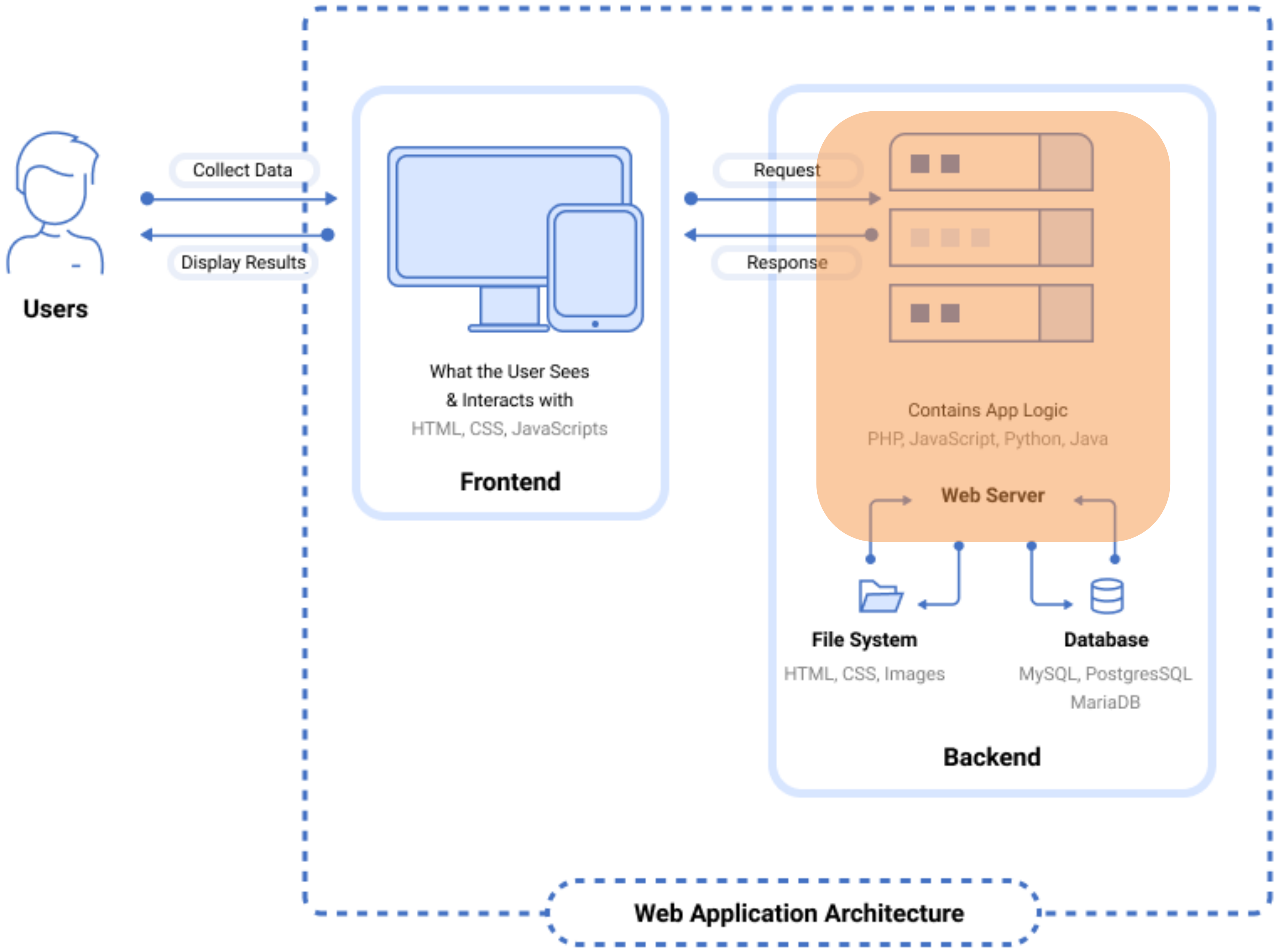
Flask

The server side

Luigi De Russis

Goal

- Create web applications
  - In Python
  - Using the learnt front-end technologies
  - On the server side
- Learn a web framework
  - Start simple, minimal
  - Extensible


Users
Collect Data
Display Results
What the User Sees
& Interacts with
HTML, CSS, JavaScripts
Frontend
Request
Response
Contains App Logic
PHP, JavaScript, Python, Java
Web Server
File System
HTML, CSS, Images
Database
MySQL, PostgresSQL
MariaDB
Backend
Web Application Architecture

Programming the Web with Python

GETTING STARTED WITH FLASK

Python meets the Web

- Python includes a `SimpleHTTPServer` to activate a web server
  - Low-level, not very friendly
- Several libraries and frameworks were developed
  - Different features and complexity
- Flask is one of the most popular micro-frameworks
  - Simple and easy to use

CherryPy

https://cherrypy.dev

Flask
web development, one drop at a time

https://flask.palletsprojects.com

django

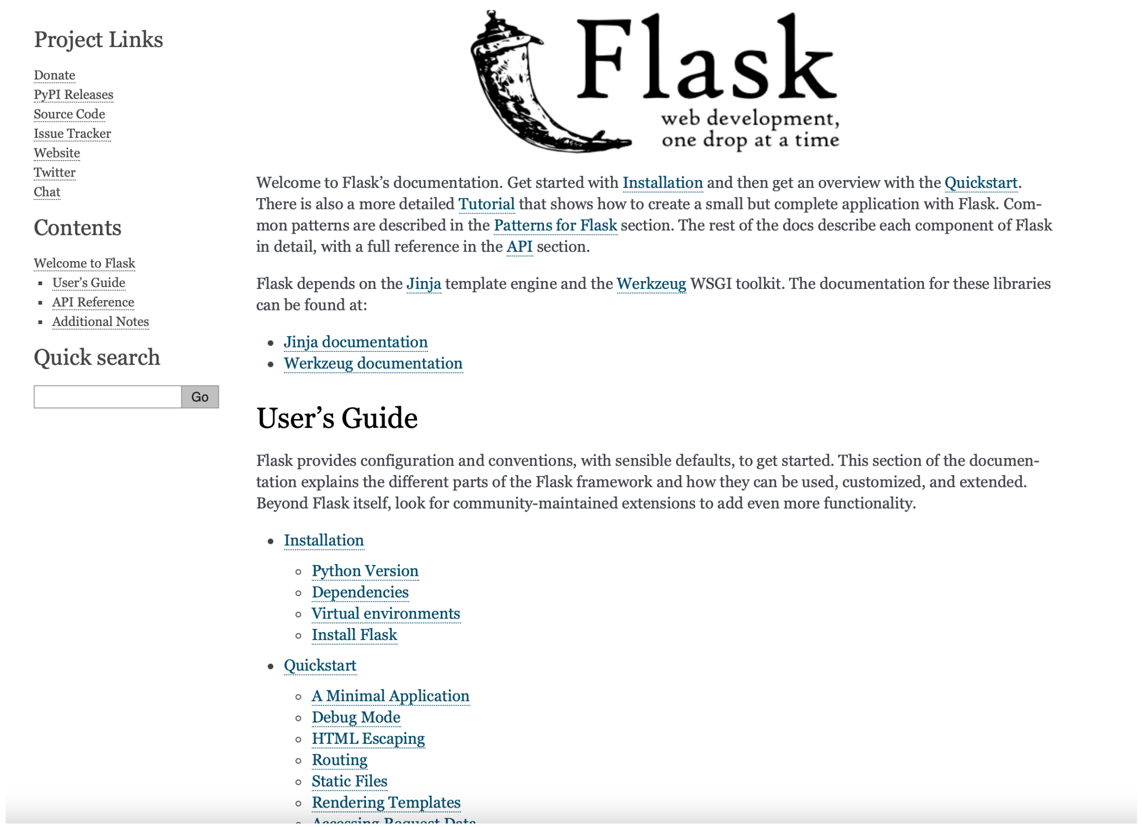
https://www.djangoproject.com

https://wiki.python.org/moin/WebFrameworks

Flask

- A non-full stack web framework for Python
  - Web server based on Werkzeug (WSGI toolkit, https://werkzeug.palletsprojects.com)
  - Application context
  - Conventions and configurations
- Include a template engine
  - Jinja, https://jinja.palletsprojects.com/
  - Easy editing of dynamic HTML pages
  - Powerful: operators and inheritance

Flask Resources (recap)



Project Links

- Donate
- PyPI Releases
- Source Code
- Issue Tracker
- Website
- Twitter
- Chat

Contents

- Welcome to Flask
  - User's Guide
  - API Reference
  - Additional Notes

Quick search

Go

Welcome to Flask's documentation. Get started with Installation and then get an overview with the Quickstart. There is also a more detailed Tutorial that shows how to create a small but complete application with Flask. Common patterns are described in the Patterns for Flask section. The rest of the docs describe each component of Flask in detail, with a full reference in the API section.

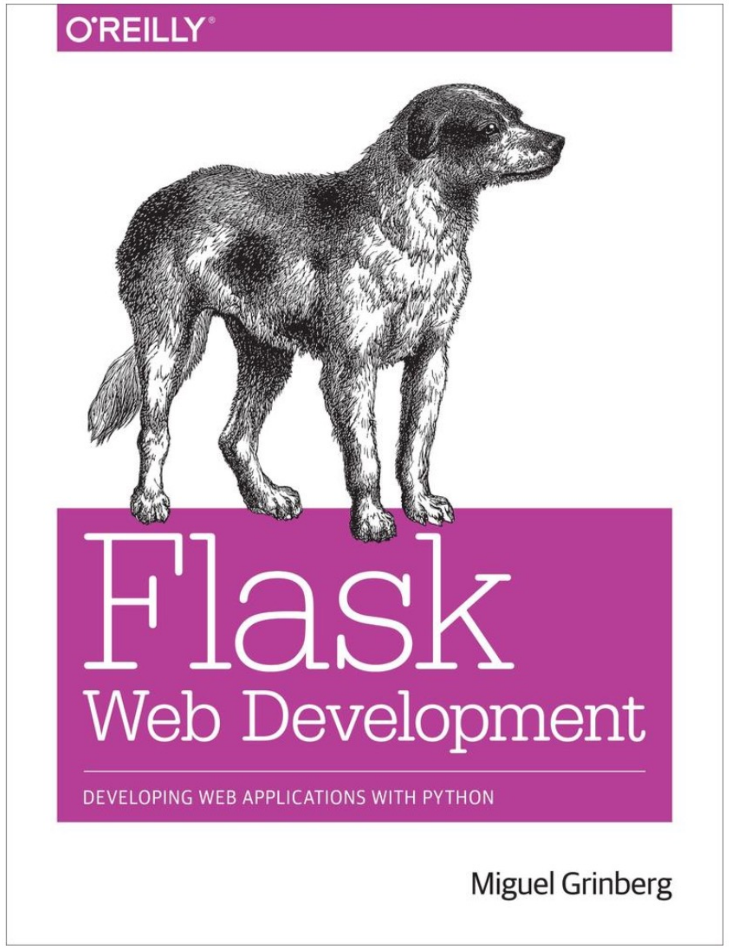
Flask depends on the Jinja template engine and the Werkzeug WSGI toolkit. The documentation for these libraries can be found at:

- Jinja documentation
- Werkzeug documentation

User's Guide

Flask provides configuration and conventions, with sensible defaults, to get started. This section of the documentation explains the different parts of the Flask framework and how they can be used, customized, and extended. Beyond Flask itself, look for community-maintained extensions to add even more functionality.

- Installation
  - Python Version
  - Dependencies
  - Virtual environments
  - Install Flask
- Quickstart
  - A Minimal Application
  - Debug Mode
  - HTML Escaping
  - Routing
  - Static Files
  - Rendering Templates

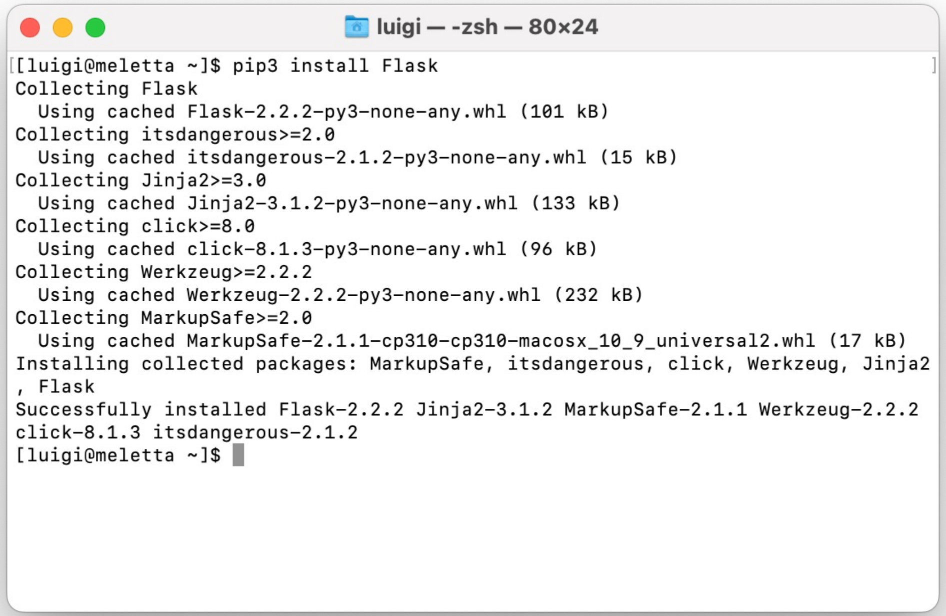


Installation

- Use `pip` (or `pip3`) to install Flask
  - It comes with Werkzeug and Jinja among its dependencies
  - You need Python 3.7 or *higher*

- **Option 1**: system-wide installation

```
pip install Flask
```



luigi — -zsh — 80×24

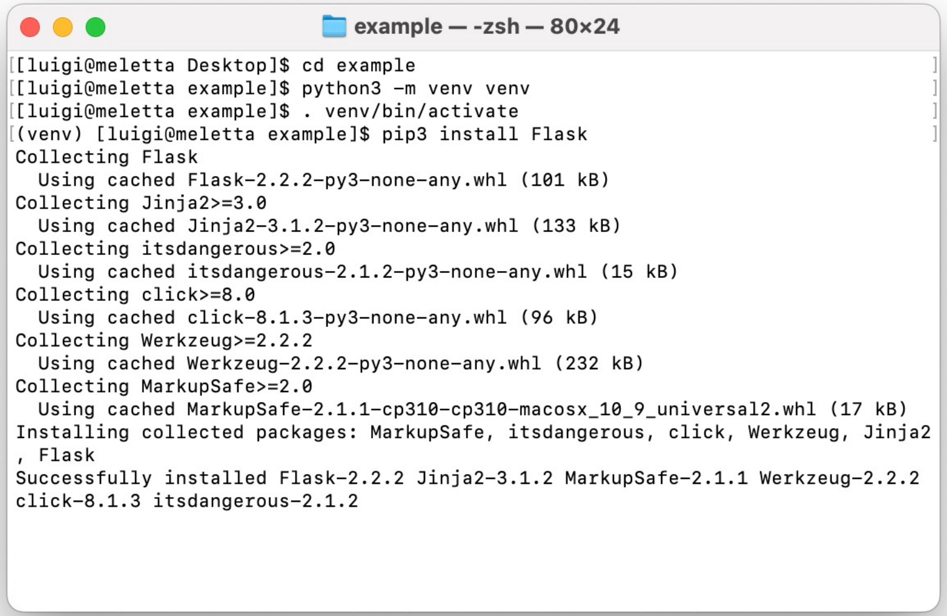
```
[luigi@meletta ~]$ pip3 install Flask
Collecting Flask
  Using cached Flask-2.2.2-py3-none-any.whl (101 kB)
Collecting itsdangerous>=2.0
  Using cached itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting Jinja2>=3.0
  Using cached Jinja2-3.1.2-py3-none-any.whl (133 kB)
Collecting click>=8.0
  Using cached click-8.1.3-py3-none-any.whl (96 kB)
Collecting Werkzeug>=2.2.2
  Using cached Werkzeug-2.2.2-py3-none-any.whl (232 kB)
Collecting MarkupSafe>=2.0
  Using cached MarkupSafe-2.1.1-cp310-cp310-macosx_10_9_universal2.whl (17 kB)
Installing collected packages: MarkupSafe, itsdangerous, click, Werkzeug, Jinja2
, Flask
Successfully installed Flask-2.2.2 Jinja2-3.1.2 MarkupSafe-2.1.1 Werkzeug-2.2.2
click-8.1.3 itsdangerous-2.1.2
[luigi@meletta ~]$
```

Installation

- **Option 2**: within a virtual environment (project-specific)

```
cd myproject
python3 -m venv venv
. venv/bin/activate (on Windows: venv\Scripts\activate)
pip install Flask
```

deactivate will close the virtual environment



```
[luigi@meletta Desktop]$ cd example
[luigi@meletta example]$ python3 -m venv venv
[luigi@meletta example]$ . venv/bin/activate
(venv) [luigi@meletta example]$ pip3 install Flask
Collecting Flask
  Using cached Flask-2.2.2-py3-none-any.whl (101 kB)
Collecting Jinja2>=3.0
  Using cached Jinja2-3.1.2-py3-none-any.whl (133 kB)
Collecting itsdangerous>=2.0
  Using cached itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting click>=8.0
  Using cached click-8.1.3-py3-none-any.whl (96 kB)
Collecting Werkzeug>=2.2.2
  Using cached Werkzeug-2.2.2-py3-none-any.whl (232 kB)
Collecting MarkupSafe>=2.0
  Using cached MarkupSafe-2.1.1-cp310-cp310-macosx_10_9_universal2.whl (17 kB)
Installing collected packages: MarkupSafe, itsdangerous, click, Werkzeug, Jinja2
, Flask
Successfully installed Flask-2.2.2 Jinja2-3.1.2 MarkupSafe-2.1.1 Werkzeug-2.2.2
click-8.1.3 itsdangerous-2.1.2
```

A Minimal Flask App

- Calling Flask() creates an application object **app**
- Incoming HTTP requests are routed to a function according to its route decorator
  - path (mandatory param), e.g., “/”
  - options, e.g., “`methods=[POST]`”
- The function bound to a route returns the message we want to display in the browser
  - HTML is the default content type

```
# import module
from flask import Flask


# create the application
app = Flask(__name__)


# define routes and web pages
@app.route('/')
def hello_world():
    return 'Hello, World!'
```

Flask Applications

- One `Flask` object represents the entire web application

```
from flask import Flask

app = Flask(__name__)
## __name__ is a shortcut for the application name
```

Routing: Decorator

- *Almost* each web page is represented by a decorator (+ a function)
- @**app**.route(path, options)
  - **app**: the Flask object
  - path: a string representing a path on the server
    - Matched with the path in the HTTP Request Message
  - options: extra options such as…
    - methods: an array of one or more HTTP Request method (GET, POST, PUT, DELETE, ...); if omitted, the default is GET
    - redirect_to: a string or a function representing the target of the redirect
    - Other (less frequent) options at https://werkzeug.palletsprojects.com/en/2.2.x/routing/#werkzeug.routing.Rule

Routing: Function

- def index():

  return 'Hello, world!'
- index is the name of the page
- The HMTL content of the page (e.g., “Hello World”) is in the return statement

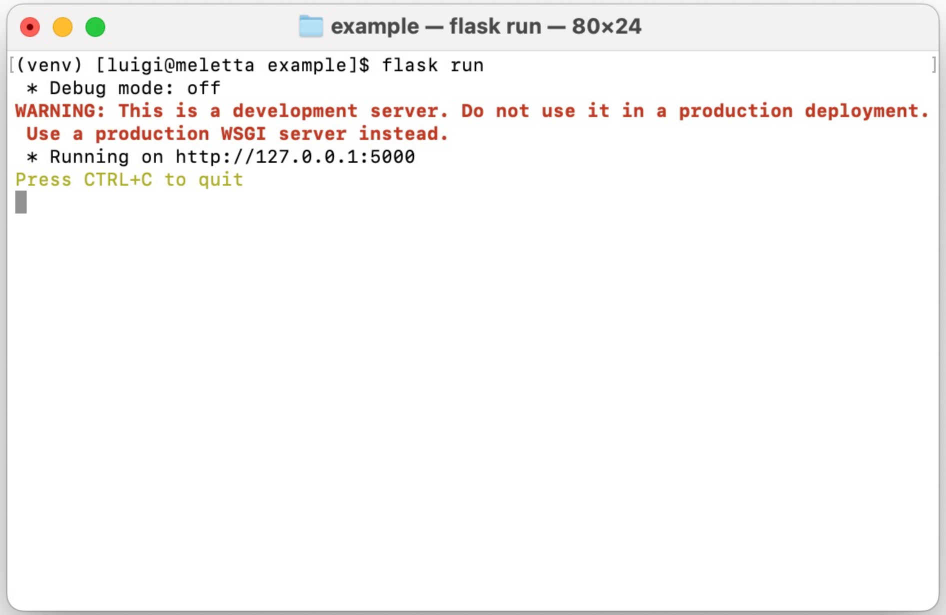
- If the decorator declares more HTTP methods, the actually called method is available in the `request.method` variable of the bound function

Running a Flask App

- To run a Flask application:

  `flask --app main run`

  – where “main” is the name (with the path, without extension) of the Python file
  – Shortcut: if the file is called “app.py”, you can just type `flask run`



example — flask run — 80×24

```
(venv) [luigi@meletta example]$ flask run
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
 Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Flask’s (Development) Web Server

- By default, Flask runs a web server at:
  - http://127.0.0.1:5000/
  - Accessible by **localhost**, only
  - Running on port 5000
  - Great for **development** and test
- It can be customized with options when launching the application (before “run”):
  - with debug mode enabled -> use the `--debug` option
  - externally-visible server -> use `--host=0.0.0.0` before “run”
  - use a different port -> use `--port=3000` (e.g., to use 3000 as the port)

Alternative Way to Run the Application

- Instead of the Flask CLI, you can start the development server in code

```
if __name__ == "__main__":
  app.run(host='0.0.0.0', port=3000, debug= True)
  # or a subset of these options
```

- Then, launch the application as a normal Python program:

```
python3 app.py
```

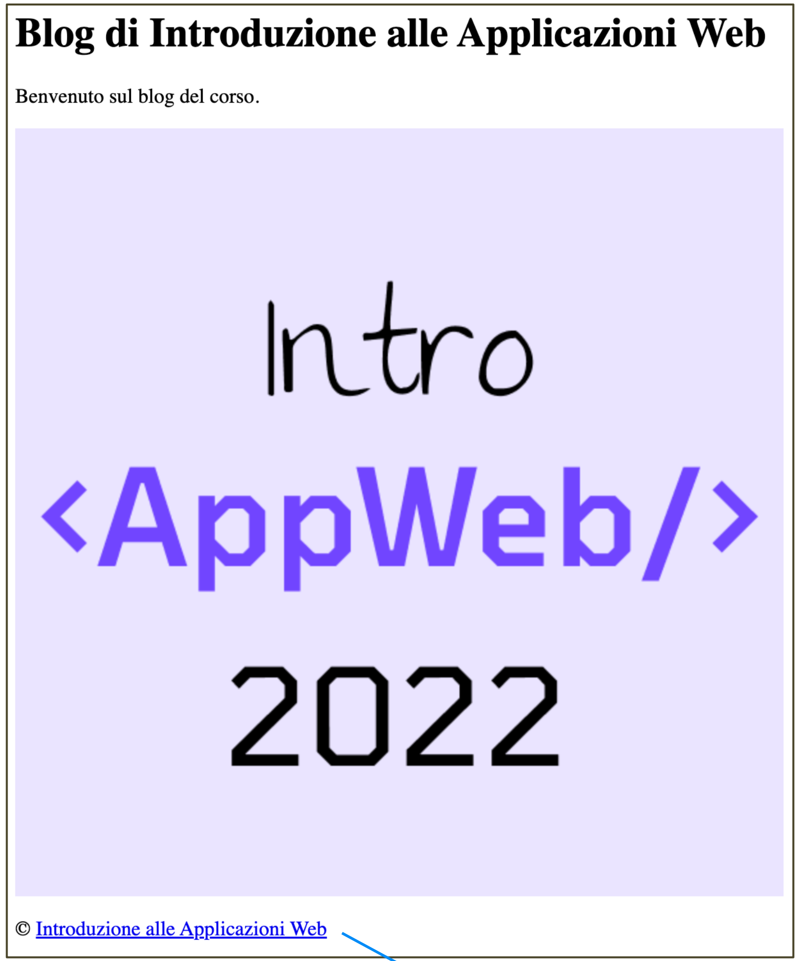
Running a ‘Public’ Web Server

- Bind to all IP addresses of your machine
  - `host='0.0.0.0'`
- Use a standard port
  - `port=80` *(must be launched as ‘root’)*
  - `port=3000` *(>1024, does not require root)*
- Check the firewall, and open the host/port combination for external access

Beware hackers and intruders!

Example

/

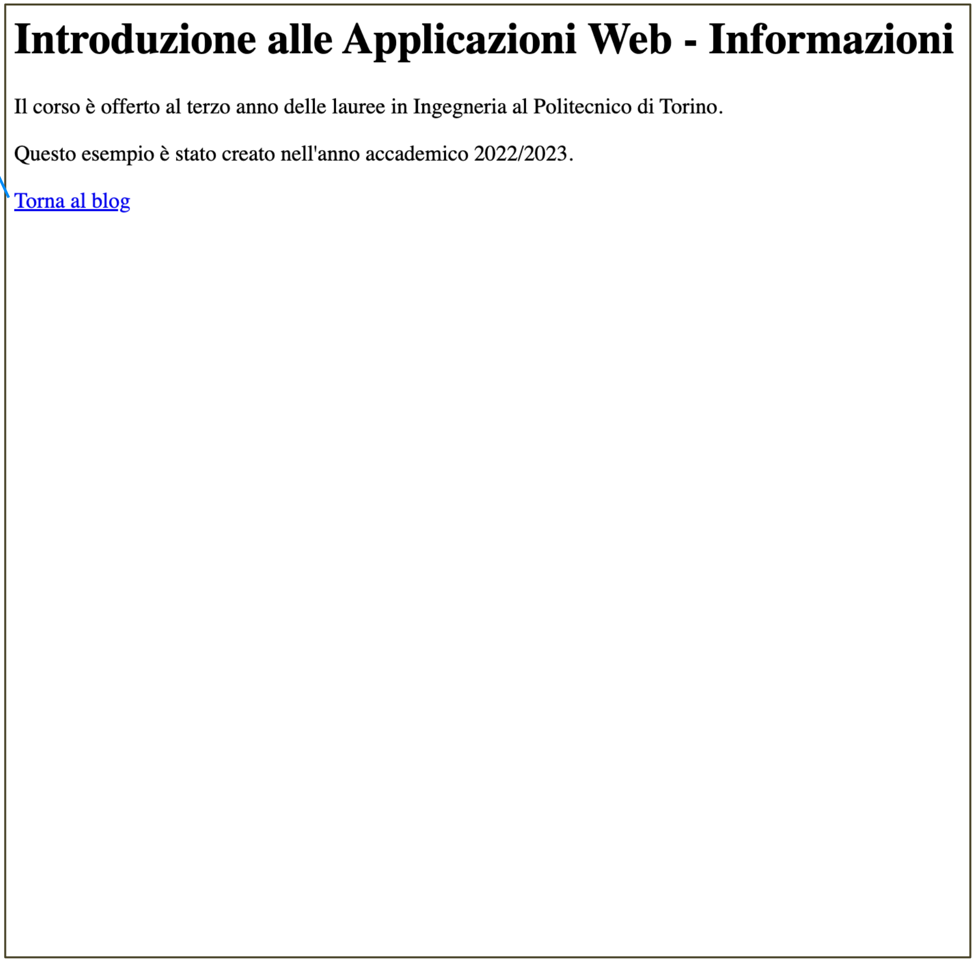


Blog di Introduzione alle Applicazioni Web

Benvenuto sul blog del corso.

© Introduzione alle Applicazioni Web

/about.html



Introduzione alle Applicazioni Web - Informazioni

Il corso è offerto al terzo anno delle lauree in Ingegneria al Politecnico di Torino.

Questo esempio è stato creato nell'anno accademico 2022/2023.

Torna al blog

Solution 1

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
  return """<html><head><title>IAW - Home</title></head>
    <body><h1>Blog di Introduzione alle Applicazioni Web</h1>
    <p>Benvenuto sul blog del corso.</p>
    <p><img src="static/logo.png"></p>
    <p>&copy; <a href="about.html">Introduzione alle Applicazioni Web</a></p>
    </body></html>
    """

@app.route('/about.html')
def about():
  return """<html><head><title>IAW - Sul corso</title></head>
    <body><h1>Introduzione alle Applicazioni Web - Informazioni</h1>
    <p>Il corso &egrave; offerto al terzo anno delle lauree in Ingegneria al Politecnico di Torino.</p>
    <p>Questo esempio &egrave; stato creato nell'anno accademico 2022/2023.</p>
    <p><a href="/">Torna al blog</a></p>
    </body></html>
    """
```

Generate URLs

- **Bad:** encode destination URL a strings! ☹
  - Why?
- **Good:** Python generates the appropriate URL for a function!

```
url_for('<function_name>')
```

- You can use it for static files (images, CSS, …), too

```
url_for('static', filename='image.jpg')
```

  - **Beware:** the subfolder <u>must</u> be called "static"

Solution 2

```python
from flask import Flask, url_for

app = Flask(__name__)

@app.route('/')
def index():
  return ('<html><head><title>IAW - Home</title></head>' +
    '<body><h1>Blog di Introduzione alle Applicazioni Web</h1>' +
    '<p>Benvenuto sul blog del corso.</p>' +
    '<p><img src="' + url_for('static', filename='logo.png') +'"></p>' +
    '<p>&copy; <a href="' + url_for('about') + '">Introduzione alle Applicazioni Web</a></p>' +
    '</body></html>')

@app.route('/about.html')
def about():
  return ('<html><head><title>IAW - Sul corso</title></head>' +
    '<body><h1>Introduzione alle Applicazioni Web - Informazioni</h1>' +
    '<p>Il corso &egrave; offerto al terzo anno delle lauree in Ingegneria al Politecnico di Torino.</p>' +
    '<p>Questo esempio &egrave; stato creato nell\'anno accademico 2022/2023.</p>' +
    '<p><a href="' + url_for('index') + '">Torna al blog</a></p>' +
    '</body></html>')
```

JINJA TEMPLATES

Templating

- 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** comes to help
  - separating the (fixed) structure of the HTML text (template) from the variable parts (interpolated Python variables)
- Flask supports the **Jinja** templating engine out of the box

Jinja Basics

- Flask looks for templates 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 have comments (not included in the final output):
  - `{# comment #}`
- Templates can access some implicit objects
  - `request`, `session`, `g`
- Templates are processed when requested by the Flask page

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

Solution 3 - app.js

```
from flask import Flask, render_template

app = Flask(__name__)

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

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

Solution 3 - Templates

templates/index.html

```
<html>
  <head>
    <title> IAW - Home</title>
  </head>
  <body>
    <h1>Blog di Introduzione alle Applicazioni 
Web</h1>
    <p>Benvenuto sul blog del corso.</p>
    <p>
      <img src="{{ url_for('static', 
filename='logo.png') }}">
    </p>
    <p>&copy; <a href="{{ url_for('about') 
}}">Introduzione alle Applicazioni Web</a></p>
  </body>
</html>
```

templates/about.html

```
<html>
  <head>
    <title> IAW - Sul corso</title>
  </head>
  <body>
    <h1>Introduzione alle Applicazioni Web - 
Informazioni</h1>
    <p>Il corso &egrave; offerto al terzo anno delle 
lauree in Ingegneria al Politecnico di Torino.</p>
    <p>Questo esempio &egrave; stato creato 
nell'anno accademico 2022/2023.</p>
    <p><a href="{{ url_for('index') }}">Torna al 
blog</a></p>
  </body>
</html>
```

Jinja Main Programming Statements

- For – loop over items in a sequence (list, dictionary)

```
{% for variable in list %} … {% endfor %}
```

- If – test a variable against a condition

```
{% if condition %} … {% elif another_condition %}
… {% else %} … {% endif %}
```

Example: For Statement

app.js

```
…
@app.route('/about.html')
def about():
   teachers =
     [{'id': 1234, 'name': 'Luigi'},
      {'id': 1356, 'name': 'Alberto'},
      {'id': 1478, 'name': 'Juan'}]
   return
     render_template('about.html’,
       teachers=teachers)
…
```

templates/about.html

```
…
<h2>Docenti</h2>
<ul>
{% for teacher in teachers %}
  <li>{{ teacher.name |e }}</li>
{% endfor %}
</ul>
…
```

Escaping: if the variable **may** include any of >, <, &, or ” you SHOULD escape it. If you are 100% sure that the variable contains well-formed and trusted HTML, you can skip the escaping.

Statements vs. Expressions

- A {% statement %} controls the flow of execution in a template
  - https://jinja.palletsprojects.com/en/3.1.x/templates/

- An {{ expression }} evaluates the variable (or the expression) and "*prints*" the results in the HTML file
  - https://jinja.palletsprojects.com/en/3.1.x/templates/#expressions

Template Inheritance

- In our examples so far, we have **duplicated** HTML code (e.g., `<html>`, part of the title)
- In more complex web applications, various pages have many common elements, for instance:
  - Navigation bar(s)
  - Footers
  - Page layout
  - …
- Jinja has blocks and template inheritance to help
- No changes are required in the Python application file!

Main Template Inheritance Tags

- `{% block <block_name> %}…{% endblock %}`
  - Defines a block, a reusable HTML component, within a template
  - `block_name` must be unique within a given template
- `{% extends "filename.html" %}`
  - Extends a parent template so that a child template can use it
  - `filename.html` can be within the same folder of the child template or in a different one, e.g., ‘`layout/filename.html`’

Example - Base Template

templates/base.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>IAW - {% block title %}{% endblock %}</title>
  </head>
  <body>
    {% block content %}{% endblock %}
  </body>
</html>
```

Example - Child Templates

templates/index.html

```
{% extends "base.html" %}
{% block title %}Blog{% endblock %}
{% block content %}
  <h1>Blog di Introduzione alle
Applicazioni Web</h1>
  <p>Benvenuto sul blog del corso.</p>
  <p>
    <img src="{{ url_for('static',
filename='logo.png') }}">
  </p>
  <p>&copy; <a href="{{
url_for('about') }}">Introduzione alle
Applicazioni Web</a></p>
{% endblock %}
```

templates/about.html

```
{% extends "base.html" %}
{% block title %}Sul corso{% endblock %}
{% block content %}
  <h1>Introduzione alle Applicazioni Web -
Informazioni</h1>
  <p>Il corso &egrave; offerto al terzo anno delle lauree
in Ingegneria al Politecnico di Torino.</p>
  <p>Questo esempio &egrave; stato creato nell'anno
accademico 2022/2023.</p>
  <hr/>
  <h2>Docenti</h2>
  <ul>
  {% for teacher in teachers %}
    <li>{{ teacher.name | e }}</li>
  {% endfor %}
  </ul>
  <p><a href="{{ url_for('index') }}">Torna al
blog</a></p>
{% endblock %}
```

Super Blocks

- It is possible to render the content of the parent block by calling `super()`
- For instance, if the parent template defines:

```
<head>
    {% block head %}
    <link rel="stylesheet" href="style.css" />
    <title>IAW - {% block title %}{% endblock %}</title>
    {% endblock %}
</head>
```

- The child block can add its own custom CSS file with:

```
{% block title %}Sul corso{% endblock %}
{% block head %}
    {{ super() }}
    <link rel="stylesheet" href="about.css" />
{% endblock %}
```

DYNAMIC ROUTES

Dynamic Routes

- A route can be dynamic
- It can include a **<parameter>** passed as a function argument

```
@app.route('/users/<username>')
def show_profile(username):
   return 'User %s' % username
```

  – In this example, this route will be called with /users/luigidr, /users/amonge, /users/jpsaenz, …

- Parameter are considered as **string** in the function

Dynamic Routes

- Parameters are considered as string in the function
- Automatic conversion is obtained by specifying the parameter type in the decorator

```
@app.route('/posts/<int:id>')
def show_post(id):
   return 'Post %d' % id # this is an integer
```

- Parameter type can be:
  - int, float
  - path (string that might include slashes)
  - missing (default to string)

Generating URLs With Parameters

- `url_for` accepts parameters
- Encoded as variable URLs, **if** the route is **dynamic**

```
@app.route('/users/<username>')
def show_profile(username):
   return 'User %s' % username
```

```
url_for('show_profile', username='luigidr')

→ /users/luigidr
```

Generating URLs With Parameters

- Instead, **if** the route is **static**, it is encoded as a GET parameters
  - Or if the route does not contain a named param

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

```
url_for('about') → /about

url_for('about', username='luigidr’)
  → /about?username=luigidr
```

FLASK EXTENSIONS

Flask Extensions

- Web applications share
  - A generally standardized architecture
  - Many common and repetitive actions
  - Many security risks associated with user input and database interactions
  - …
- Many extensions are available to automate most of the most boring or most risky tasks
  - https://flask.palletsprojects.com/en/2.2.x/extensions/

Some Useful Flask Extensions

- **Flask-WTF**: Integration with WTForms (form creation, validation, regeneration)
- **Flask-SQLAlchemy**: Integration with SQLAlchemy, and object-relational mapping for database storage
- **Bootstrap-Flask**: Help render Flask-related data and objects to Bootstrap markup HTML more easily. Suggested now!
- **Flask-Login**: Management of user sessions for logged-in users
- **Flask-Session:** Add support for Server-side Session to a Flask application
- **Flask-RESTful**: Tools for building RESTful APIs
- **Flask-Mail:** Sdds SMTP mail sending to a Flask application

Bootstrap-Flask

- Bootstrap-Flask packages Bootstrap 4 or 5 into an extension that can help render components in Flask.
- Package available at
  - https://pypi.org/project/Bootstrap-Flask/
  - Install with pip
- Documentation available at
  - https://bootstrap-flask.readthedocs.io/en/stable/

How To Use

- Import and initialize it

```
from flask_bootstrap import Bootstrap5
from flask import Flask

app = Flask(__name__)
bootstrap = Bootstrap5(app)
```

- Use `{{ bootstrap.load_css() }}` and `{{ bootstrap.load_js() }}` to load Bootstrap's resources in the template
- Create a base template such as https://bootstrap-flask.readthedocs.io/en/stable/basic/#starter-template

How To Use

- Use the pre-defined blocks you need
- For instance, a sample navbar can be:

```
{% from 'bootstrap5/nav.html' import render_nav_item %}

<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="navbar-nav mr-auto">
        {{ render_nav_item('index', 'Home') }}
        {{ render_nav_item('explore', 'Explore') }}
        {{ render_nav_item('about', 'About') }}
    </div>
</nav>
```

- where the render_nav_item() renders a navigation item according to the Bootstrap style

Next Steps

- A list of all the pre-defined block, with code examples, is available at
  - https://bootstrap-flask.readthedocs.io/en/stable/macros/
- Demo at http://173.212.227.186 (for Bootstrap 5)
- The extension works nicely with other extensions:
  - Flask-WTF for form handling
  - Flask-SQLAlchemy for database access

License

- These slides are distributed under a Creative Commons license "**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**"
- **You are free to:**
  - **Share** — copy and redistribute the material in any medium or format
  - **Adapt** — remix, transform, and build upon the material
  - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
  - **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
  - **NonCommercial** — You may not use the material for commercial purposes.
  - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
  - **No additional restrictions** — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.
- https://creativecommons.org/licenses/by-nc-sa/4.0/