Flask

web development, one drop at a time

An Overview

James R. Small, Principal Architect

Michigan!/usr/group

mug.org – A Free and Open Source Michigan Community

HOW DID I GET HERE?

- Udacity's Full Stack Web Developer Nanodegree

- Miguel Grinberg’s Flask Mega-Tutorial Course

- Matt Mikai’s Full Stack Python Site

- Michael Kennedy’s Talk Python to Me Podcast

FLASK OVERVIEW – ROADMAP

- **Where Flask fits within the Web Ecosystem**
- Your first Flask application
- Building a Flask application
- Next steps and interesting adjacent areas

THE WEB ECOSYSTEM

FrontEnd

Browser

- HTML/CSS Rendering Engine
- JavaScript Engine
- DevTools

Languages

- HTMLv5
- CSSv3
- JavaScript/ES2018 (ECMAScript)
- TypeScript
- WebAssembly

JavaScript Frameworks/Libraries

- Sites: Bootstrap, jQuery, Foundation
- SPA: React, Angular, Vue.js

BackEnd

Web Servers

- Apache
- Nginx
- IIS

Web Application Servers (WSGI)

- Gunicorn (Python WSGI Servers)
- uWSGI
- mod_wsgi

Web Frameworks (Python)

- Django (Full Featured)
- Flask (Micro - DIY)
- Sanic (Async)

Databases

- PostgreSQL
- MySQL/MariaDB
- Oracle
- SQL Server
- MongoDB

THE WEB ECOSYSTEM

FrontEnd

Browser
- HTML/CSS Rendering Engine
- JavaScript Engine
- DevTools

Languages
- HTMLv5
- CSSv3
- JavaScript/ES2018 (ECMAScript)
- TypeScript
- WebAssembly

JavaScript Frameworks/Libraries
- Sites: Bootstrap, jQuery, Foundation
- SPA: React, Angular, Vue.js

BackEnd

Web Servers
- Apache
- Nginx
- IIS

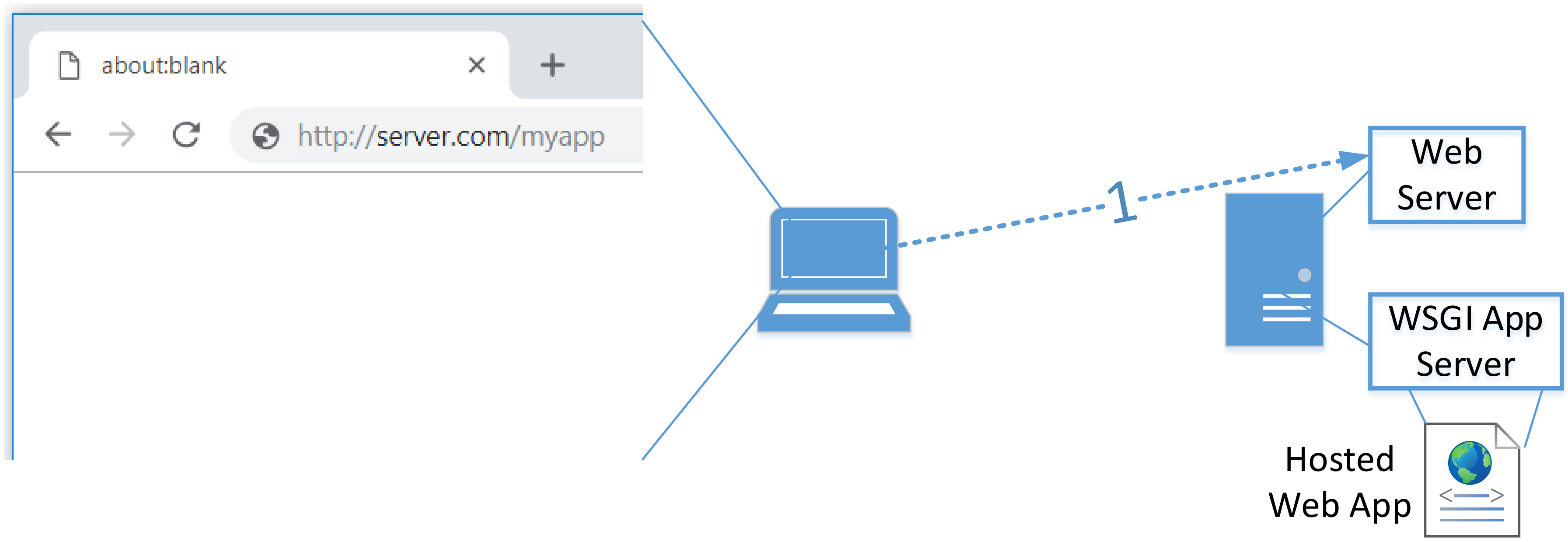
Web Application Servers (WSGI)
- Gunicorn (Python WSGI Servers)
- uWSGI
- mod_wsgi

Web Frameworks (Python)
- Django (Full Featured)
- Flask (Micro - DIY)
- Sanic (Async)

Databases
- PostgreSQL
- MySQL/MariaDB
- Oracle
- SQL Server
- MongoDB

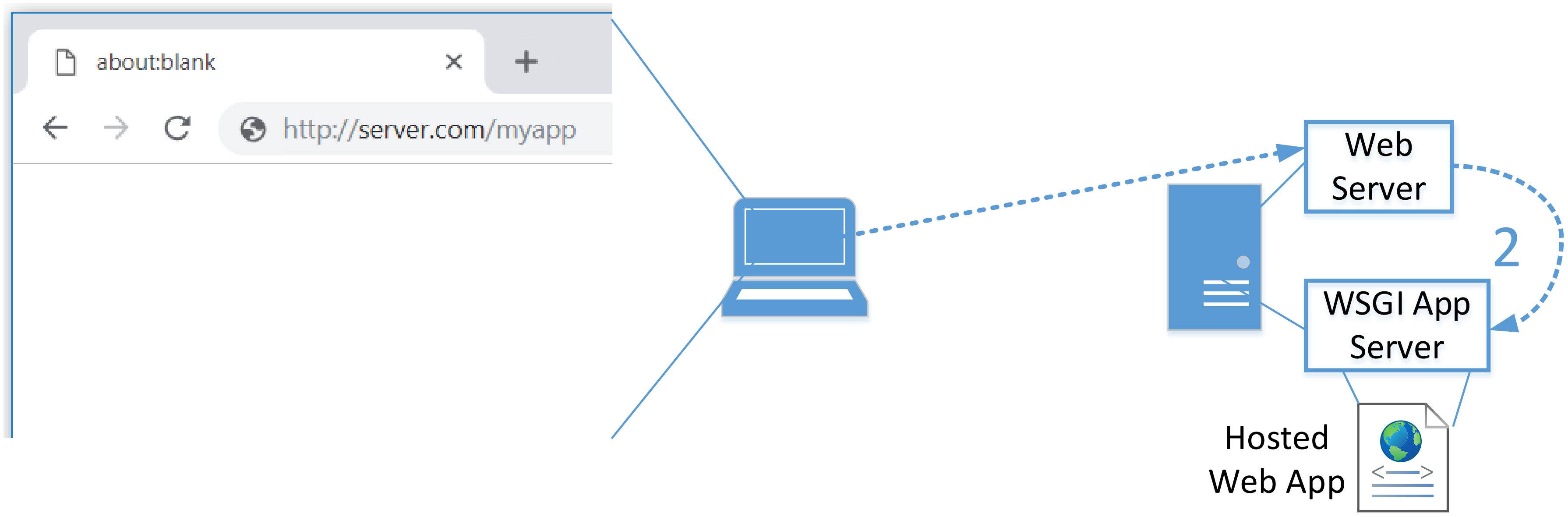
HOW DOES A WEB FRAMEWORK HELP?

- Clients (browsers) connect to web server using HTTP



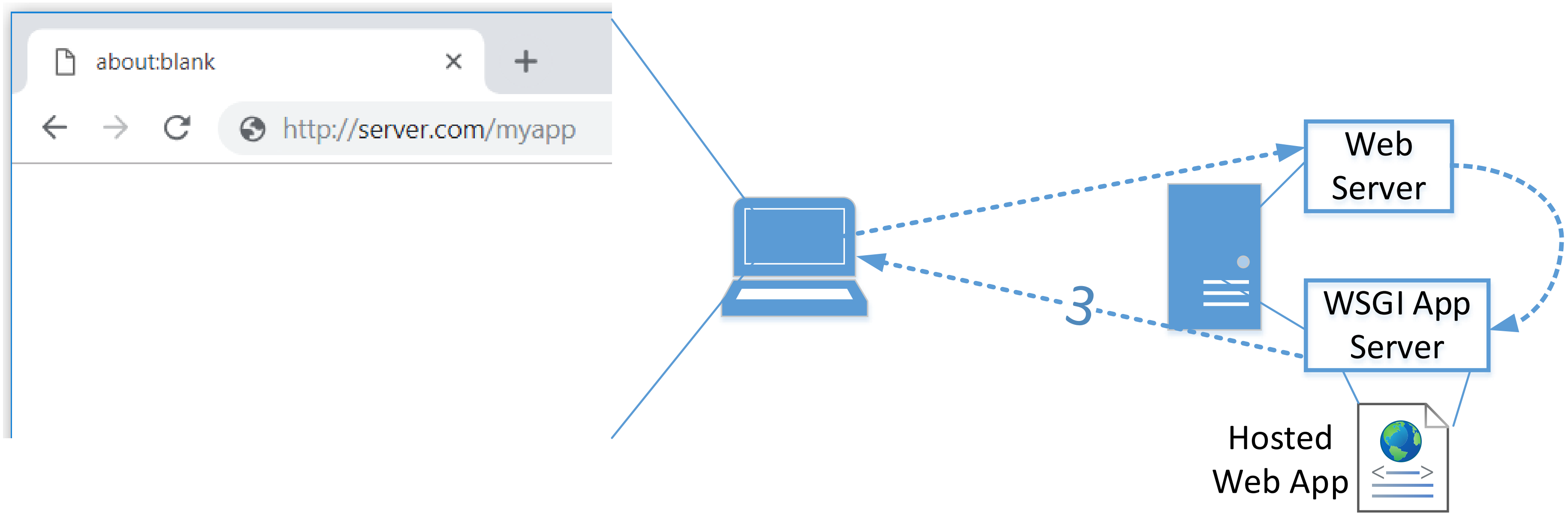
HOW DOES A WEB FRAMEWORK HELP?

- Server delegates requests to WSGI server



HOW DOES A WEB FRAMEWORK HELP?

- The web (application) framework works with WSGI server to handle requests/responses allowing the client to interact with the web application



WHAT DOES A WEB FRAMEWORK OFFER?

Frameworks have components to help process requests including:

Component	Description
Routing	Map URL to view function
Templates	Dynamic response/page creation
Input Forms	Processing/validation
ORM	Facilitate database interaction
Security	Protection from CSRF, XSS, ...
Sessions	Management/validation

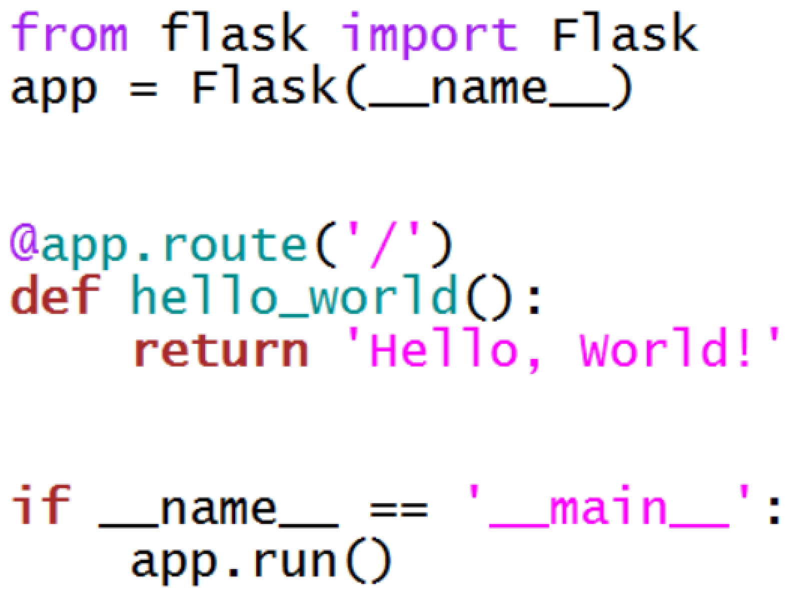
POPULAR PYTHON WEB FRAMEWORKS

- Django – batteries included, larger/complex
- ***Flask – choose your own adventure, smaller/simpler***
- Pyramid – middle ground, smaller to larger but myriad choices required
- Tornado – alternate, early async, can run without WSGI layer, targets “C10K”
- Sanic – newer and even faster async framework, requires Python 3.5+

FLASK OVERVIEW – ROADMAP

- Where Flask fits within the Web Ecosystem
- **Your first Flask application**
- Building a Flask application
- Next steps and interesting adjacent areas

FLASK HELLO WORLD



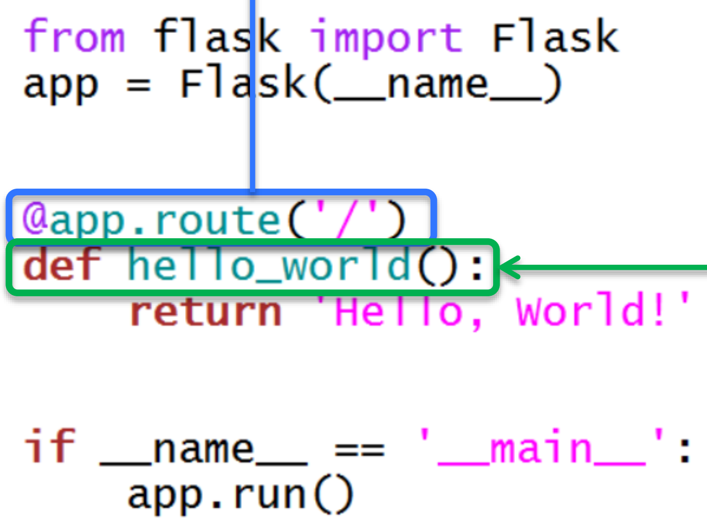
```python
from flask import Flask
app = Flask(__name__)


@app.route('/')
def hello_world():
    return 'Hello, World!'


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

```
Server> python hello.py
 * Serving Flask app "hello" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

ROUTING

- Flask receives inbound request – parses out URL path
- Flask maintains a table of URL paths and corresponding “view” functions:

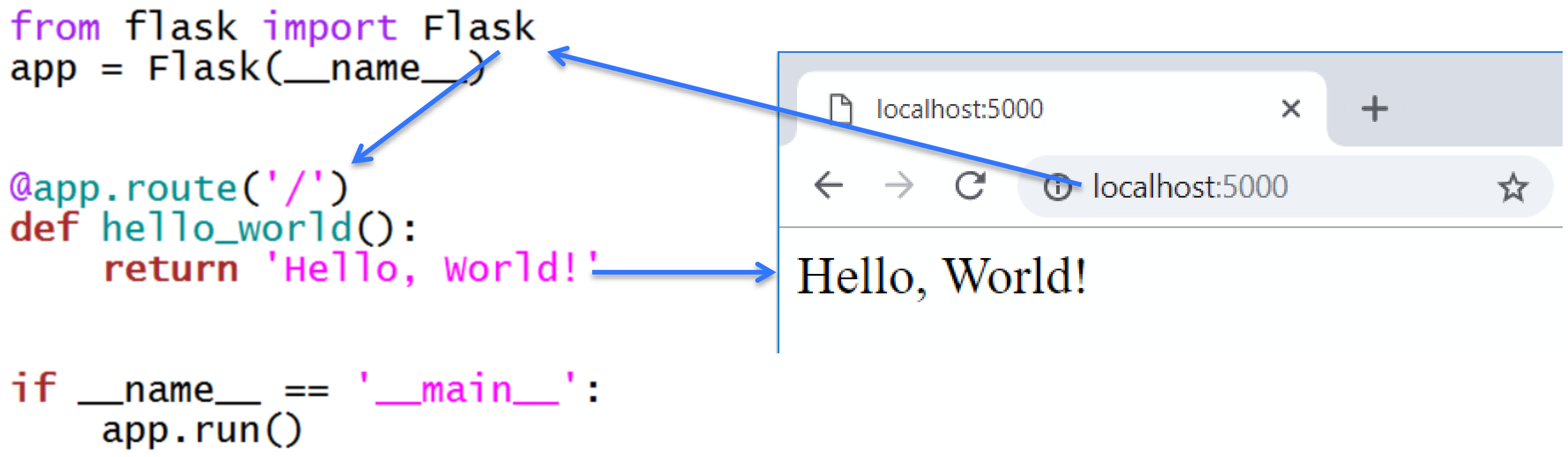
URL Path	View Function
“/”	hello_world
No entry above	Return 404 status



```
from flask import Flask
app = Flask(__name__)


@app.route('/')
def hello_world():
    return 'Hello, World!'


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

ROUTES AND VIEW FUNCTIONS

- Flask uses decorators to connect inbound request URL paths to view functions
- The view function is responsible for generating the content and returning it
- Flask uses this to build the response



```python
from flask import Flask
app = Flask(__name__)


@app.route('/')
def hello_world():
    return 'Hello, World!'


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

ASIDE ON SETUP

We'll use:

- Python 3.6 (3.7+ OK too)
- Python Virtual Environment
- Git
- GitHub to host the code

FLASK OVERVIEW – ROADMAP

- Where Flask fits within the Web Ecosystem
- Your first Flask application
- **Building a Flask application**
- Next steps and interesting adjacent areas

SETUP ENVIRONMENT

- Python 3.6/3.7 installed

- Retrieve application from GitHub:
  - » git clone https://github.com/sockduct/flaskapp.git
  - » cd flaskapp

- Create virtual environment
  - » python –m venv venv

- Activate virtual environment
  - » Windows: venv\scripts\activate
  - » Linux/macOS: source venv/bin/activate

GETTING STARTED

- Virtual Environment starts out with no packages
  - pip list will show just pip and setuptools
- First upgrade pip, then install flask:
  - python –m pip install --upgrade pip
  - pip install flask
- Start building the app
  - git checkout v0.1   # Look at app.py

```
#!/usr/bin/env python3

from flask import Flask


app = Flask(__name__)


@app.route('/')
def index():
    return '<h1>Welcome to the Journal!</h1>'
```

THINKING ABOUT OUR APP

- This app will be an online journal
- We need to allow users to register, login, logout, and create an entry

URL Path	View Function	Description
/	main	main page – shows all journal entries
/register	register	User sign up
/login	login	User login
/logout	logout	User logout
/entry	entry	Create journal entry
(other)	-	Return 404 status

PASSWORD STORAGE

- Flask is part of the Pallets Projects
- This collection of Python web development libraries comprises the Flask microframework

Library	Description
Click	Composable CLI
Flask	Flexible web framework
itsdangerous	Cryptographic signing
Jinja2	Template engine
Markupsafe	Safe HTML markup
Werkzeug	WSGI utility library

HOW NOT TO STORE PASSWORDS

- What’s wrong with storing plaintext passwords or secrets?

Google

Quora

T · · Mobile ·®

ebay

CHASE

EQUIFAX

Marriott®

HOTELS · RESORTS · SUITES

BRIEF ASIDE ON HASHING

- Hash Functions

  sha256('password') → '5e884898da2804715...42d8'

  » Changing one character completely alters the hash:

  sha256('passw0rd') → '8f0e2f76e22b43e285...34a9'

- Can't reverse hash output:

  '5e884898da2804715...42d8' $\not\rightarrow$ 'password'

- So is this a good way to store passwords and if so what's the best function to use?

HOW NOT TO STORE PASSWORDS

- OK, so we need to store the password hash. Python includes a hash library:

```
>>> import hashlib
>>> hashlib.algorithms_available  # 32 available!
{'shake_128', 'SHA224', 'sha3_384', 'shake_256', 'SHA256’, ...}
>>> hashlib.sha256(‘password’)
TypeError: Unicode-objects must be encoded before hashing
>>> hashlib.sha256(‘password’.encode(‘utf8’))
<sha256 HASH object @ 0x012B20E0>
>>> hashlib.sha256(‘password’.encode(‘utf8’)).digest()
b"^\x88H\x98\xda(\x04qQ\xd0\xe5o\x8d\xc6)..."
>>> hashlib.sha256('password'.encode('utf8')).hexdigest()
'5e884898da28047151d0e56f8dc6292773603d0...’
```

HOW NOT TO STORE PASSWORDS

- Is this a good way to store passwords?

  '5e884898da28047151d0e56f8dc6292773603d0...'

- No... Why not?
  - Susceptible to brute force decryption
  - Rainbow tables accelerate this

- What's the solution?
  - "salt"

HOW TO STORE PASSWORDS

- Storing passwords securely is hard
  - Don’t try to do it yourself!
  - Use a library designed for this purpose
  - Considerations
    - OWASP Password Storage Cheat Sheet
    - OWASP has recommended libraries

- Recommendation for Flask
  - Use werkzeug’s security helper functions
  - These use an OWASP recommended library
  - These use reasonable defaults for the myriad knobs

PASSWORD HASH GENERATE WALK THROUGH

```
>>> from werkzeug.security import check_password_hash,
generate_password_hash

>>> generate_password_hash('Top-Secret!')
'pbkdf2:sha256:50000$PtKDpmtg$b33fd2de254447205016608
9146b4498b25977c8bad40035dbacc2789b26a44b'

# Note – different salt = different password hash
>>> generate_password_hash('Top-Secret!')
'pbkdf2:sha256:50000$AGtcjyB8$33084c04370100d99a12581
5177d8e7a64b07e77bf41d174be8c91ca77b29989'

>>> check_password_hash(_, 'Top-Secret!')
True

# Password must exactly match
>>> check_password_hash(_, 'Top_Secret!')
False
```

HOW TO STORE TEST PASSWORD?

- Don’t want to add to source code
- Use environment variable
- Library designed to leverage this:
  - pip install python-dotenv
- Create .env file in application directory
  - TEST_PASSWD=pbkdf2:sha256:50000$PtKD...
- Update app.py to use

TEMPLATES

- App returning strings or simple markup from view functions
- Really want to return HTML page
- Doing this from Python script undesirable
- Templates let us use separate files for the HTML as well as inserting dynamic elements
  - » Separation of concerns

CREATE TEMPLATE FOR EACH URL PATH

- Flask expects a templates folder in the application directory
- Start with base template so don’t have to re-do boilerplate for each page
- Create a template for the main page
- /logout and /entry require context – we’ll defer those for now
- /login, /register, and /entry must process user input – we’ll cover that next

FORMS

- We could write our own code to process and validate user input
- Rather than re-inventing the wheel there are Flask extensions

Flask Extension	Description
Flask-WTF	Form handling
Flask-Login	User session management
<Many Others>	flask.pocoo.org/extensions

ASIDE ON USER INPUT

- Forms handle user input – an inherently dangerous task
- How do you know the input is valid? (In a minute…)
- How do you know the input is really from the user and not forged/spoofed?

ASIDE ON USER INPUT AND CSRF

- Make user input difficult to forge
- Cross-Site Request Forgery (CSRF or Sea-Surf) – common way to forge requests
- Attacker tricks a user into taking an action
  - » e.g., transferring all user's money into his bank account
- A good form handling library protects against CSRF by using hidden token

ASIDE ON ANTI-CSRF AND KEYED HASHES

- Creating hidden tokens to protect against CSRF requires a secret key

- A secret key is just like a password – if a human creates it, it's weak

- Why does it matter?

```
>>> import hmac, hashlib
>>> hmac.new('password'.encode('utf8’),
        'my data'.encode('utf8'), hashlib.sha256).hexdigest()
'64cbc073ae170d6db9c7203d18ce4a...bd52'
>>> hashlib.sha256('my data'.encode('utf8')).hexdigest()
'b2167b0aa7ef7794740b055ac7a880...b9de'
```

ASIDE – HOW SECRETS PROTECT COOKIES

- Let’s walk through using a cookie for session state:

```
>>> from werkzeug.contrib.securecookie import
SecureCookie as scookie
>>> mycookie = scookie({'userid': 1001,
                        'data': [1, 'apple', 3.5]}, 'password')
>>> mycookie.serialize()
b'Zili...VsT8=?data=gANd...BlLg==&userid=gANN6QMu'
>>> external_cookie = _
>>> scookie.unserialize(external_cookie, 'password')
<SecureCookie {'data': [1, 'apple', 3.5], 'userid': 1001}>
# Only works if not tampered with:
>>> scookie.unserialize(external_cookie + b' ', 'password')
<SecureCookie {}>
```

FORMS

- Install the Flask-WTF extension

- Using Templates
  - Create a secret key
  - Using hidden tokens – CSRF protection
  - Messaging users – flash
  - Rendering templates
  - The POST-Redirect-GET Pattern

USER SESSION MANAGEMENT

- Use the Flask-Login extension
- Leverage cookies for user sessions
- Support user login/logout
- Restrict some views to authenticated users only

ERROR HANDLING

- Users are experts at finding bugs!
- Need to deal with 404s (no such location)
- Need to deal with 500s (application error)
- A brief chat about debug mode

FLASK OVERVIEW – ROADMAP

- Where Flask fits within the Web Ecosystem
- Your first Flask application
- Building a Flask application
- **Next steps and interesting adjacent areas**

WHAT’S NEXT?

- Databases and ORMs (ODMs)
- HTML/CSS Styling
- Utilities – Logging, Email
- Interactive application – use JavaScript Frameworks/Libraries
- Deployment Options
- Blueprints to organize larger applications
- Unit Testing
- Beyond HTTP - Websockets

QUESTIONS

@sockduct

GitHub Repo

Appendix

ROADMAP – LINUX ADMINISTRATION

- Linux administration
  - » Most popular distro, especially for cloud is Ubuntu
  - » Ubuntu training and certification handled by LPI

- Certifications/Training Courses

LPI Course	Certification Exam(s)
Linux Essentials	LPI 010
DevOps Tools Engineer	LPI 701
LPIC-1 Certified Linux Administrator	LPI 101, 102
LPIC-2 Certified Linux Engineer	LPI 201, 202 + LPIC-1
LPIC-3 Linux Enterprise Pro, Mixed Env	LPI 300 + LPIC-2
LPIC-3 Linux Enterprise Pro, Security	LPI 303 + LPIC-2
LPIC-3 Linux Enterprise Pro, Virt + HA	LPI 304 + LPIC-2

ROADMAP – GETTING INTO PROGRAMMING

- Python development
  - Getting Started – Python for Everybody series via Coursera
  - Created by Charles Severance from U of M Ann Arbor
  - One of the highest rated online courses – over 50k ratings
  - Includes quizzes and peer reviewed assignments – critical!

- Alternate option
  - Udacity's Intro Programming Nanodegree
  - More expensive, but broader and includes more expert guidance and support
  - Projects reviewed by trained experts vs. students
  - In addition to Python also covers HTML, CSS, and JavaScript

ROADMAP – WEB DEVELOPMENT

- Web development
  - Udacity's Full Stack Web Developer Nanodegree is a tour de force of web development. You need prior experience (see previous getting into programming slide). However, completing this along with the projects will take you to the next level.
  - The Flask Megatutorial by Miguel Grinberg – the ultimate resource if want to dive deep with Flask.
  - Full Stack Python – Everything you could possibly want to know about the Full Stack from a Python point-of-view.

PYTHON PODCASTS

- Talk Python to Me, Michael Kennedy – Hour long interviews from notable members of the community about Python, Python Libraries, and interesting use cases
- Python Bytes, Michael Kennedy & Brian Okken – Half hour segment featuring six notable things from Python, Libraries, Community for last week
- Test and Code, Brian Okken – Varying segments about testing and interesting things in the Python community
- Podcast.__init__, Tobias Macey – Similar to Talk Python to me but hits different areas/people/projects