Server-Side Web Programming: Python (Part 2)

Robert M. Dondero, Ph.D
Princeton University

Objectives

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

Agenda

- **Web application frameworks**
- The Flask web application framework
- Separation of concerns
- The Jinja2 template engine
- 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:

Framework	GitHub Stars
Django (Python)	85.3K
Laravel (PHP)	82.3K
Spring-Boot (Java)	78.6K
Nest (JavaScript)	72.9K
Flask (Python)	70.5K
Express (JavaScript)	67.9K
Spring (Java)	58.9K
Rails (Ruby)	57.7K
Meteor (JavaScript)	44.7K
Fiber (Go)	38.0K

According to GitHub as of 10/5/25

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**
- Separation of concerns
- The Jinja2 template engine
- 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 for COS 333 projects

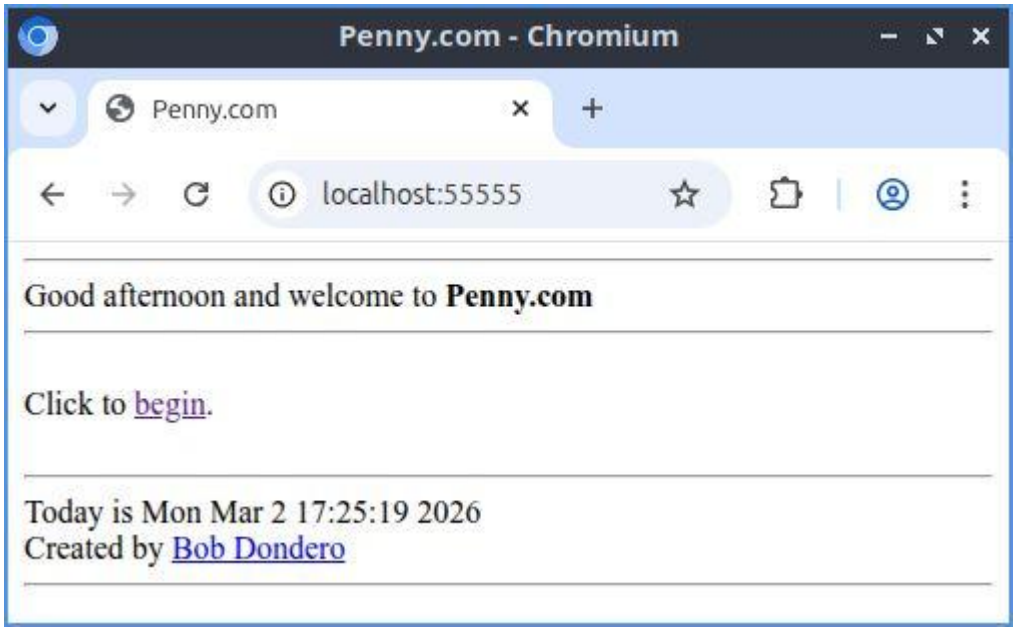
The Flask Web App Framework

- 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
```

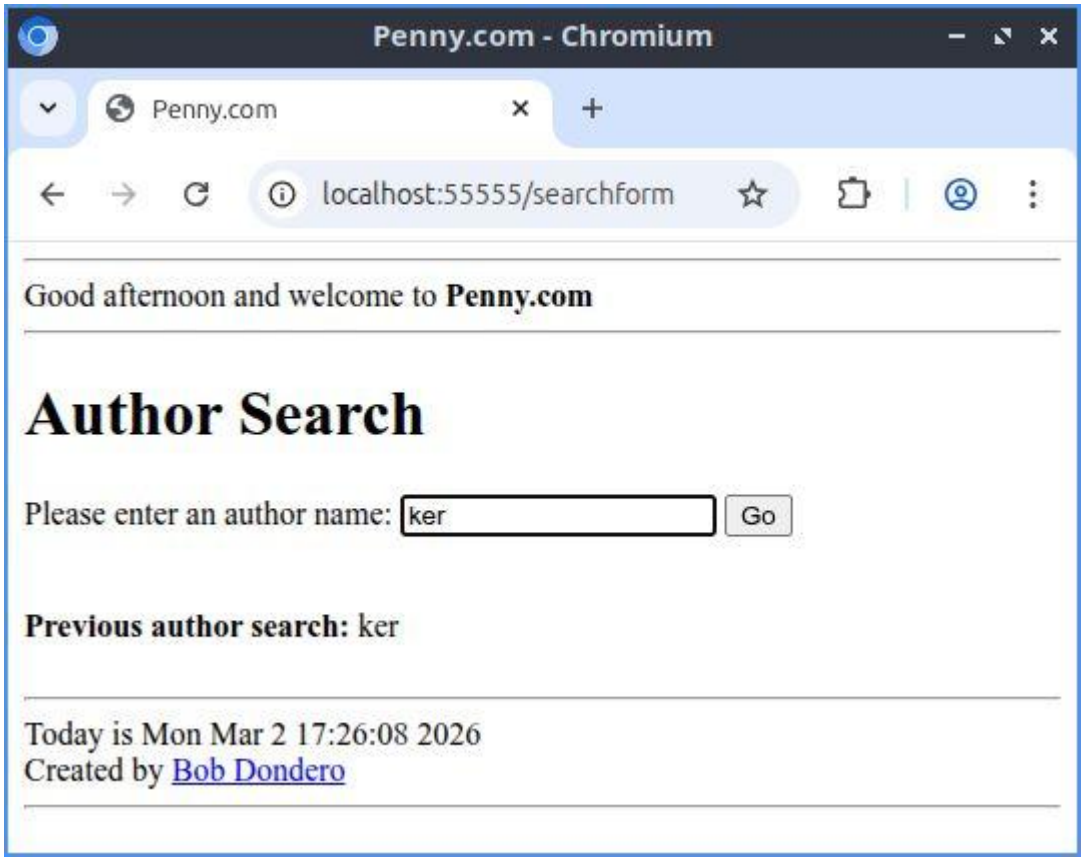
The Flask Web App Framework

- See **PennyFlask** app



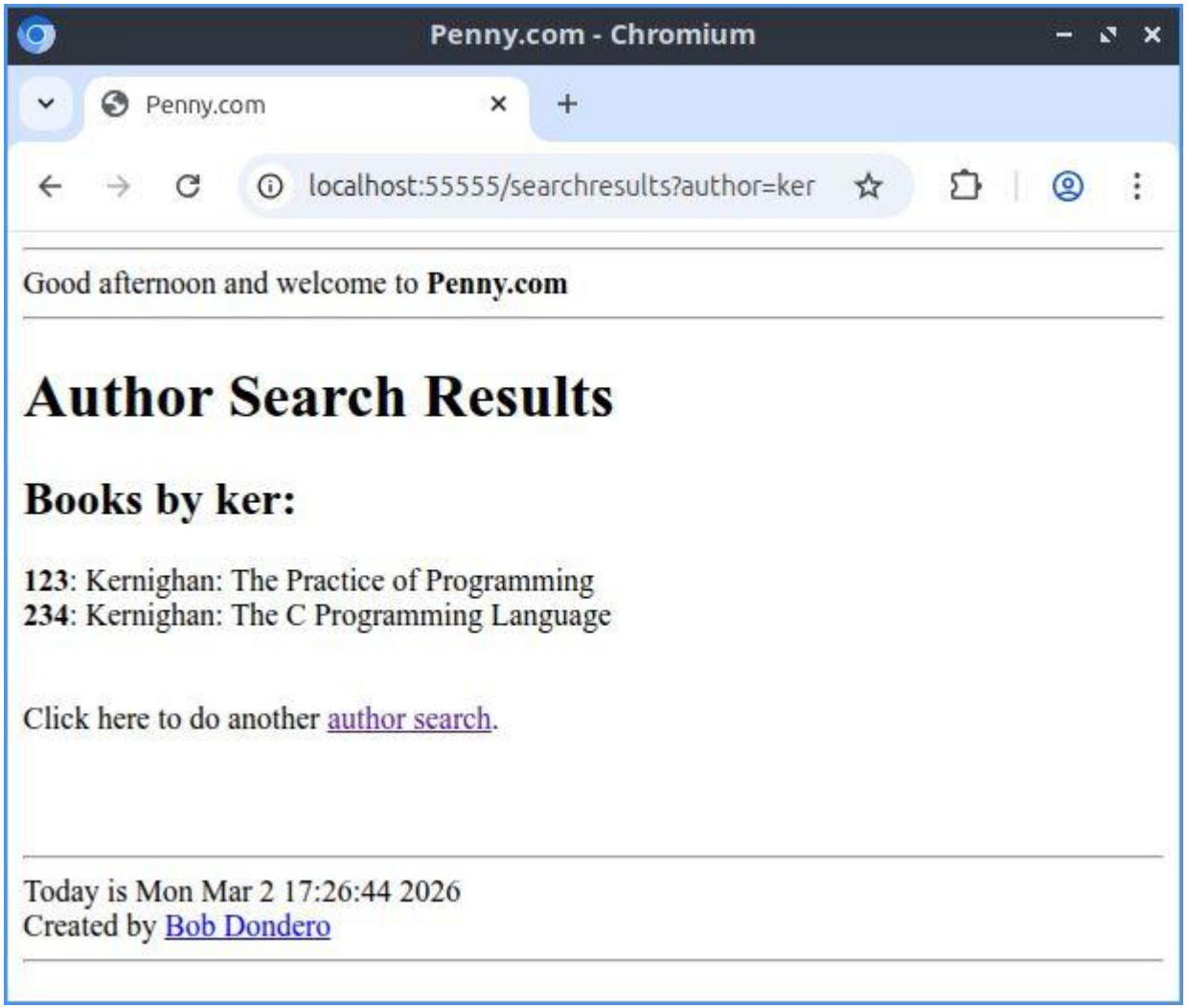
The Flask Web App Framework

- See **PennyFlask** app



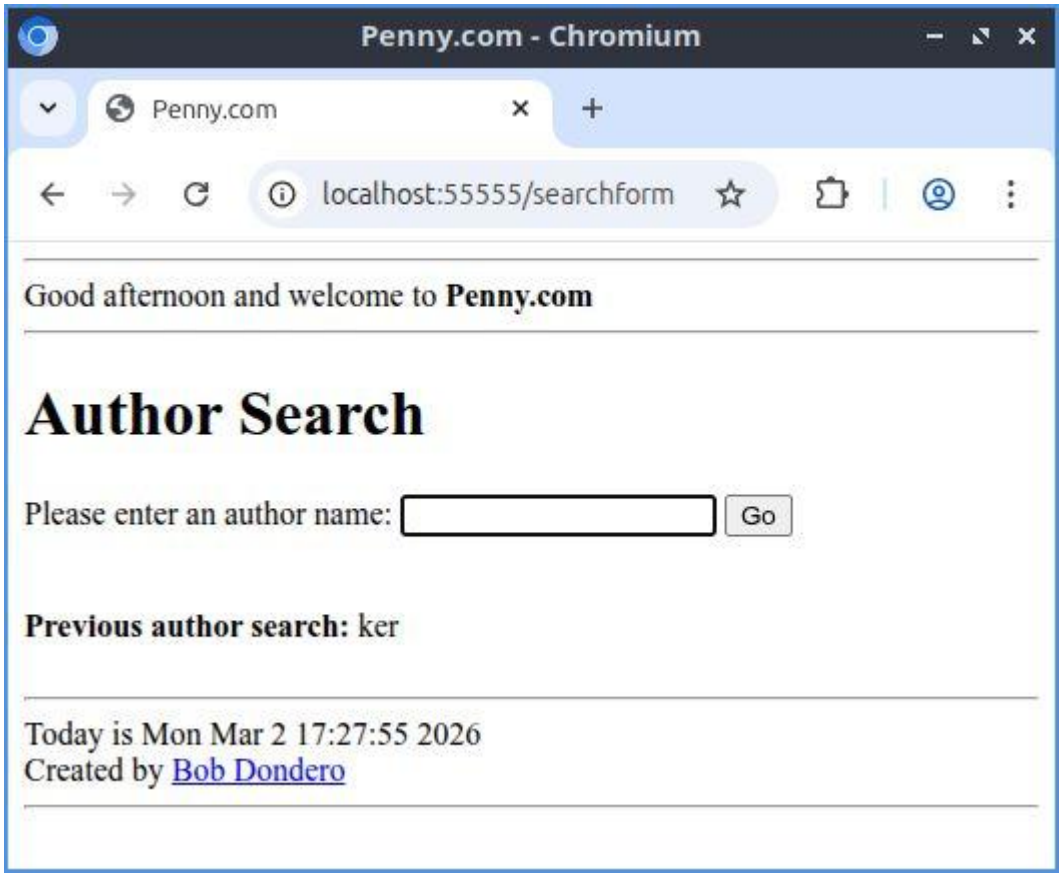
The Flask Web App Framework

- See **PennyFlask** app



The Flask Web App Framework

- See **PennyFlask** app



The Flask Web App Framework

- See **PennyFlask** app (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
- **Separation of concerns**
- The Jinja2 template engine
- HTTP in COS 333

Separation of Concerns

- **Problem**:
  - PennyFlask mixes Python code and HTML code
  - Poor separation of concerns
- **Solution 1:**
  - Python string format() method

Separation of Concerns

Python string format() method

```
school = 'Princeton'
enrollment = 5826
…
'The school {:s} has Undergraduate enrollment {:d}'
    .format(school, enrollment)
…
'The school {} has Undergraduate enrollment {}'
    .format(school, enrollment)
…
'The school {sch} has Undergraduate enrollment {enr}'
    .format(enr=enrollment, sch=school)
…
```

Separation of Concerns

- See **PennyFlaskTemplates** app
  - runserver.py
  - penny.sql, penny.sqlite
  - database.py
  - ~~common.py~~
  - **templates.py**
  - **penny.py**
- Note the imperfection

Agenda

- Web application frameworks
- The Flask web application framework
- Separation of concerns
- **The Jinja2 template engine**
- HTTP in COS 333

Separation of Concerns

- **Problem**:
  - PennyFlask mixes Python code and HTML code
  - Poor separation of concerns
- **Solution 1:**
  - Python string format() method
- **Solution 2:**
  - Jinja2 template engine

The Jinja2 Template Engine

- See **PennyFlaskJinja** app
  - runserver.py
  - penny.sql, penny.sqlite
  - database.py
  - ~~templates.py~~
  - **header.html, footer.html**
  - **index.html**
  - **searchform.html, searchresults.html**
  - **penny.py**
- Note the imperfection

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

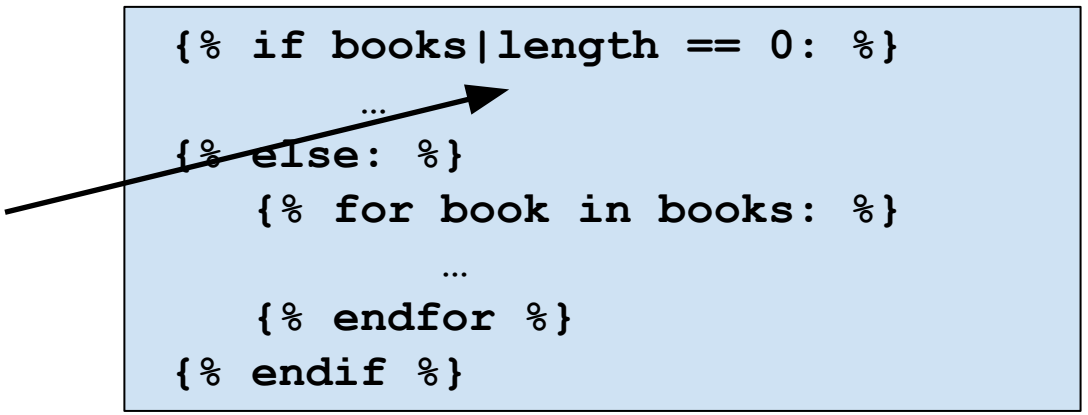
- Jinja2 ***render_template*** function
  - Accepts a template to be rendered
  - Accepts the values of Jinja2 variables used in that template
  - Returns a str object
  - Example:

```
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***



The Jinja2 Template Engine

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

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

The Jinja2 Template Engine

- Other Jinja2 features:
  - Automatic escaping (described later)
  - Functions
  - Conditional inclusion
  - Assignment
  - Error and exception handling
  - Template inheritance

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

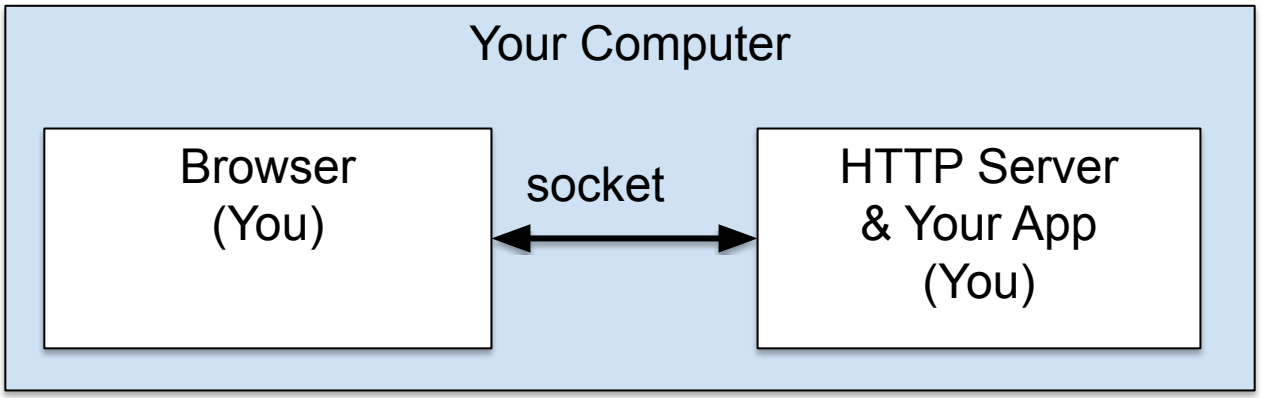
- Parting thoughts…
- Use of template engines facilitates modularity
  - Separates concerns
  - Hides design decisions

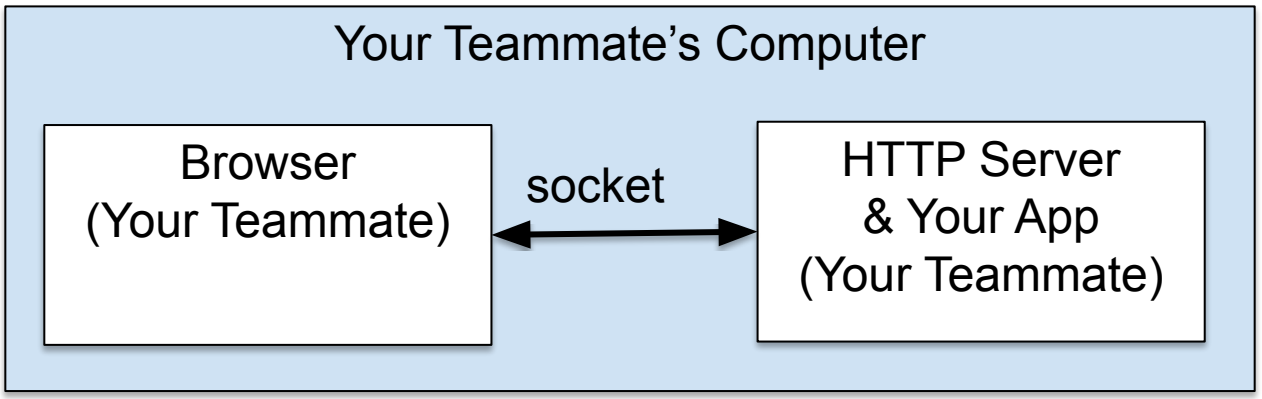
Agenda

- Web application frameworks
- The Flask web application framework
- Separation of concerns
- The Jinja2 template engine
- **HTTP in COS 333**

HTTP in COS 333

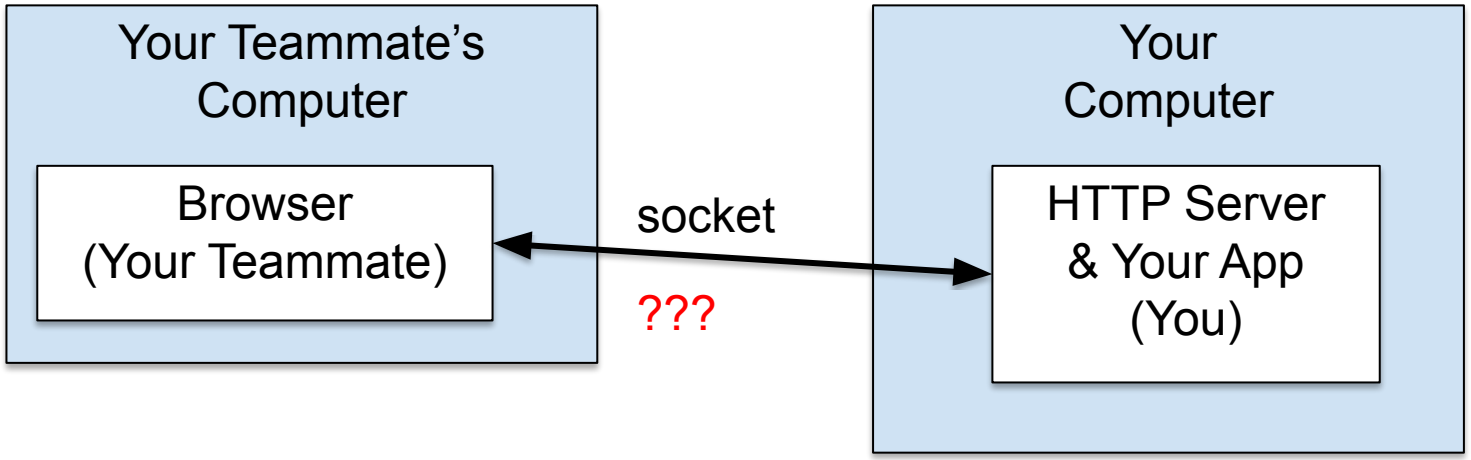
Option 1: Run HTTP server on local computer
Run browser on same local computer





HTTP in COS 333

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



Won't work unless both computers are on Eduroam

HTTP in COS 333

- Suggestion:
  - Use **option 1** during development
  - Use **option 2** to test network comm
    - If working alone:
      - Feel free to see an instructor, or…
      - Run the browser on your smart phone!

Lecture Summary

- In this lecture we covered:
  - Web application frameworks
  - The Flask web app framework
  - The Jinja2 template engine
  - HTTP in COS 333

Lecture Series Summary

- In this lecture series we covered:
  - Python WSGI programming
  - Web app frameworks
  - The Flask web app framework
  - The Jinja2 template engine
- See also:
  - **Appendix 1**: Python Decorators
  - **Optional lecture**: Django

More Information

- The COS 333 *Lectures* web page provides references to supplementary information

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

```
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()
```

One approach

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
```