Server-Side Web Programming: Python (Part 2)

Robert M. Dondero, Ph.D
Princeton University

Objectives

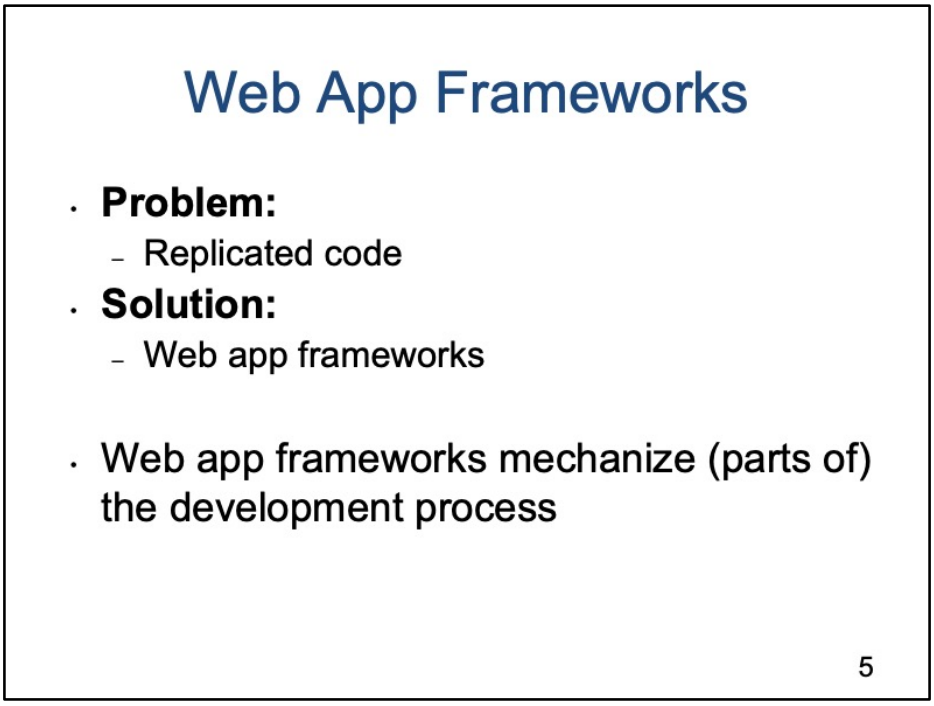
- We will cover:
  - Web app frameworks
  - The Flask web app framework
  - The Jinja2 template engine
  - The MVC architecture

Agenda

- **Web application frameworks**
- The Flask web application framework
- The Jinja2 template engine
- The MVC architecture

Web App Frameworks

Old approach to building a website:
- Write ad hoc server-side code
  - In Python, Java, PHP, etc.
  - Write ad hoc code to access DB
- Write ad hoc client-side code
  - In HTML, CSS, JavaScript, etc.



Web App Frameworks

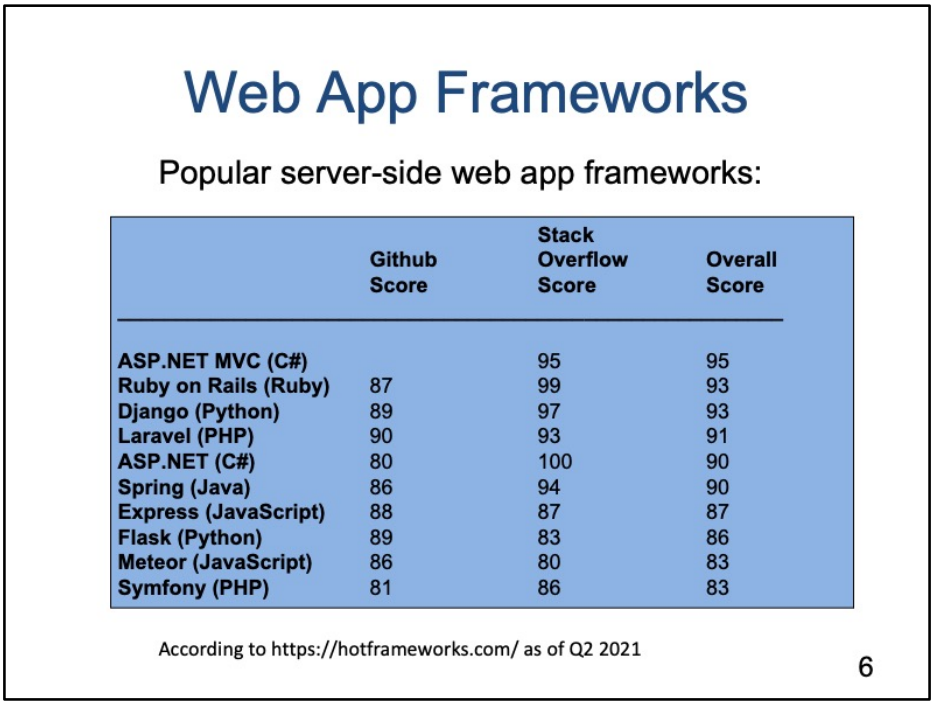
Problem:

Replicated code
Common patterns within and across web apps
Replicated code within and across web apps
Web app development involved lots of tedious/replicated effort

Solution:

Web app frameworks

Web app frameworks mechanize (parts of) the development process



	Github Score	Stack Overflow Score	Overall Score
ASP.NET MVC (C#)		95	95
Ruby on Rails (Ruby)	87	99	93
Django (Python)	89	97	93
Laravel (PHP)	90	93	91
ASP.NET (C#)	80	100	90
Spring (Java)	86	94	90
Express (JavaScript)	88	87	87
Flask (Python)	89	83	86
Meteor (JavaScript)	86	80	83
Symfony (PHP)	81	86	83

Web App Frameworks

[see slide]

Some observations:

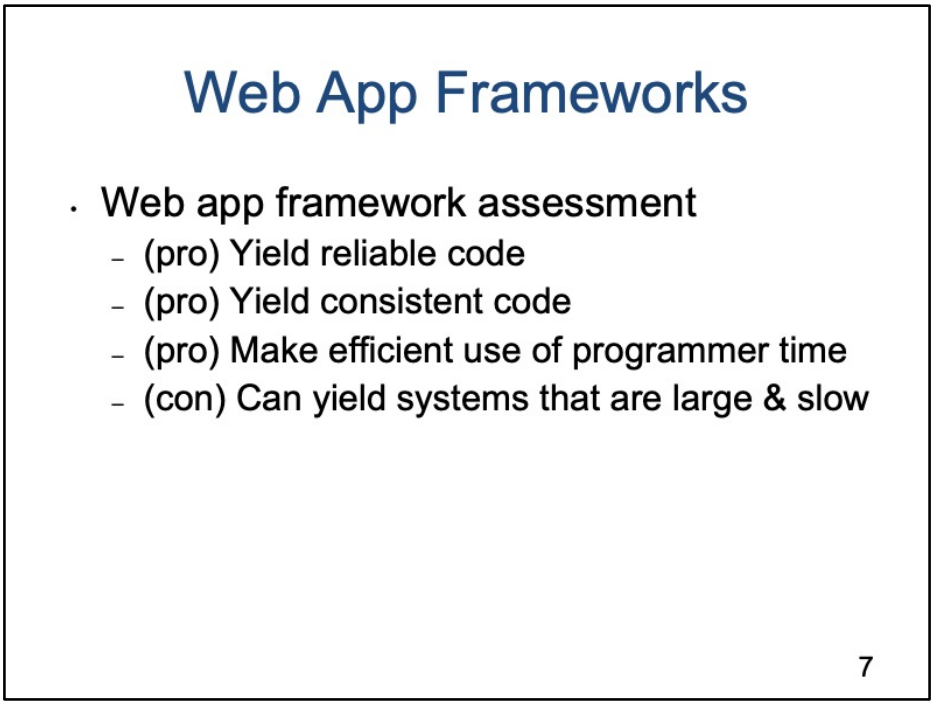
For Python: Django, Flask
For Java: Spring
For PHP: Laravel
For JavaScript: Express

Some observations:

Flask is #8

Some observations:

Stack overflow score could be misleading
Flask may be used just as often as Django, but may be much easier to use!



Web App Frameworks

Positives
- Yield reliable code
  - Framework code has been thoroughly tested
- Yield consistent code
  - Code has uniform structure across apps
- Make efficient use of programmer time
  - Framework handles repetitive parts
  - “DRY” (don’t repeat yourself) is encouraged

Negatives
- Can yield systems that are large & slow

Agenda

- Web application frameworks
- **The Flask web application framework**
- The Jinja2 template engine
- The MVC architecture

The Flask Web App Framework

- **Who**: Armin Ronacher
- **When**: 2004
- **Why**: A micro web framework written in Python

- Built on top of WSGI

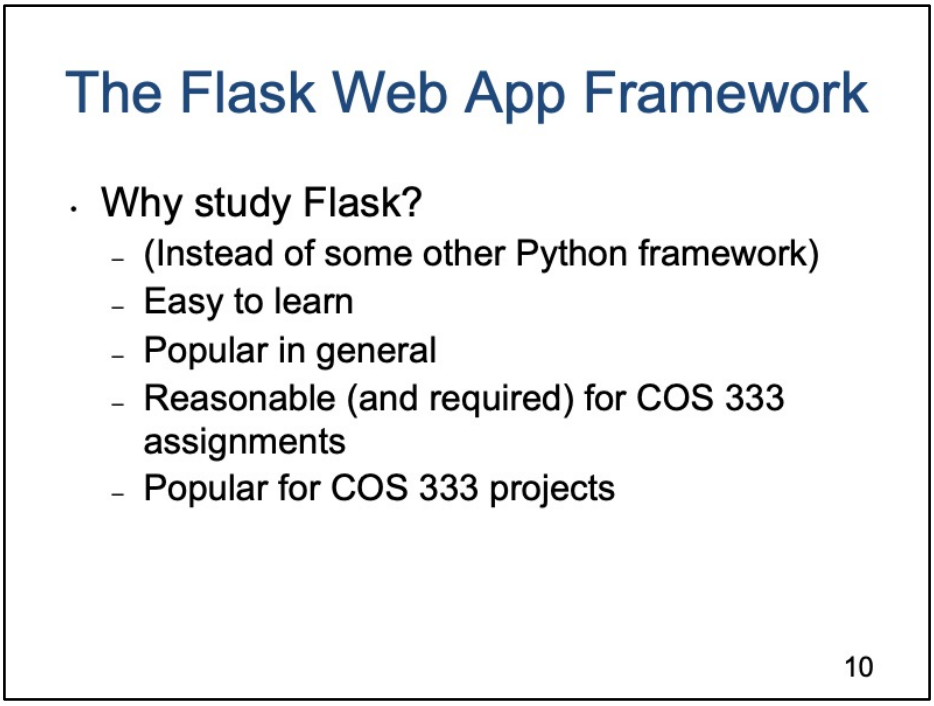
9

The Flask Web App Framework

Who: Armin Ronacher
When: 2004
Why (from Wikipedia):

> "**Flask** is a micro web framework written in Python.
> It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.
> However, Flask supports extensions that can add application features as if they were implemented in Flask itself."

Built on top of WSGI



The Flask Web App Framework

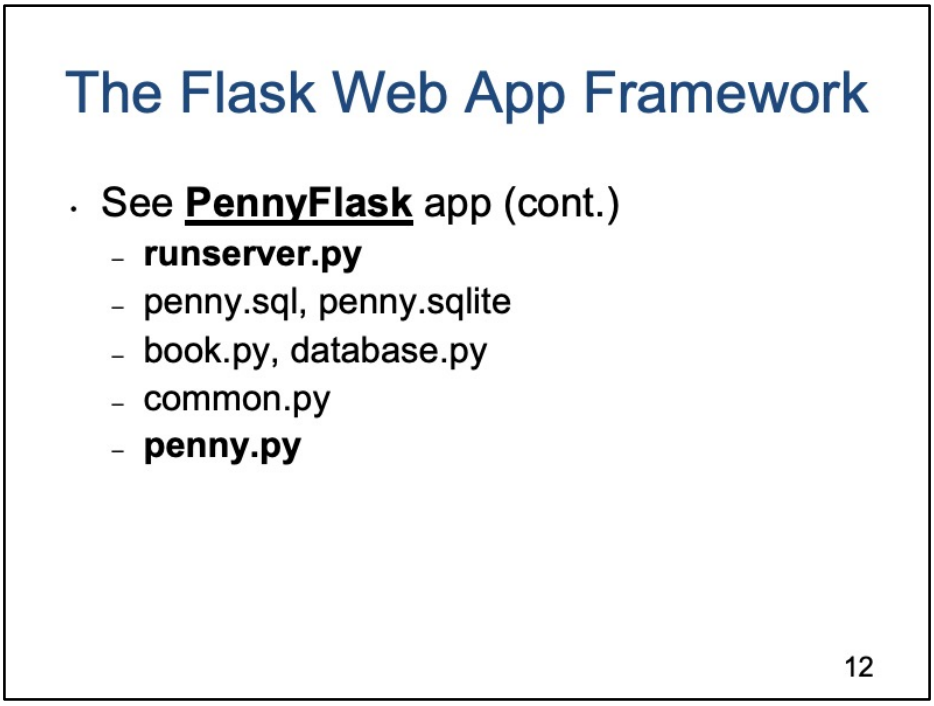
Why study Flask?
- (Instead of some other Python framework)
- Easy to learn
  - Simple (“micro-framework”)
  - Good documentation
- Popular in general
  - As we just noted
- Reasonable (and required for some of the COS 333 assignments)
- Popular for COS 333 projects

The Flask Web App Framework

- See **PennyFlask** app

```
$ python runserver.py 55555
 * Serving Flask app 'penny' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a
production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a
production deployment.
 * Running on http://192.168.1.8:55555/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 529-977-844
```

Runs test WSGI server that's bundled with Flask



The Flask Web App Framework

[see slide]

Code notes: runserver.py

Launches the Flask test server

Code notes: penny.py

Uses Python decorators to map HTTP requests for:

/ or /index to index()

/searchform to search_form()

/searchresults to search_results()

Uses request object to fetch name=value pairs and cookies

The Flask Web App Framework

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)
...
```

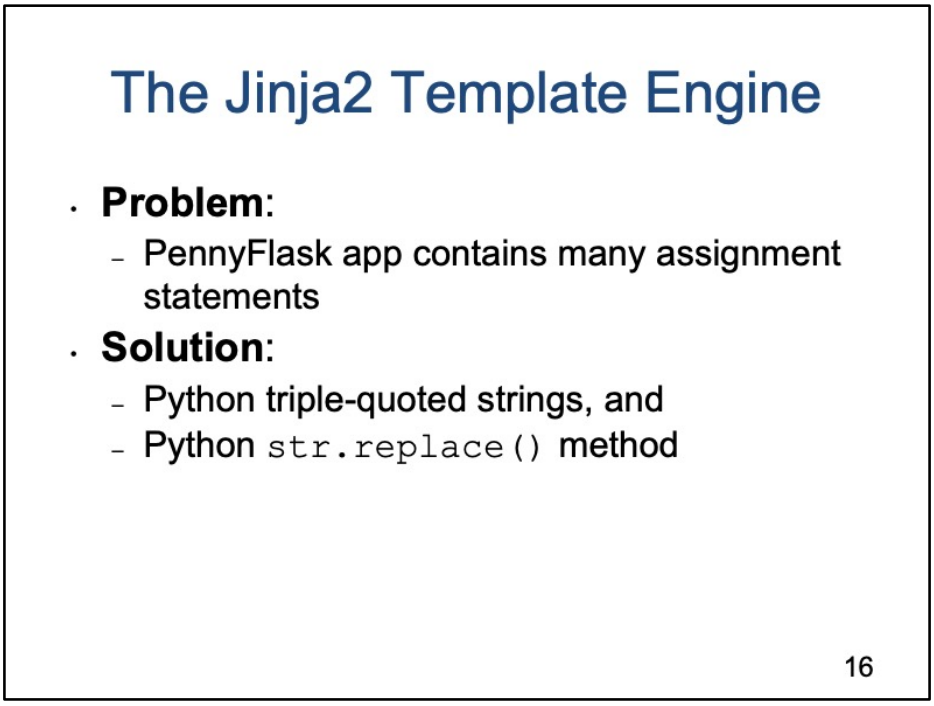
See **Appendix** for explanation

The Flask Web App Framework

- Flask provides utility functions
  - Functions to fetch “name=value” pairs
  - Functions to fetch/create cookies
  - Function to implement HTTP redirect

Agenda

- Web application frameworks
- The Flask web application framework
- **The Jinja2 template engine**
- The MVC architecture



The MVC Architecture

Problem:

PennyWsgi app contains many assignment statements (+= operator) to compose HTML code

Bulky; awkward; inefficient; error prone

Solution:

Python triple-quoted strings, and

Python `str.replace()` method

Aside: Triple-Quoted Strings

Triple-quoted string:

```
myStr = '''This is line one.
This is line two.
This is line three.'''
```

Allows a str literal to extend across multiple source code lines

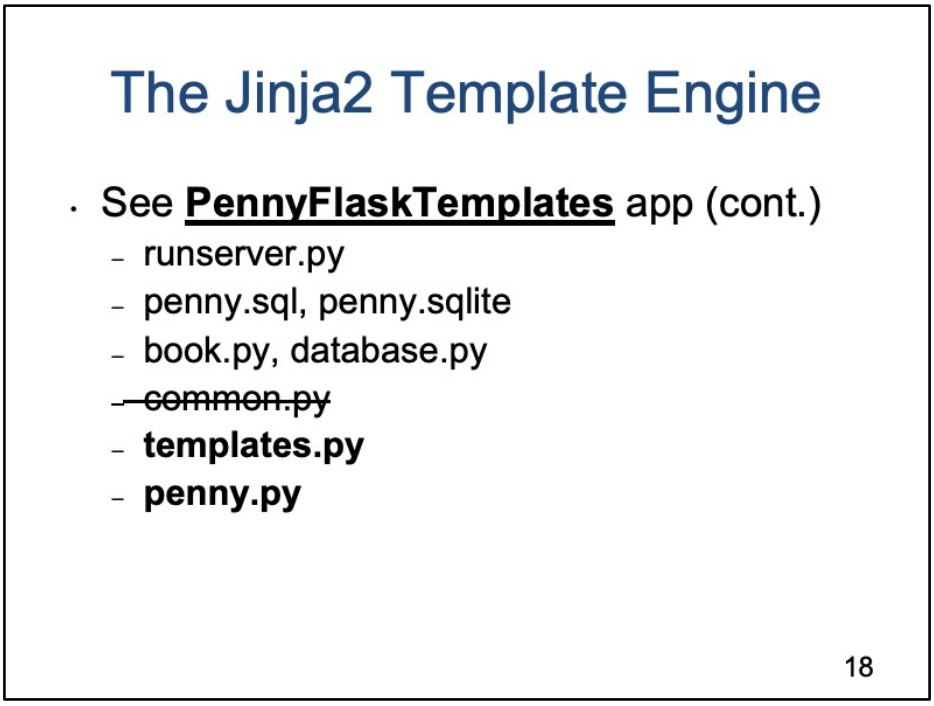
Equivalent:

```
myStr = 'This is line one.\n\
This is line two.\n\
This is line three.\n'
```

Could use double quotes instead of single quotes

Equivalent:

```
myStr = ('This is line one\n' +
'This is line two\n' +
'This is line three.\n')
```



The Flask Web App Framework

[see slide]

Code notes: runserver.py

Launches the Flask test server

Code notes: penny.py

Uses Python decorators to map HTTP requests for:

/ or /index to index()

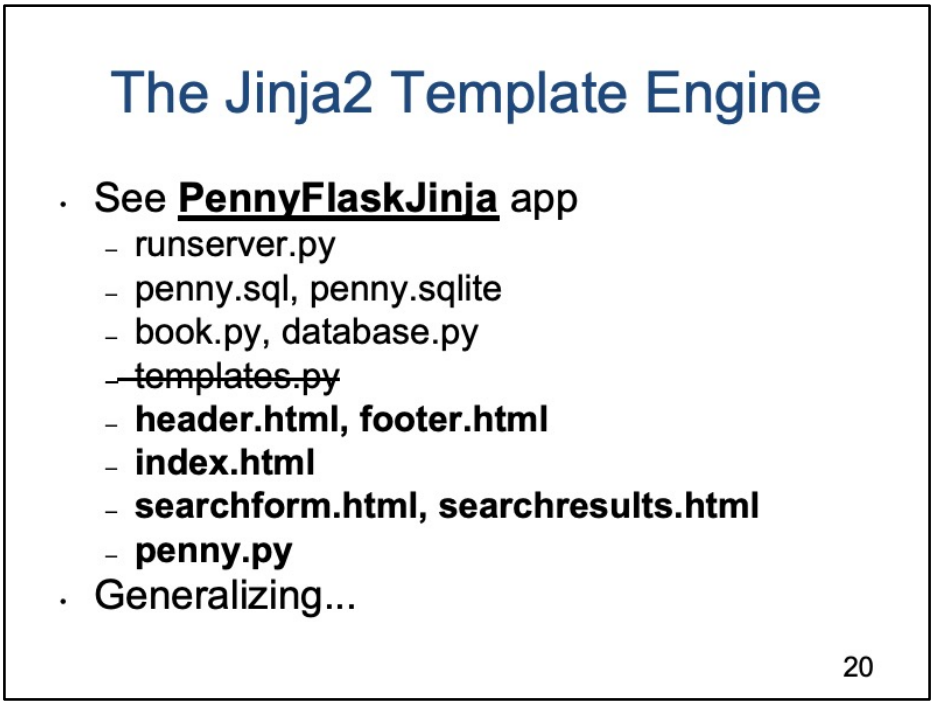
/searchform to search_form()

/searchresults to search_results()

Uses request object to fetch name=value pairs and cookies

The Jinja2 Template Engine

- **Problem**:
  - Still awkward
  - Common
- **Solution**:
  - ***Jinja2*** template engine
  - (Offers additional advantages too!)



The Jinja2 Template Engine

[see slide]

Code notes: header.html
A Jinja2 template, containing a placeholder whose key is ampm
Code notes: footer.html
A Jinja2 template
Code notes: index.html
A Jinja2 template; used to generate the index page
One template can include another
Code notes: searchform.html
A Jinja2 template; used to generate the searchform page
Code notes: searchresults.html
A Jinja2 template; used to generate the searchresults page
The text between double curly braces can be any pseudo-Python expression that evaluates to a str or can be turned into a str via an implicit call of a __str__() method
Can intermingle lines of pseudo-Python code
Syntax: {% lineOfPseudoPythonCode %}

Code notes: penny.py

Instantiates Jinja2 templates to generate HTML strings

The Jinja2 Template Engine

- ***Template*** (informally)
  - HTML doc with placeholders
  - Each placeholder is identified by a key

hello.html

```
Hello <strong>{{username}}</strong>
and welcome
```

The Jinja2 Template Engine

- To instantiate a template:
  - `render_template('somefile.html', key1=value1, key2=value2, …)`
    - For each placeholder identified by key in `somefile.html`, replaces the placeholder with the value associated with key
    - Returns the resulting string

The Jinja2 Template Engine

- To instantiate a template:
  - Example:

hello.html

```
Hello <strong>{{username}}</strong>
and welcome
```

```
html = render_template('hello.html',
    username='Bob')
```

equivalent to

```
html = 'Hello <strong>Bob</strong>
and welcome'
```

The Jinja2 Template Engine

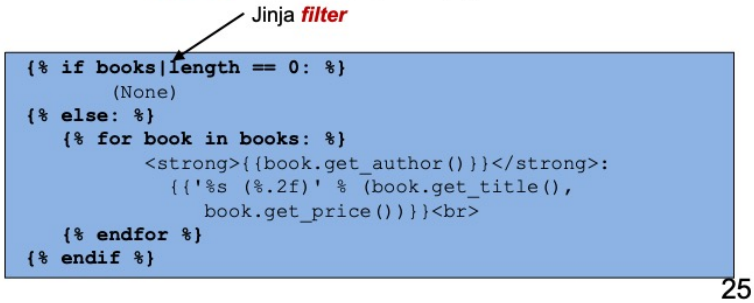
- Template can contain:
  - Inline Jinja2 expressions
    - …{{expr}}…
    - Similar to Python expressions

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

```
... {{book.get_author()}} ...
```

The Jinja2 Template Engine

- Embedded Jinja2 code
  - `{% embedded Jinja2 code %}`
  - Similar to Python code; some differences:
    - Indentation ignored; blocks closed with end statements
    - Use ***filters*** instead of __builtin__ functions




```
{% if books|length == 0: %}
       (None)
{% else: %}
   {% for book in books: %}
          <strong>{{book.get_author()}}</strong>:
            {{'%s (%.2f)' % (book.get_title(),
               book.get_price())}}<br>
   {% endfor %}
{% endif %}
```

The Jinja2 Template Engine

- Includes of other templates
  - `{% include 'other.html' %}`

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

Agenda

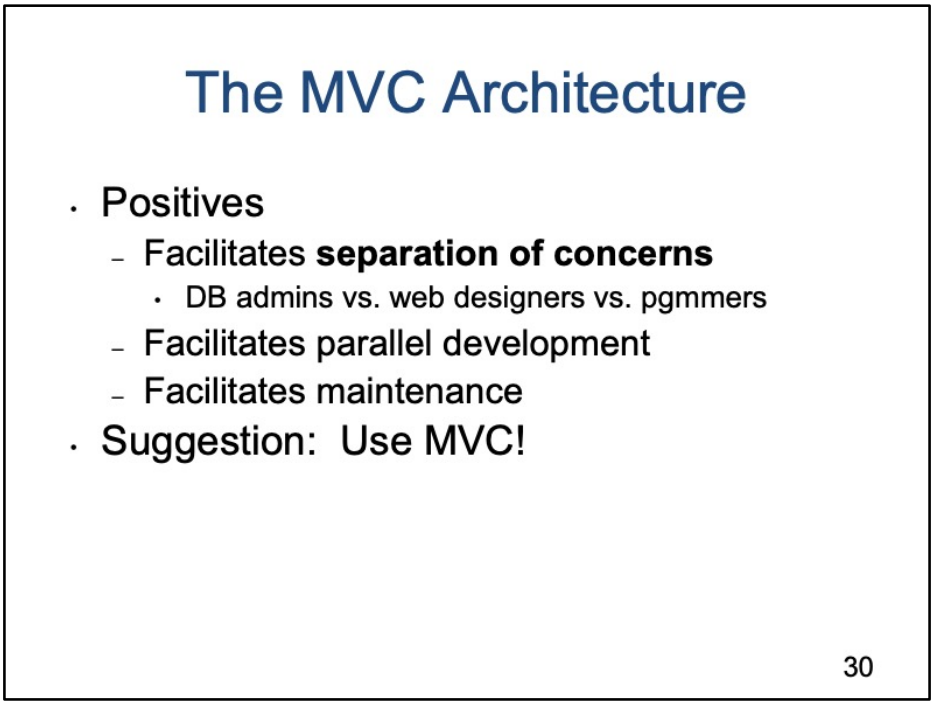
- Web application frameworks
- The Flask web application framework
- The Jinja2 template engine
- **The MVC architecture**

The MVC Architecture

- PennyFlaskJinja app uses..
- The ***Model-View-Controller (MVC)*** architecture
  - ***Model***: The system's **data access** code
  - ***View***: The system's **data presentation** code
  - ***Controller***: (vague) The system's **business code**; the logic that connects the model with its view(s)

The MVC Architecture

- In PennyFlaskJinja app:
  - **Model**: penny.sqlite, book.py, database.py
  - **View**: *.html files
  - **Controller**: penny.py



The MVC Architecture

Positives

Facilitates separation of concerns:

Model: DB administrators (SQL, normalization, ...)
Views: Web designers (art, aesthetics, HCI principles, HTML, CSS, ...)
Controller: Programmers (Python, Java, …)

Facilitates parallel development

(To some extent) can develop model, views, controller concurrently

Facilitates maintenance

MVC framework provides loose coupling
(To some extent) can change model, views, controller independently

The MVC Architecture

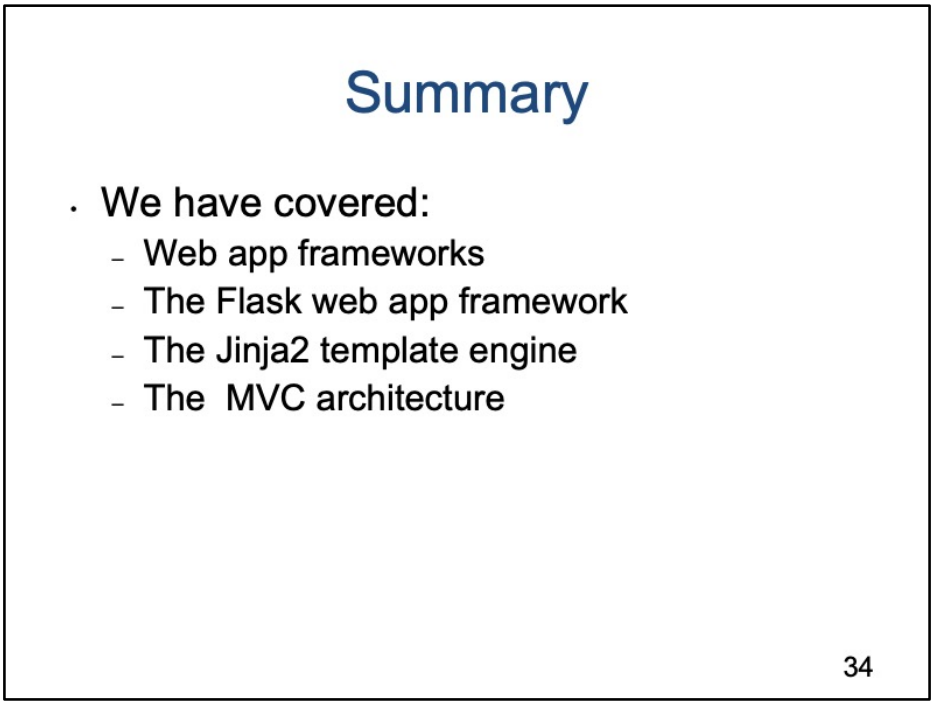
- Note:
  - PennyFlaskJinja implements MVC imperfectly…
  - searchresults.html (view) contains some Python-like code (controller)

The MVC Architecture

- Flask/Jinja2 suggestions:
  - Use MVC via Jinja2 templates!!!
  - But keep views simple
    - Avoid Python-like code in views/templates when you reasonably can

For More Information

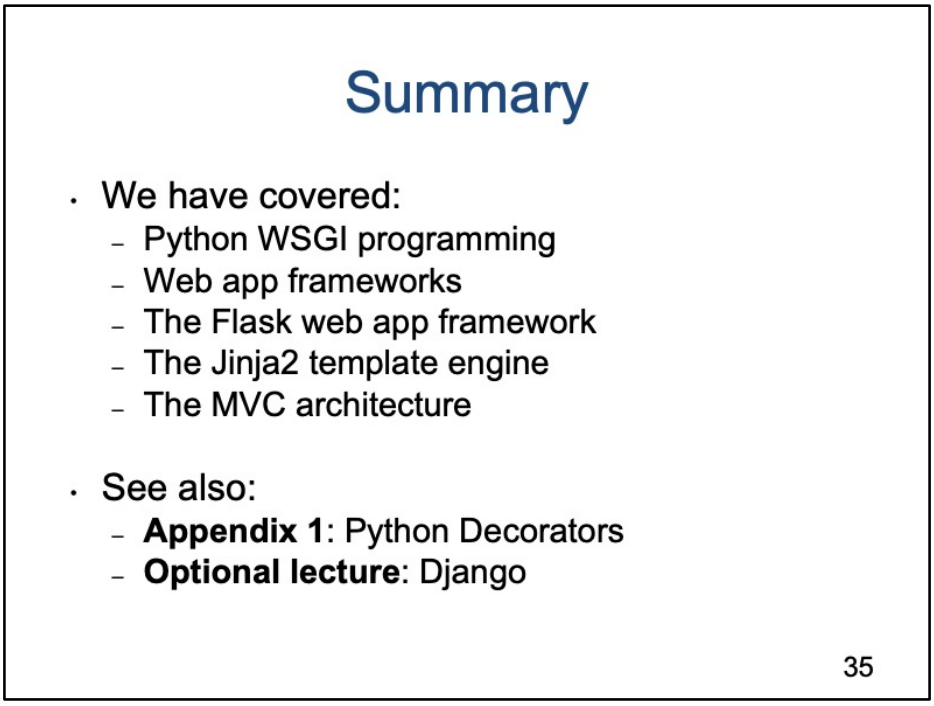
- There is much more to Flask and Jinja2
- Flask documentation:
  - https://flask.palletsprojects.com/en/2.0.x/api/
- Flask tutorial:
  - https://www.tutorialspoint.com/flask/
- Jinja2 documentation:
  - https://jinja.palletsprojects.com/en/3.0.x/



Summary

In this lecture…

[see slide]



Summary

In this sequence of lectures on Python WSGI Web Programming…

[see slide]

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

def sqr(i):
    return i * i
sqr = printNameDecorator(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 printNameDecorator(f):
    def fwrapper(i):
       print(f.__name__, 'was called')
       return f(i)
    return fwrapper

@printNameDecorator
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
```