for example)

Passing variable into template

```
<h2>Here is random friend for you:</h2>

{{friend}}
```

- (add from random import randint)

```
@app.route("/hello/Tyna/")
def hello():
    friends= ['Anna', 'Tyna', 'Josh', 'Lisa' ]
    randomNumber = randint(0,len(friends)-1)
    friend= friends[randomNumber]
    return render_template(
    'flask01.html',**locals())
```

Templates - advanced

- Jinja2 can do *if* conditions
- loops are supported
- more advanced styling with supported imports

```
{% if title %}
<title>{{ title }}</title>
{% else %}
<title>Welcome to PyLadies</title>
{% endif %}
```

Web forms

- For login, blogposts and other communication
- *pip3 install flask-wtf*
- create config.py file with basic secret key configuration

```
import os
class Config(object):
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'pyladies-vienna-are-awesome'
```

Web forms

- add to main file running the app:

```
from config import Config

app.config.from_object(Config)
```

forms.py

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, BooleanField, SubmitField
from wtforms.validators import DataRequired

class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    remember_me = BooleanField('Remember Me')
    submit = SubmitField('Sign In')
```

Web forms

- All the codes can be found here:

github.com/UndeadFairy/pyladies_vienna/flask/login_app

GET and POST method

- GET is used to request data from a specified resource
- POST is used to send data to a server to create/update a resource
- Main difference between POST and GET method is that GET carries request parameter appended in URL string while POST can carry request parameter in message body which makes it more secure way of transferring data from client to server in http protocol.

PythonAnywhere.com

- easy and simple way to run/deploy your Python web app
- free and paid plans, interactive tutorial on register
- beginner plan: one limited web app at <username>.pythonanywhere.com
- Python consoles, files, web apps, paid jupyter notebooks
- deploy own web app via manual configurations or create a simple one there
- support for simple scheduled tasks, database support (postgresql, mysql)

PythonAnywhere.com deploy username.pythonanywhere.com

- **from scratch:** *webapps -> Add a new web app*, choose *Flask*, default domain -> *next*, *select Python version*, default file path -> next
- **Get your app there:** 1) **upload** or **git clone** your code to */home/username/mysite*, check config if *SERVER_NAME* was set
- mkvirtualenv --python=/usr/bin/python3.8 my-virtualenv && pip install flask
- configure WSGI file at /var/www/ with importing your flask app

```
# app = Flask(__name__) at /home/username/mysite/flask_app.py
import sys
path = '/home/username/mysite'
if path not in sys.path:
   sys.path.insert(0, path)
from flask_app import app as application
```

1) https://help.pythonanywhere.com/pages/Flask/
2) https://blog.pythonanywhere.com/121/

Project - Weather app

Idea: enter name of city/coordinates, via HTML form, send a request for current weather or forecast to a weather data provider API, parse the returned response, pick information relevant for you and display it

Example of weather data provider: https://openweathermap.org registration needed, data provided in JSON, HTML, XML

API token (appid) created instantly, but activated in a few hours, in the meantime, use: **0f90e177e9b3f49f88ca6c99f92c16b3**

Project - Weather app

Examples of requests:

api.openweathermap.org/data/2.5/weather?q=London&APPID=<yourAppId>

api.openweathermap.org/data/2.5/weather?q=London,uk&APPID=<yourAppId>

api.openweathermap.org/data/2.5/weather?lat=<lat>&lon=<lon>&APPID=<yourAppId>

Fetch from API with requests library:

```
city = 'Prague'
url = 'https://api.openweathermap.org/data/2.5/weather?q={}&appid=<yourAppId>'
r = requests.get(url.format(city)).json()
weather = {
    'city' : city,
    'temperature' : r['main']['temp'],
}
return render_template('weather.html', weather_data=[weather])
```

web app considerations

- ensure your app does not crash on bad user input, if there is any
- your remote API ID should NEVER be committed to git! - more on next page - how to go around it: https://help.pythonanywhere.com/pages/environment-variables-for-web-apps/
- while requesting something from remote (API), consider bad connection, handle errors on their side = outside of your control, inform user about error status code 1)
- Try remote requests locally first, inspect response, (RestMan browser plugin, Postman software, curl - linux)

- https://www.restapitutorial.com/httpstatuscodes.html

Sum it up

- Flask is a lightweight web framework useful for fast and simple app development
- Easier to develop first app and deliberately more simple than on Django
- Still powerful enough for apps like weather api

Resources and materials general

- advent of code - adventofcode.com
- hackerrank - hackerrank.com
- Django Girls django tutorial -> (29th August in Vienna)
- https://www.practicepython.org
- Nice Python exercices at one place
  https://github.com/tystar86/python_exercises
- https://automatetheboringstuff.com
- https://diveintopython3.problemsolving.io

Next topics

Databases

Graphics

GUI

fill the form regarding your interests please :) → https://forms.gle/UtfgVGe6AhhRwx539

Thank you and see you next time

Coding session - **September 2020**

Next workshop - **23.8.2020** Databases