

Server-Side Web Programming: Python (Part 2)

Copyright © 2025 by
Robert M. Dondero, Ph.D
Princeton University

Objectives

- We will cover:
 - Web app frameworks
 - The Flask web app framework
 - The Jinja2 template engine
 - The MVC architecture
 - HTTP in COS 333

Agenda

- **Web application frameworks**
- The Flask web application framework
- The Jinja2 template engine
- The MVC architecture
- HTTP in COS 333

Web App Frameworks

- Old approach to building a website:
 - Write server-side code from scratch
 - Write client-side code from scratch

Web App Frameworks

- **Problem:**
 - Replicated code
- **Solution:**
 - Server side web app frameworks
 - Client-side web app frameworks
- Web app frameworks mechanize (parts of) the development process

Web App Frameworks

Some popular server-side web app frameworks:

Rank	Framework	GitHub Stars
1	Laravel (PHP)	74.9K
2	Django (Python)	73.4K
3	Flask (Python)	64.5K
4	Express (JavaScript)	62.0K
5	Ruby on Rails (Ruby)	53.6K
6	Spring (Java)	53.3K
7	Meteor (JavaScript)	43.8K
8	Symfony (PHP)	28.7K

According to GitHub as of 10/4/23

Web App Frameworks

- Web app framework assessment
 - (pro) Yield reliable code
 - (pro) Yield consistent code
 - (pro) Make efficient use of programmer time
 - (con) Can yield systems that are larger & slower than necessary

Agenda

- Web application frameworks
- **The Flask web application framework**
- The Jinja2 template engine
- The MVC architecture
- HTTP in COS 333

The Flask Web App Framework



Armin
Ronacher

The Flask Web App Framework

- Why study Flask?
 - (Instead of some other Python framework)
 - Easy to learn
 - Popular in general
 - Reasonable (and required) for COS 333 assignments
 - Popular for COS 333 projects

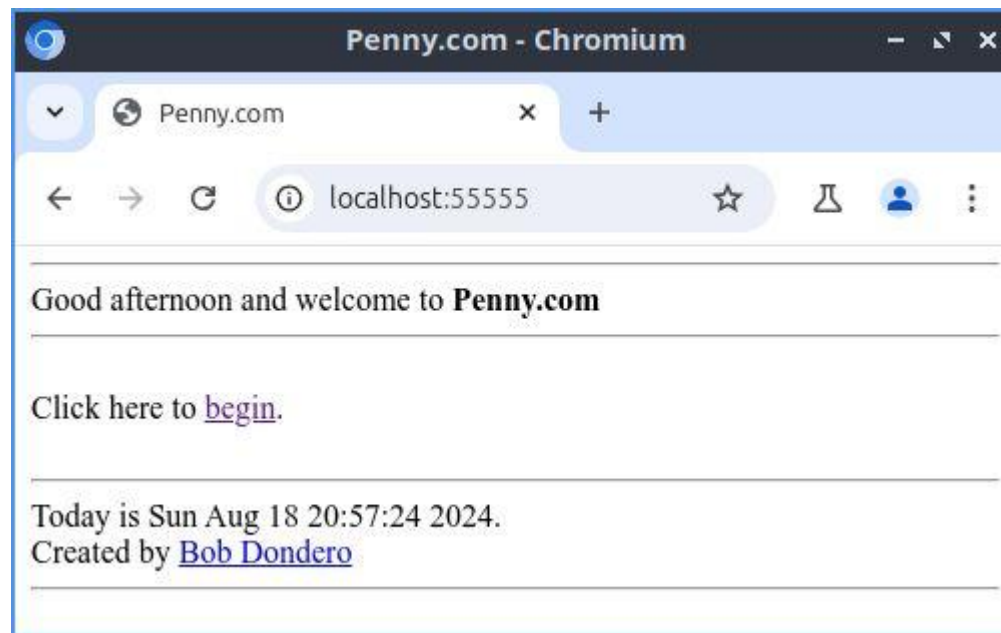
The Flask Web App Framework

- See **PennyFlask** app

```
$ python runserver.py 55555
* Serving Flask app 'penny'
* Debug mode: on
WARNING: This is a development server. Do not use it in a
production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:55555
* Running on http://192.168.1.10:55555
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 457-747-552
```

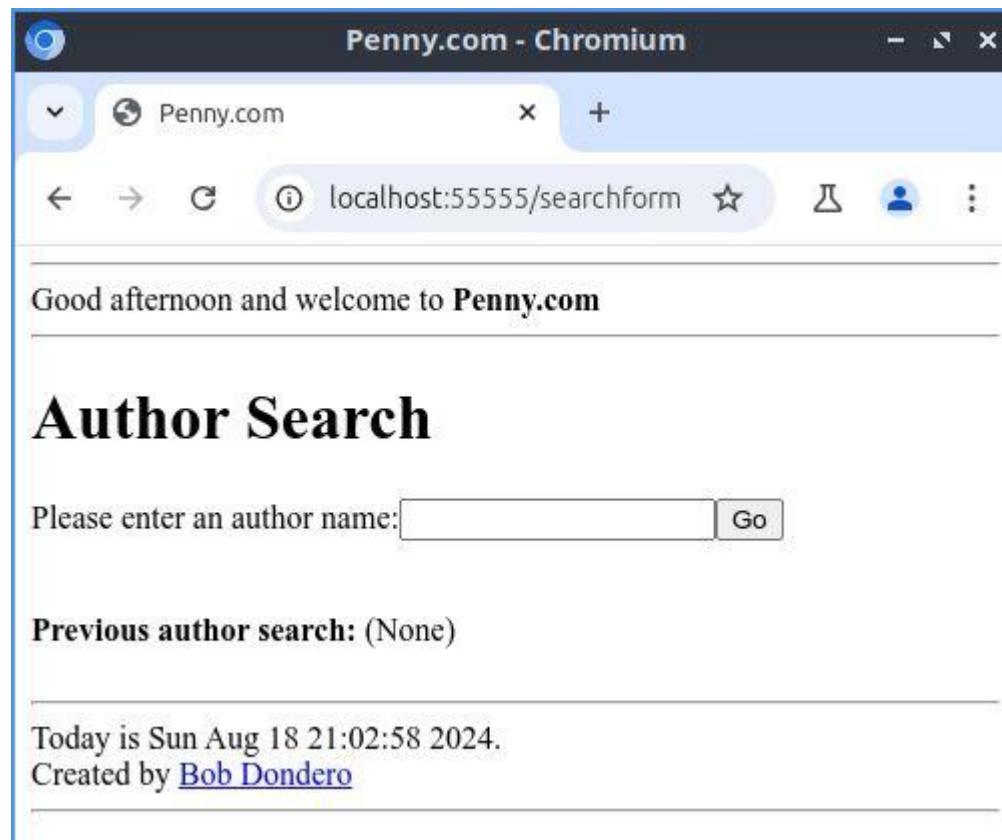
The Flask Web App Framework

- See **PennyFlask** app



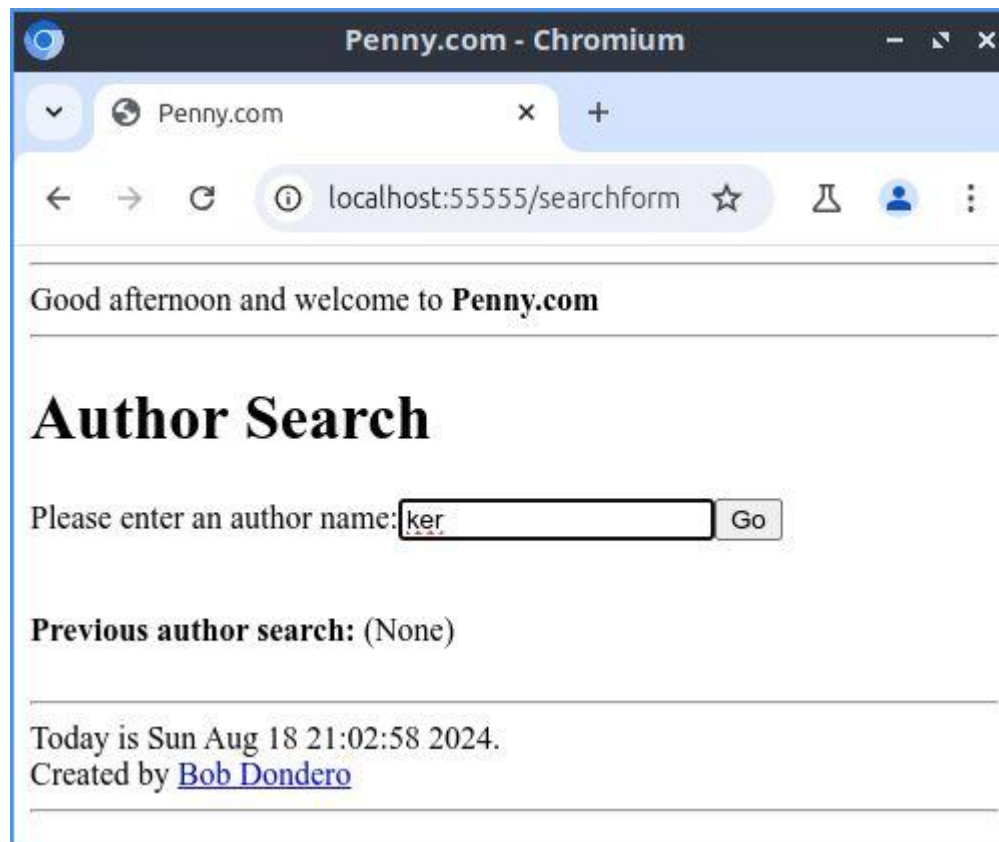
The Flask Web App Framework

- See **PennyFlask** app



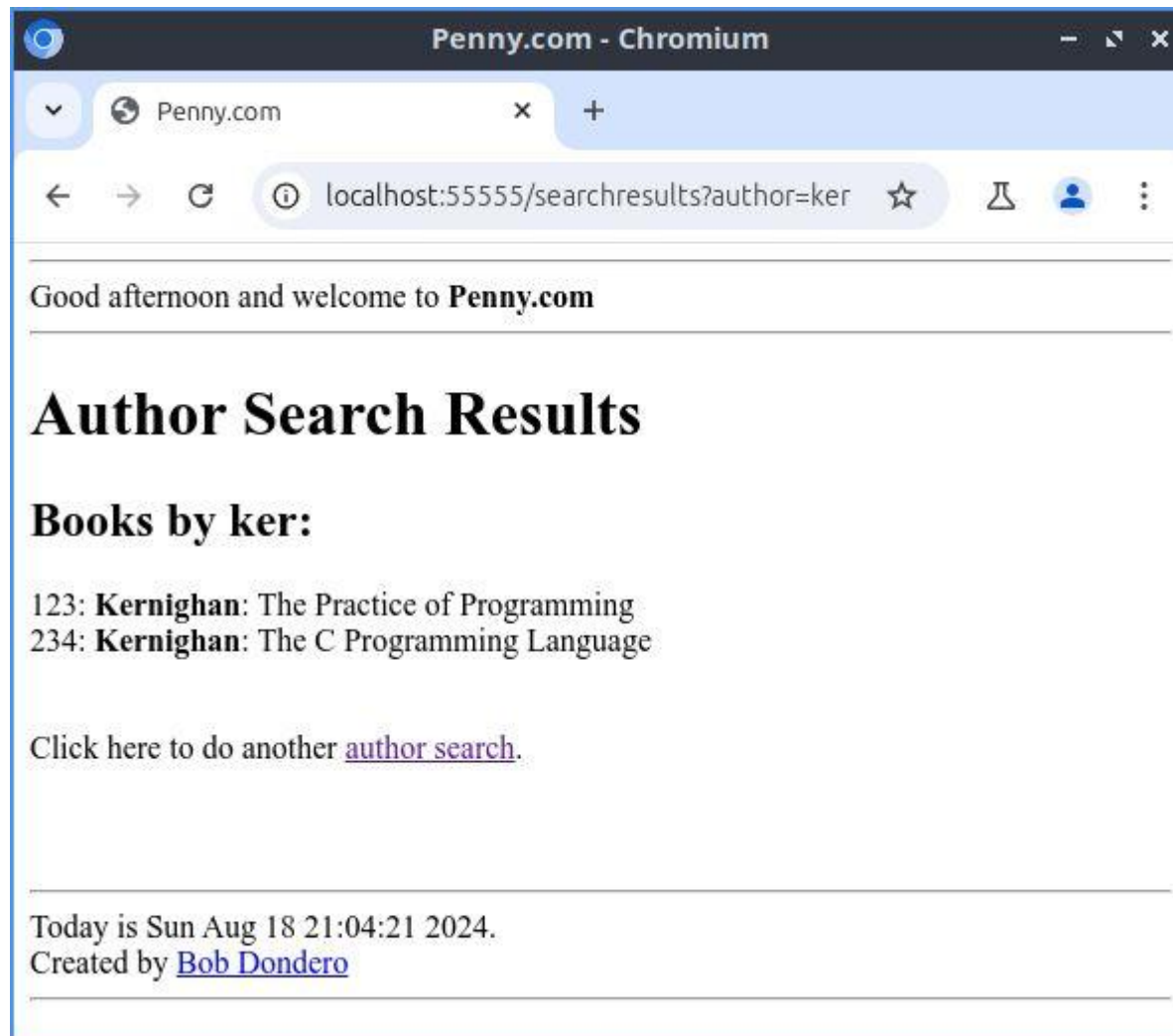
The Flask Web App Framework

- See **PennyFlask** app



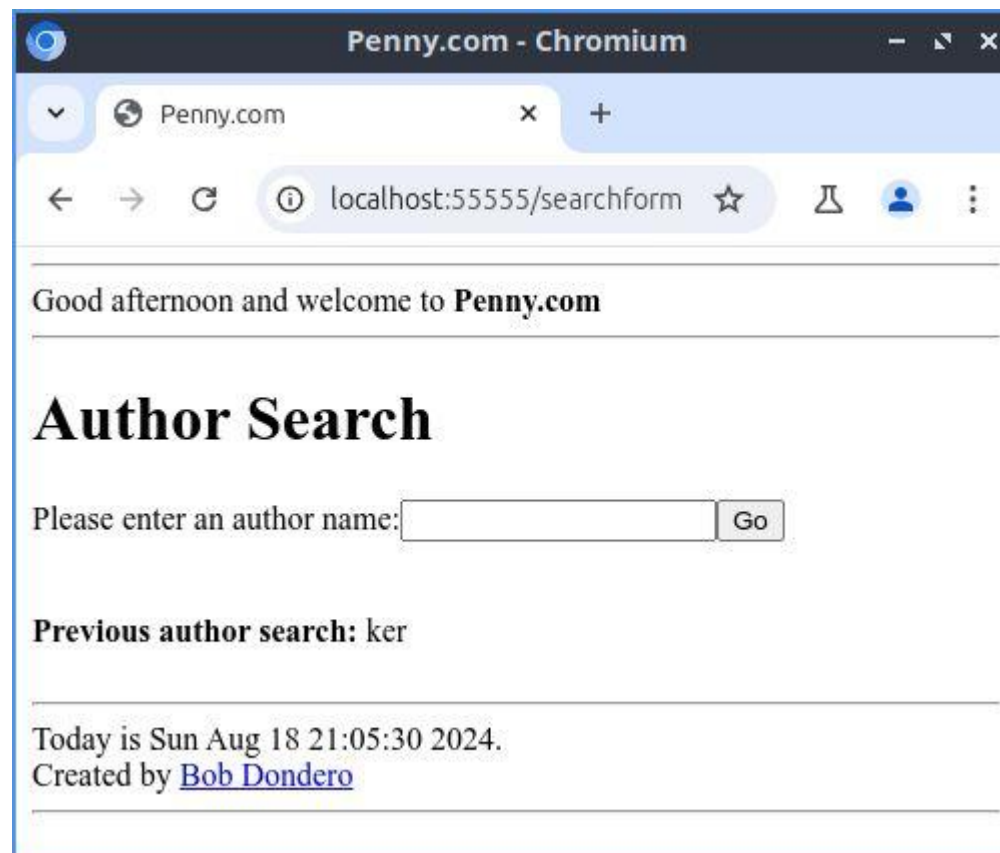
The Flask Web App Framework

- See **PennyFlask** app



The Flask Web App Framework

- See **PennyFlask** app



The Flask Web App Framework

- See **PennyFlask** app (cont.)
 - **runserver.py**
 - penny.sql
 - penny.sqlite
 - database.py
 - common.py
 - **penny.py**

Aside: Python Decorators

Flask uses Python *decorators*

With a decorator

```
...  
@app.route('/searchform')  
def search_form():  
    ...  
...
```

Without a decorator

```
...  
def search_form():  
    ...  
app.add_url_rule('/', 'searchform', search_form)  
...
```

See **Appendix 1** for explanation

Agenda

- Web application frameworks
- The Flask web application framework
- **The Jinja2 template engine**
- The MVC architecture
- HTTP in COS 333

The Jinja2 Template Engine

- **Problem:**
 - PennyFlask app builds HTML code strings incrementally
- **Solution:**
 - *Jinja2 template engine*

The Jinja2 Template Engine

- Python
 - Mustache, CheetahTemplate, Django, Genshi, **Jinja2**, Kid, Topsite, ...
- JavaScript
 - **Mustache**, Squirrelly, Handlebars, ...
- Java
 - **Mustache**, FreeMarker, Hamlets, Tiles, Thymeleaf, WebMacro, WebObjects, Velocity, ...

https://en.wikipedia.org/wiki/Web_template_system

The Jinja2 Template Engine

- See **PennyFlaskJinja** app
 - runserver.py
 - penny.sql, penny.sqlite
 - database.py
 - **header.html, footer.html**
 - **index.html**
 - **searchform.html, searchresults.html**
 - **penny.py**

The Jinja2 Template Engine

- Jinja2 *template*
 - HTML doc with *placeholders*
 - Each placeholder can contain:
 - A Jinja2 expression
 - A Jinja2 statement

The Jinja2 Template Engine

- Jinja2 *expression*
 - Similar to a Python expression
 - Uses Jinja2 *variables*
 - Examples:

```
... {{prev_author}} ...
```

```
... {{book['author']}} ...
```


The Jinja2 Template Engine

- Jinja2 *render_template* function
 - Accepts a template to be rendered
 - Accepts the values of Jinja2 variables used in that template
 - Returns a str object
 - Example:

```
html =  
render_template('sometemplate.html',  
                var1=value1, var2=value2, ...)
```

The Jinja2 Template Engine

- Jinja2 *statement*
 - Similar to a Python statement
 - Some differences:
 - Ignores indentation; blocks closed with end statements
 - Uses *filters* instead of builtins functions
 - Example:

Jinja
filter

```
{% if books|length == 0: %}  
    ...  
{% else: %}  
    {% for book in books: %}  
        ...  
    {% endfor %}  
{% endif %}
```

The Jinja2 Template Engine

- Jinja2 *include*
 - Includes one Jinja2 template into another
 - Example:

```
...  
{% include 'header.html' %}  
...
```

Agenda

- Web application frameworks
- The Flask web application framework
- The Jinja2 template engine
- **The MVC architecture**
- HTTP in COS 333

The MVC Architecture

- The ***Model-View-Controller (MVC)*** architecture
 - ***Model***: data access code
 - ***View***: data presentation code
 - ***Controller***: business code

The MVC Architecture

- Positives
 - Facilitates **separation of concerns**
 - Facilitates parallel development
 - Facilitates maintenance
- Suggestion: Use MVC!

The MVC Architecture

- Note:
 - PennyFlaskJinja implements MVC imperfectly

The MVC Architecture

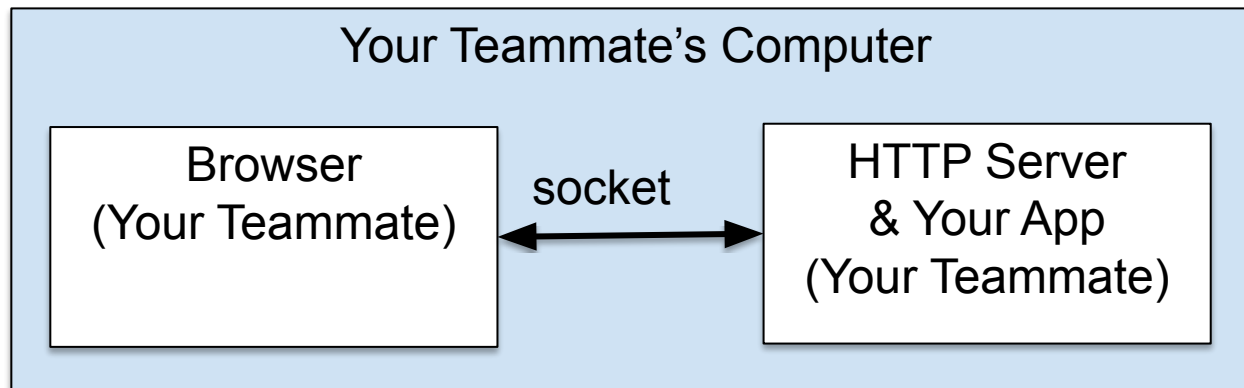
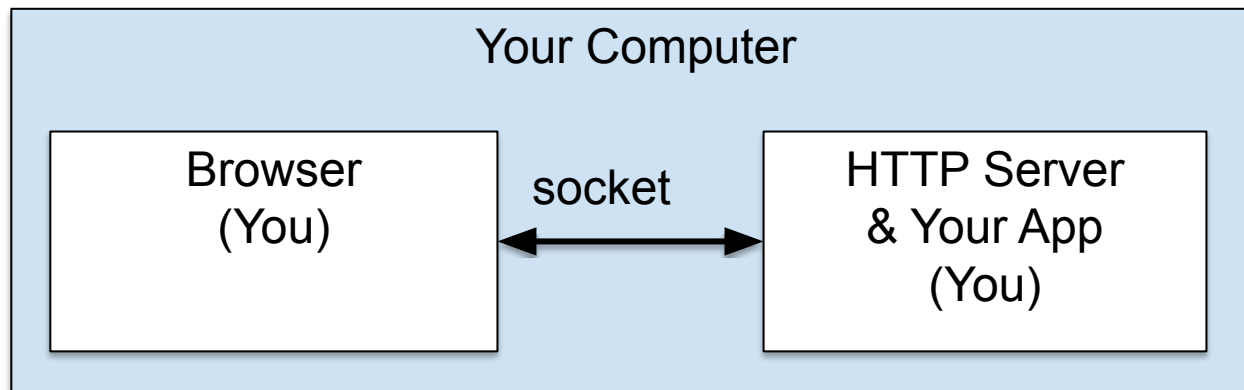
- Flask/Jinja2 suggestions:
 - Use MVC via Jinja2 templates!!!
 - But keep views simple

Agenda

- Web application frameworks
- The Flask web application framework
- The Jinja2 template engine
- The MVC architecture
- **HTTP in COS 333**

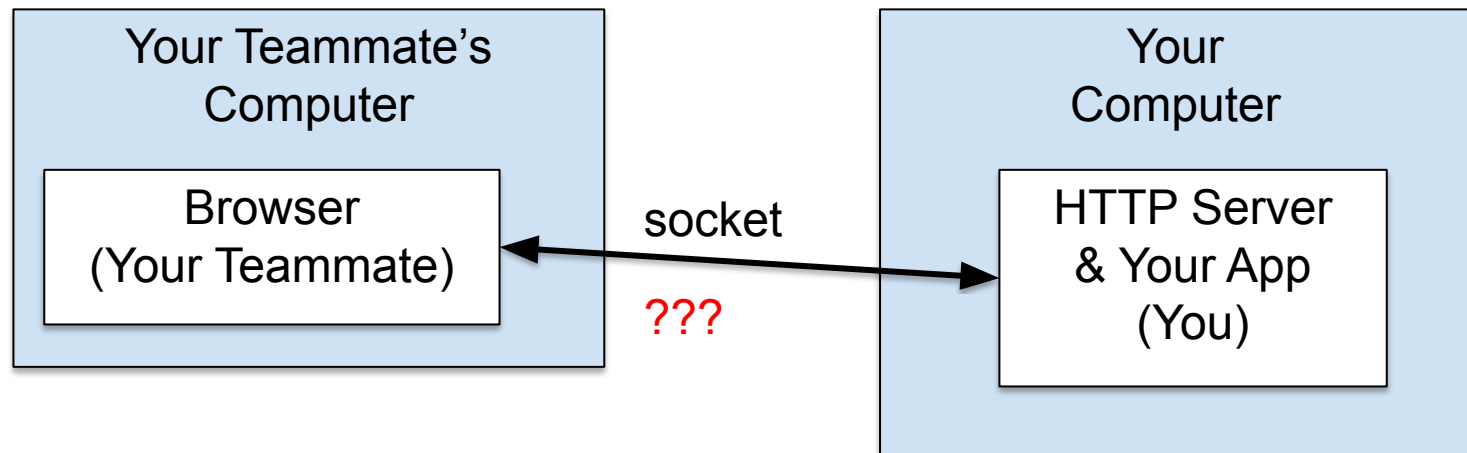
HTTP in COS 333

Option 1: Run HTTP server on local computer
Run browser on same local computer



HTTP in COS 333

Option 2: Run HTTP server on local computer
Run browser on different local computer



Won't work if either computer is not on Eduroam

HTTP in COS 333

- Suggestion:
 - Use **option 1** during development
 - Use **option 2** to test network comm
 - Feel free to see an instructor if working alone

For More Information

- There is much more to Flask and Jinja2
- Flask documentation:
 - <https://flask.palletsprojects.com/en/2.0.x/api/>
- Flask tutorial:
 - <https://www.tutorialspoint.com/flask/>
- Jinja2 documentation:
 - <https://jinja.palletsprojects.com/en/3.0.x/>

Summary

- We have covered:
 - Web app frameworks
 - The Flask web app framework
 - The Jinja2 template engine
 - The MVC architecture
 - HTTP in COS 333

Summary

- We have covered:
 - Python WSGI programming
 - Web app frameworks
 - The Flask web app framework
 - The Jinja2 template engine
 - The MVC architecture
- See also:
 - **Appendix 1: Python Decorators**
 - **Optional lecture: Django**

Appendix 1: Python Decorators

Python Decorators

```
def sqr(i):  
    return i * i  
  
def main():  
    result = sqr(5)  
    print(result)  
  
if __name__ == '__main__':  
    main()
```

Wanted:

sqr() prints “sqr was called” each time it is called

Python Decorators

```
def sqr(i):  
    print('sqr was called')  
    return i * i  
  
def main():  
    result = sqr(5)  
    print(result)  
  
if __name__ == '__main__':  
    main()
```

OK, but...

Requires edit of def of sqr()

Python Decorators

One approach

```
def print_name_decorator(f):
    def fwrapper(i):
        print(f.__name__, 'was called')
        return f(i)
    return fwrapper

def sqr(i):
    return i * i

sqr = print_name_decorator(sqr)
# Defines fwrapper as this:
#     def fwrapper(i):
#         print('sqr', 'was called')
#         return sqr(i)
# and then does this:
#     sqr = fwrapper

def main():
    result = sqr(5)
    print(result)

if __name__ == '__main__':
    main()
```

Trace:

```
result = sqr(5)
result = fwrapper(5)
    fwrapper(5) prints 'sqr was called'
    fwrapper(5) returns sqr(5)
result = 25
```

Prints:

```
sqr was called
25
```

Python Decorators

```
def print_name_decorator(f):  
    def fwrapper(i):  
        print(f.__name__, 'was called')  
        return f(i)  
    return fwrapper
```

```
@print_name_decorator
```

```
def sqr(i):  
    return i * i
```

```
def main():  
    result = sqr(5)  
    print(result)
```

```
if __name__ == '__main__':  
    main()
```

Using a
decorator

Prints:

```
sqr was called  
25
```