Week Four

CSOH| Python Study Group

Cloud Security Office Hours

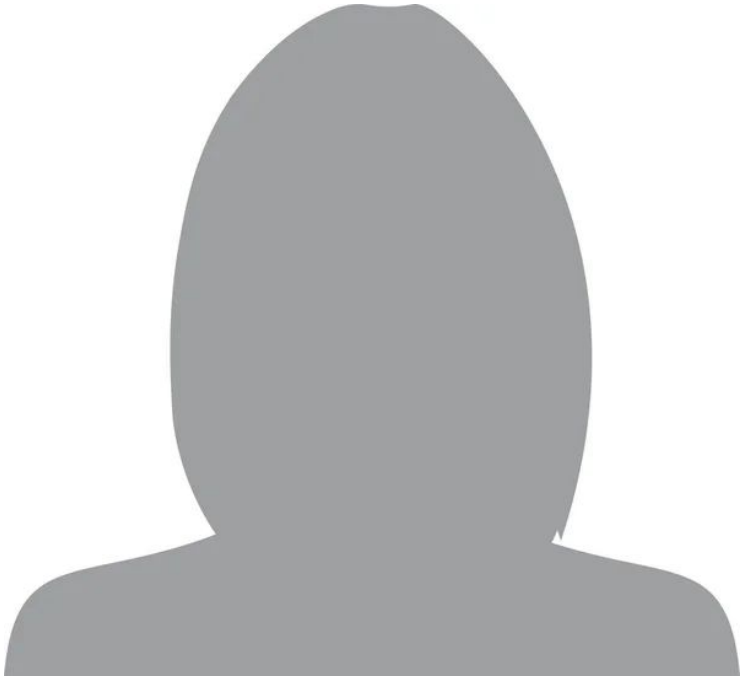
Thanks for the space!

Once a week, every Friday, we host a Zoom call that is an open forum for novices and experts in Cloud Security to collaborate and encourage each other in this field.

Fridays @ 10:00 AM
Same Zoom

https://csoh.org/

`id`



i am D (they/them) or Hetz

recent cybersecurity grad

interested in malware analysis and application security

not an expert, just an enthusiast tired of tutorial hell

and chihuahuas…lots of chihuahuas

Grounding Assumptions

1. **We're all here to learn.**
   Everyone starts somewhere, it's okay to ask questions and make mistakes. This is a space for us to work and learn together.

2. **No prior coding experience required.**
   We'll explain each project and ask our advanced group members for support, when needed.

3. **Python is forgiving.**
   Small errors happen! Syntax mistakes are part of learning how code "thinks."

4. **We'll focus on understanding, not memorizing.**
   Knowing *why* something works is more valuable than remembering every detail.

5. **Try it yourself.**
   The best way to learn coding is by typing, testing, and experimenting.

6. **Respectful collaboration.**
   Share ideas, help each other, and be patient- we're learning together.

7. **Keep it simple today.**
   We'll start with the basics and build up over time — no need to rush.

Schedule

1. Codespace Review (10 min)
2. Lesson Review (25 min)
3. Mini Project (30 min)
4. Show & Tell (25 min)

Focus & Objectives

Focus:

Learn the basics of Flask, Python's lightweight web framework.

Understand how a simple web server works and create a "Hello World" app with routes that respond to different URLs.

Contributers

Special Thanks!

Kyle Ingersoll - Managing the Github Workspace

Sam M. - Recording each session

Neil - Wisdom and admin powers

https://csoh.org/

Intro To Flask

Hello World!

What is Flask?

Lightweight web framework for serving content.

Each **route** is a URL endpoint connected to a function called **view**

Think of Flask as a receptionist: it listens for requests and hands them off to the right function for a response.

Intro To Flask

Getting Started

1. Make sure Python and `pip` are working.

2. Install Flask (in terminal):
   pip install flask

3. Verify:
   python -m flask --version

4. Create a new folder named `week4_flask` and open it in VS Code.
5. Do not name application flask.py

Flask Basics

Create a basic application

`@app.route("/")` maps a URL to a function.

The function returns what the browser will show.

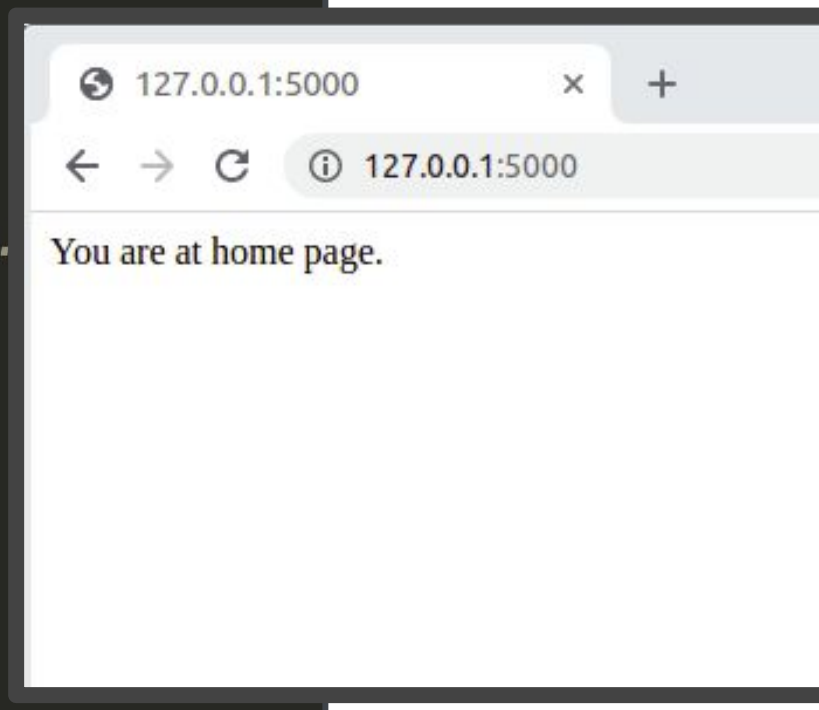
`debug=True or --debug` lets Flask reload automatically and show errors.

```python
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

The first argument is the name of the application's module or package. `__name__` is a convenient shortcut for this that is appropriate for most cases. This is needed so that Flask knows where to look for resources such as templates and static files

```
$ flask --app hello run
 * Serving Flask app 'hello'
 * Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
```

```python
#importing the flask Module
from flask import Flask

# Flask constructor takes the name of
# current module (__name__) as argument
app = Flask(__name__)

@app.route('/')
# '/' URL is bound with hello_world() function.
def home():
    return 'You are at home page.'

@app.route('/allow')
def allow():
    return 'You have been allowed to enter.'

@app.route('/disallow')
def disallow():
    return 'You have not been allowed to enter.'

# main driver function
if __name__ == '__main__':
    # run() method of Flask class runs the application
    # on the local development server.
    app.run()
```



[Geeks For Geeks Source]

Dynamic Routes

Building Dynamic URLs

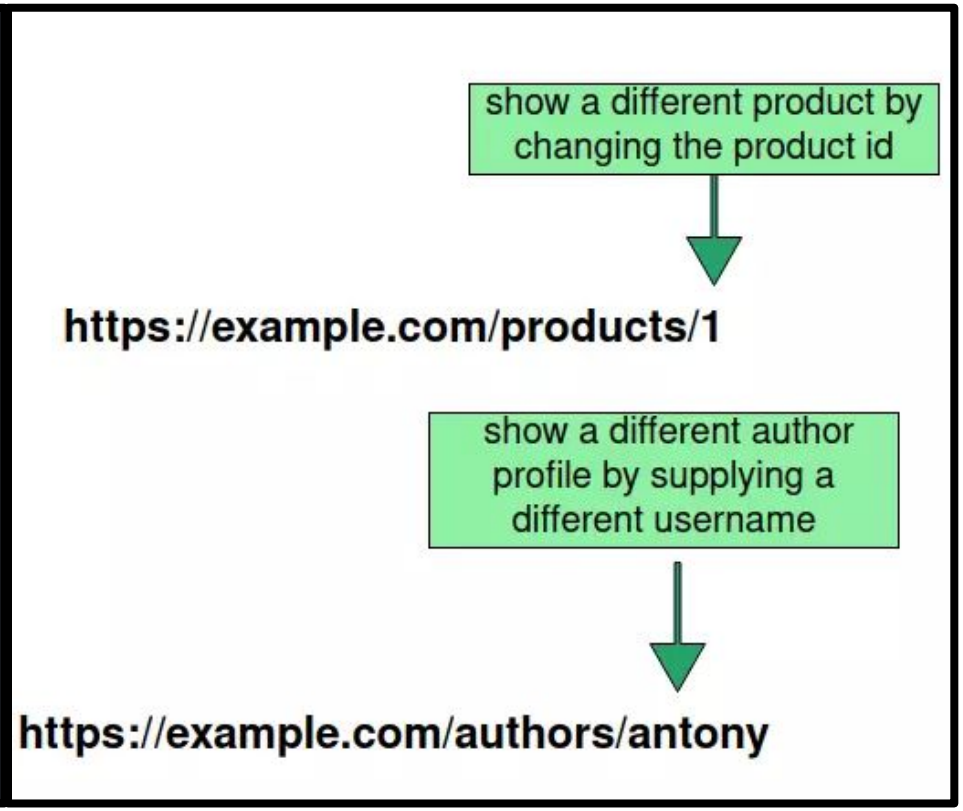
```
@app.route('allow/<variable name>')

OR

@app.route('allow/<converter: variable name>')
```

- Add a <variable name> with each route **OR**
- Define the converter with each variable name <converter: variable name>

String(default) - accepts text without a slash
int- accepts positive integers
float- accepts positive floating point values
path- like string but also accepts slashes
uuid- accepts UUID strings

This is how Flask handles user input through URLs. It is powerful for building search or report pages.

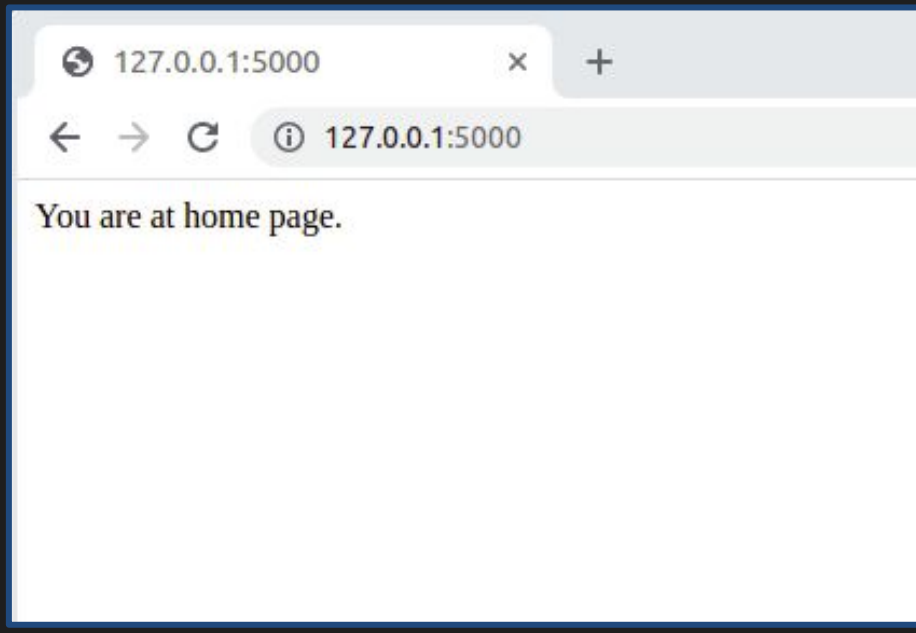
```python
from flask import Flask

app = Flask(__name__)

@app.route('/users/<username>')
def show_user_profile(username):
    return f'User: {username}'

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



[Flask URL Routing]

Geeks For Geeks Source

```python
#importing the flask Module
from flask import Flask

# Flask constructor takes the name of
# current module (__name__) as argument
app = Flask(__name__)

@app.route('/')
# '/' URL is bound with hello_world() function.
def home():
    return 'You are at home page.'

# Use of <converter: variable name> in the
# route() decorator.
@app.route('/allow/<int:Number>')
def allow(Number):
    if Number < 25:
        return f'You have been allowed to enter because your number is {str(Number)}'
    else:
        return f'You are not allowed'

# main driver function
if __name__ == '__main__':
    # run() method of Flask class runs the application
    # on the local development server.
    app.run()
```



```python
from flask import Flask, url_for

app = Flask(__name__)

@app.route('/')
def index():
    return "This is a basic flask application"

@app.route('/authors/<username>')
def show_author(username):
    return f"Return the author profile of {username}"

@app.route('/post/<int:post_id>/<slug>')
def show_post(post_id):
    return f"Showing post with ID: {post_id}"

if __name__ == '__main__':
    with app.test_request_context():
        # Generate URLs using url_for()
        home_url = url_for('index')
        profile_url = url_for('show_author', username='antony')
        post_url = url_for('show_post', post_id=456, slug='flask-intro' )

        print("Generated URLs:")
        print("Home URL:", home_url)
        print("Author profile URL:", profile_url)
        print("Post URL:", post_url)
```

Jinja

Jinja

Jinja allows developers to embed Python-like syntax directly within HTML files, creating dynamic web pages.

render_template(): Flask's render_template() function is used to render Jinja templates.

- Jinja is a fast, expressive, extensible templating engine.
- Special placeholders in the template allow writing code similar to Python syntax. Then the template is passed data to render the final document.
- Flask renders HTML files using Jinja2 templates by default.

Jinja

Jinja

```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def index():
    user_name = "Alice"
    return render_template('index.html', name=user_name)
```

Get started on templating

Code:

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def home():
    return render_template("index.html")
```

Templates keep our code clean and let us separate design from logic.This becomes important when building dashboards or security alert pages.

[Get started on templating]

index.html:

```
<!doctype html>
<html>
  <head><title>Flask Hello World</title></head>
  <body>
    <h1>Hello Flask Template!</h1>
    <p>This is rendered from an HTML file.</p>
  </body>
</html>
```

In Summary

- Flask creates web servers with minimal code.
- Routes map URLs to functions.
- Flask can serve HTML directly or via templates.
- Dynamic URLs make your app interactive.

resources

- Flask Official Docs — Quickstart
- Real Python — Flask by Example
- W3Schools — Flask Tutorial

Projects (Suggestions)

- Experiment with `render_template` to create a simple HTML page.
- Build a simple Flask web app with 3 routes
- Add a new route `/logs` that shows fake log entries (list from Python).
- Task Manager: Use a simple Python list or dictionary to store tasks in memory for initial development.
- Build the beginnings of an Event Management system

Optional Challenges

1. Regex Routing
2. Build a Web Scraper
3. Validate CAPTCHA
4. Implement ChatGPT

Share your work, if you'd like!