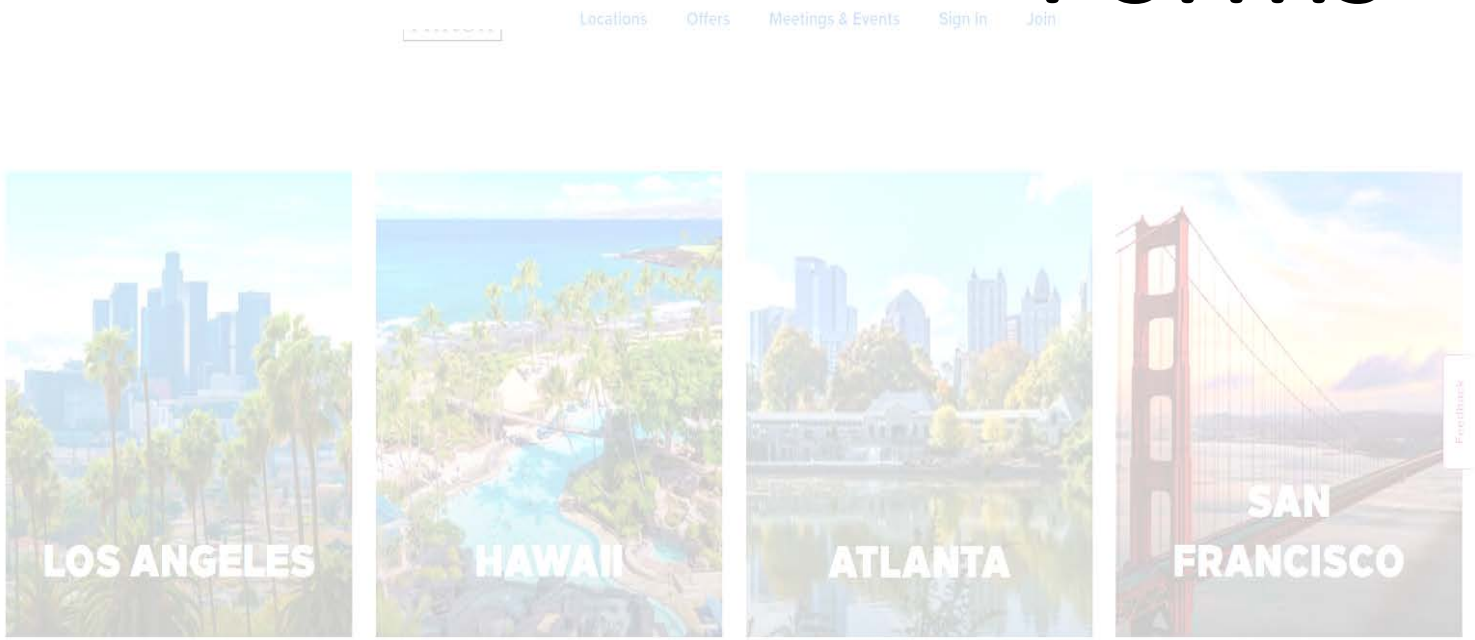
Dynamic content generation and Forms



Aims of this lecture

- Generating dynamic content in HTML
- Reusing HTML with templates
  - Template inheritance
- Introduction to Flask forms

Assessment 2 – Due Sept 15, 11:59 PM

- Create **static** HTML pages
  - Use Bootstrap

- **Four** pages
  - Landing or Main Page
  - Item details page
  - Item creation page
  - Manage items page

- Support navigation using tags.

Assessment 3 – Week 13 Workshop

- **Team work**
  - Team size (3-4)
  - Peer Review of your contributions
  - Workshop participation

- **Workshop**
  - Week 7 – Identify team members in the same workshop
  - Week 8 - Finalize team members
  - Week 13- Present/Demo your marketplace

- Monday Workshop Teams
  - Different time will provided in the week 13

marketplace

HTML CSS

Flask

PostgreSQL

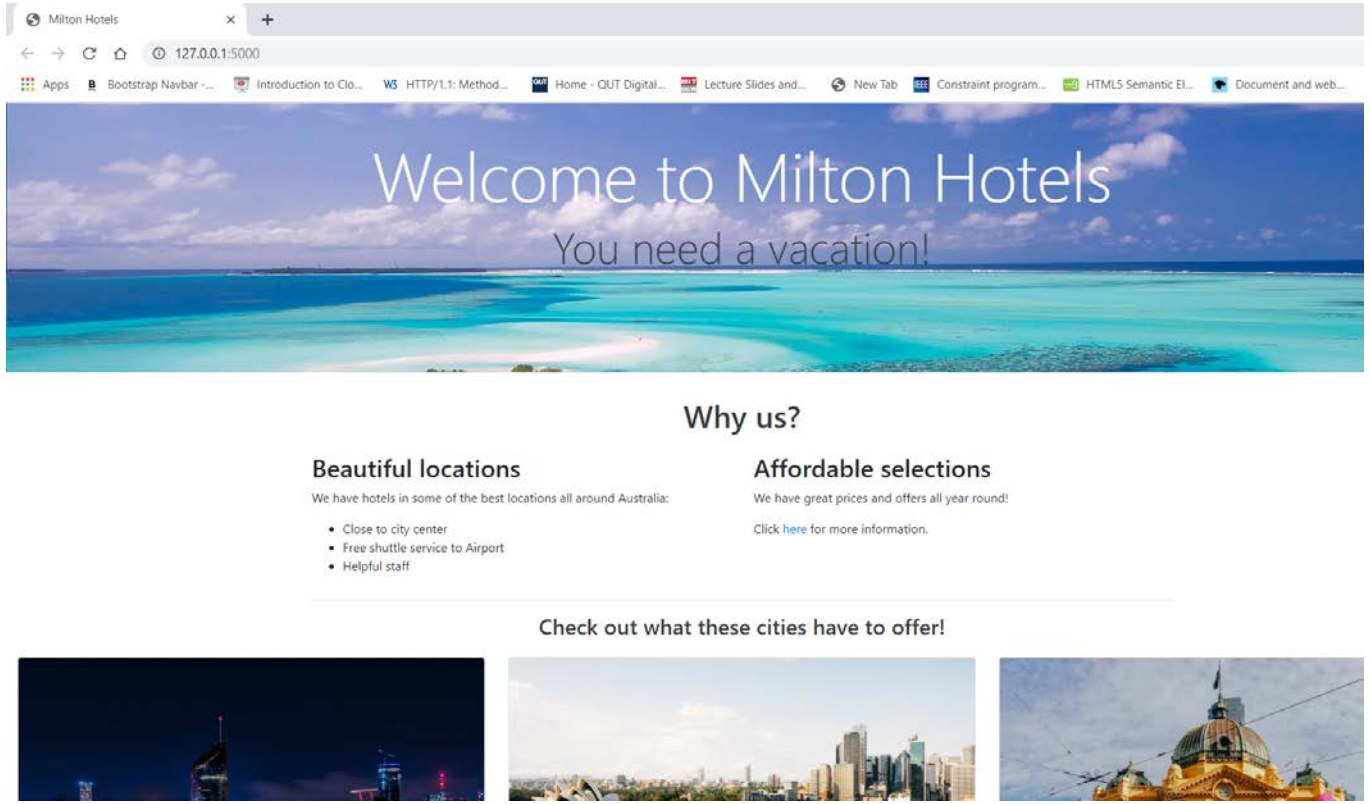
heroku

Topics covered Last Lecture

- Web application protocol
  - HTTP
  - Request types
- Frameworks and their functions
- The **Flask** framework
  - Routes
  - Session
  - Rendering template

Accessing a web application/page/resource

http://127.0.0.1:5000



http – Hypertext Transfer Protocol

Quiz

Flask application

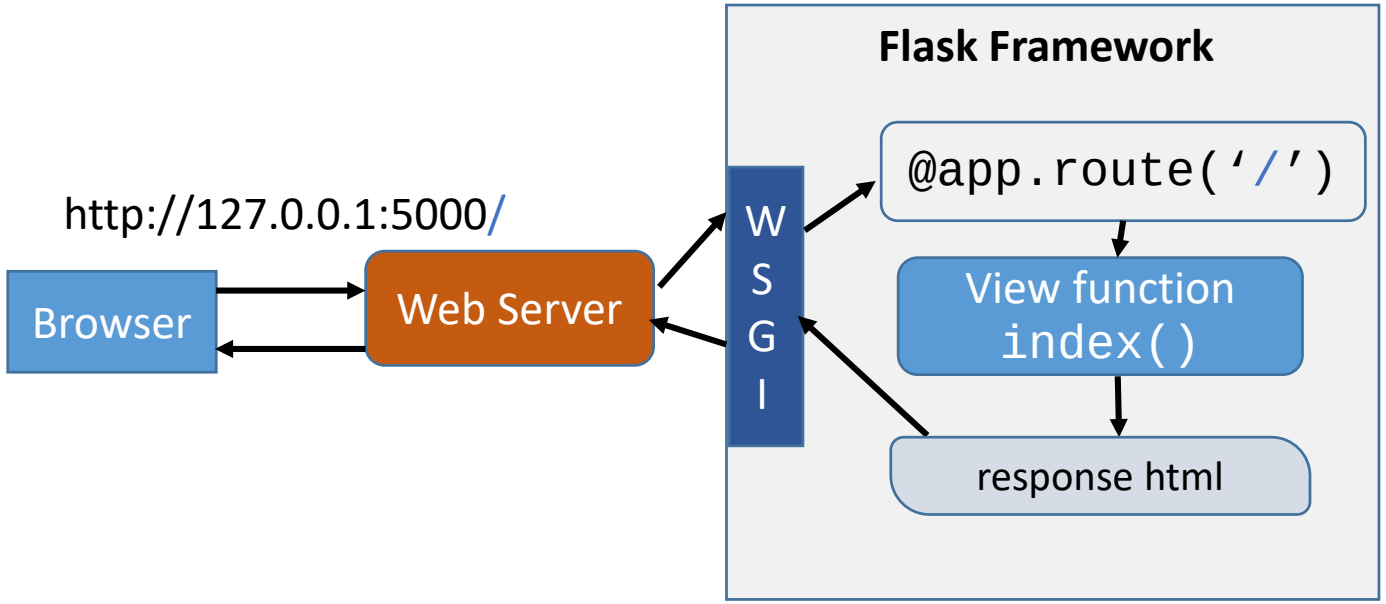
- Flask web application code and demo (5 lines of code)

```
from flask import Flask, request

#create a flask web application
app = Flask(__name__)
app.debug=True

#@app.route('/')
def index():
        return '<h1>Hello</h1>'
```

Flask App and View functions



Flask comes with a small development web server

Passing path parameters

```
@app.route('/<name>')
def index_name(name): #view function with path parameter
        print(name)
        return '<h1>Hello {} </h1>'.format(name)
```

By Default, the parameter is string.

@app.route(‘/<int:id>’)
@app.route(‘/<float:id>’)
@app.route(‘/<path:val>’)

path – is a special string type that can have forward slash in it (used rarely)

Sessions in Flask

- What is a session

  Session is created by the server

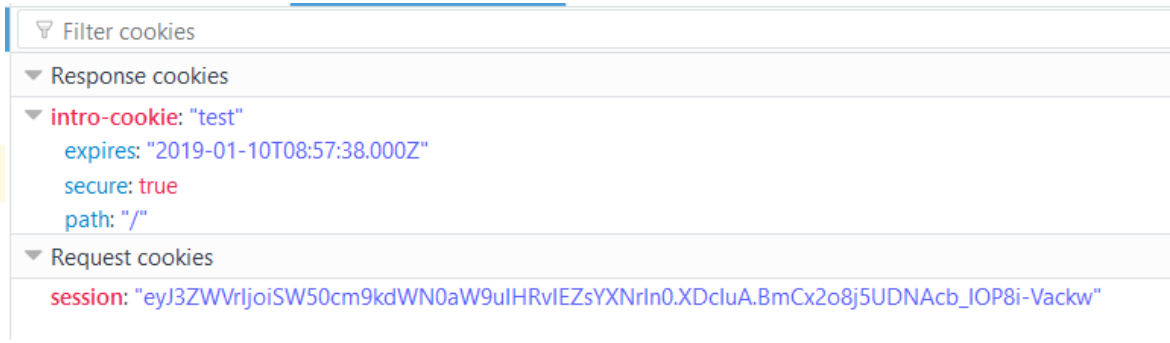
  Available only till the browser is open

  Encoded (not encrypted) – It cannot be modified by the client as it needs the secret key

```
@app.route('/setsession/<name>')
def set_session(name): #view function
    session['week']='Introduction to Flask'
    return '<h3>Set the session week</h3>'


@app.route('/getsession')
def get_session(): #view function
    value = session['week']
    return '<h3>Value is {}</h3>'.format(value)
```

Session viewed in a browser



Starter code on Blackboard

- Code walkthrough of
  - main.py
  - views.py
  - __init__.py

- Directory structure

- Blueprints

week6_lecture_variables.zip

Rendering the HTML

- How do we use the HTML file created in Week2 and Week3

- Jinja Template engine can be used

- How does it work?
  - Templates are HTML files with extra syntax to handle data dynamically
  - Syntax for place holders in the template allow the dynamic data to be added.

  - **Rendering**: Template engine replaces the place holders with actual values and generates the final response

Example

```
from flask import Flask, render_template

app = Flask(__name__)  # This is my flask ap
app.debug=True



@app.route('/')
def index(): #view function
    return render_template('index.html')
```

- render_template is the function that is used to return html
- Flask looks for templates in the ‘templates’ directory

Variable substitution with Templates

- Jinja template allows specifying variables

```
<!DOCTYPE html>
<html>
<body>

<p>{{ variable }}</p>
```

- render_template uses the variables refered to in the HTML
  - Take in variable number of named arguments

```
@mybp.route('/trial')
def trial():
        return render_template('trial.html', variable='sending data from server')
```

Variable substitution with Templates

- Pass another variable to the HTML template

Add a session variable and use it in HTML

- Session object is passed to the template directly

- Can access the session in the template HTML

```
<!DOCTYPE html>
<html>
<body>

<p>{{ variable }}</p>
<p>The session object is accessed: {{session['user']}}</p>
<p>This is a paragraph.</p>

</body>
</html>
```

Control Structures (for, if)

If statement

```
<h1 class="display-2">Welcome to {{hotel_name|title}} </h1>

    {% if session['user'] %}
    <h3 class="display-4">Hello {{session['user']}} </h3>
    {% else %}
    <h3 class="display-4">Hello we offer great prices</h3>
    {% endif %}
```

for statement

```
{% for hotel in hotels %}
<div class="card">
    <img class="card-img-top" src={{'../static/img/' + hotel.image }}
    <div class="card-body">
        <h4 class="card-title">{{hotel.name}}</h4>
        <p class="card-text">{{hotel.description}}</p>
    </div>
    <div class="card-footer"><a href="#" class="btn btn-primary">Detai
</div>
{% endfor %}
```

Control Structures (if, for)

- Useful for looping through data structures

```
{% for i in range(loop) %}

<h3>{{variable}} {{i}}</h3>

{% endfor %}
```

Creating a heading3 based on the loop and the variable

```python
@mybp.route('/trial_for')
def trial_for():
    return render_template('trial.html', variable='printing values', loop=3)
```

Control Structures (if, for)

- Adding an if statement in the template

url_for function

- During development, routes can change.
  - Routes can be referred to in multiple HTML files
  - Maintaining and changing the routes can be difficult

- **url_for**
  - Returns the URL for a view function

```
url_for('view function name')
url_for('<blueprint>.view function name')
```

- Use url_for to:
  - Redirect to a page
  - Display it (HTML href)

redirect function

- Function that redirects to another page

- Server needs to redirect the request (e.g. redirect to login page)

```
@mybp.route('/myredirect')
def redirection():
        return redirect('/trial_for')
```

```
@mybp.route('/myredirect')
def redirection():
        return redirect(url_for('main.trial_for'))
```

Dynamic HTML

- Code walkthrough of
  - models.py
  - view.py
  - index_list.html
- Passing parameters

week6_lecture_templates.zip

Reuse Templates (Template inheritance)

- Create a base html containing header, footer or any content that needs to be used across the application

- Add place holder for the content that needs to change

- Extend pages from base html

Starter code on Blackboard

- Code walkthrough of
  - base.html
  - index_reuse.html

- Passing parameters

Error pages – Create your own error pages

- Error handling – no routes available, or server error
  - Response code – 400, 500

```
@app.errorhandler(404)  #error handlers need to return template and status code
def not_found(e):
    return render_template('404.html'),404
```

```
{% extends "base.html" %}

{% block body %}

<div class="jumbotron myjumbotron">
    <div class="container-fluid">
        <h1 class="display-4">Sorry, cannot find the page</h1>
    </div>
</div>


{% endblock %}
```

Flask WTForms

- Install Flask forms
  - pip install flask-wtf
  - pip install flask-bootstrap

- Flask WTForms (Wrapper over WTForms)
  - FlaskForm – Is the main container
  - Field – Renders and captures data
  - Validator – Takes input and validates a criteria – length of a string, format of a string.
  - Widget – Render the field in a suitable manner

https://wtforms.readthedocs.io/en/stable/

Starter code on Blackboard

- Code walkthrough of
  - forms.py

week6_lecture_forms.zip

Form Creation

```
from flask_wtf import FlaskForm
from wtforms import StringField

class ContactForm(FlaskForm):
    username = StringField('Name')
    email = StringField('Email Address')
```

FlaskForm containing two Fields

Fields

-

Attribute/method	Description
StringField	A text field
TextAreaField	A multi-line text field
DateField	A text field that accepts a datetime.date value
DateTimeField	A text field that accepts a datetime.datetime value
FileField	A file upload field
HiddenField	A hidden field (used by forms to pass the hidden token)
PasswordField	Password text field to hide characters entered
RadioField	Radio button field
SubmitField	Form submission button

- Other HTML fields such as SelectField, SelectMultipleField,FloatField

Form Display in HTML

```
@mainbp.route('/')
def index():
        form=ContactForm()
        return render_template('index.html',form=form, hotels=hotels)
```

Create a Form object

Pass the Form to render it in HTML

Form Display in HTML

The form is posted to the URL

```
<h2>Contact Form</h2>
<!--form in the HTML replace with Flask Form-->
<form method="post" action={{url_for('main.create_contact')}}>
<!-- keeps the label and text in a group vertically stacked-->

<div class="form-group">
{{form.user_name.label}}
{{form.user_name (class="form-control")}}
</div>
```

Use the Form object in the HTML

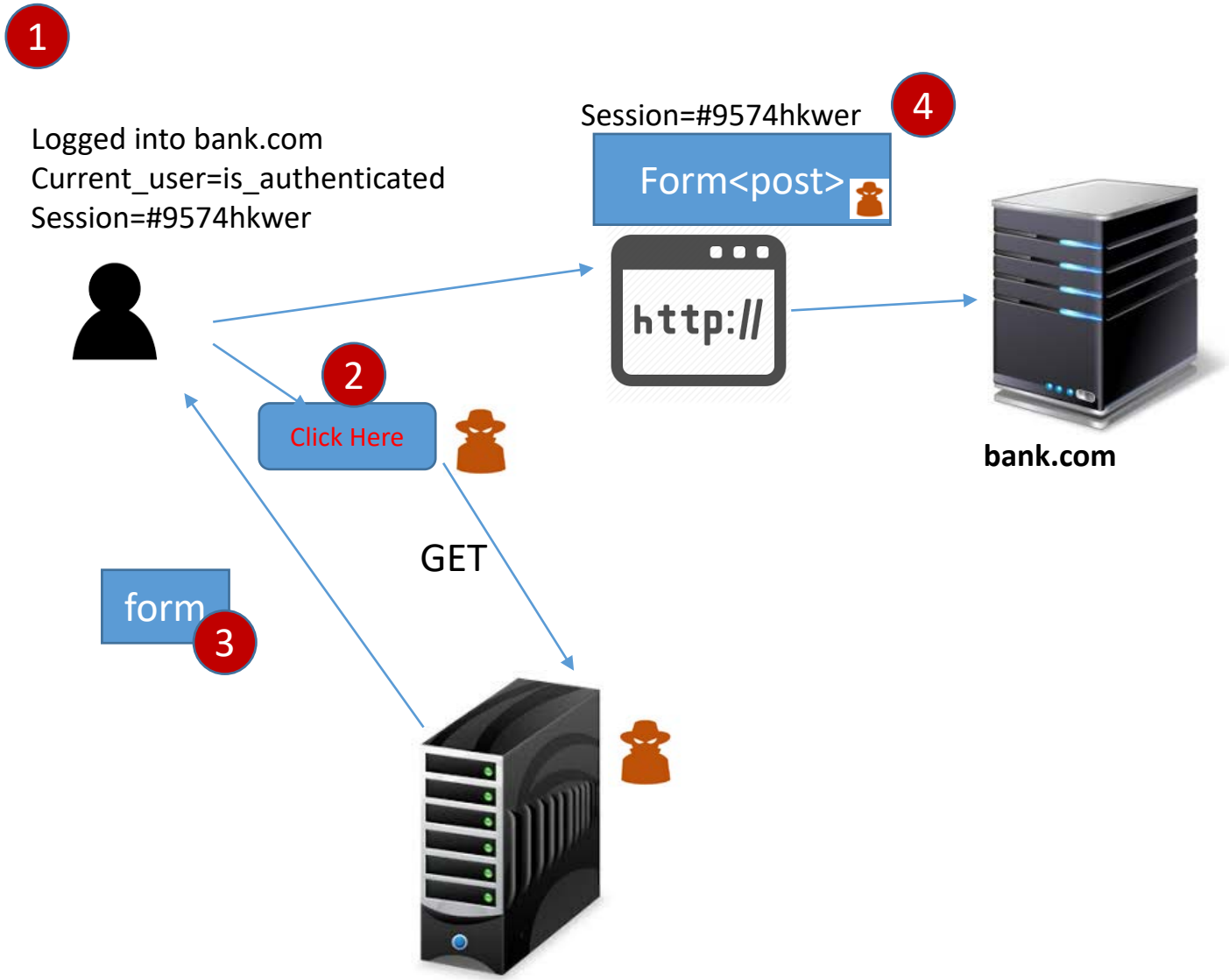
Accessing User Inputs

Support multiple request types

```
@mainbp.route('/contact', methods=['GET','POST'])
def create_contact():
        form = ContactForm()
        if form.validate_on_submit():
                print("Form has been submitted successfully")
                print(request.form['user_name'])
        return redirect(url_for('main.index'))
```

1. Create the form object – Template engine creates the form
2. Validate form based on HTML input
3. Access form data from the request

Cross Site Request Forgery (CSRF)



- User is logged into a web application (bank.com)
- The user opens a new tab and clicks on a link
- The site can silently send a form
- Fires http request with user session to bank.com
- As long as the session is active, the request can be forged
- CSRF typically occurs using a POST request

Flask CSRF Protection

- Sessions are encoded with the secret key
  - Read the session but cannot modify it

- Form protection
  - Generates security token for all forms
  - The token is put in a hidden field on the form named *and rendered by the* template
  - The token should be passed back to the view function
  - Form validation fails without a csrf_token

```
{{form.csrf_token}}
```

Using Bootstrap to render form

```
{{ wtf.quick_form(form) }}
```

Form macro reference

quick_form(*form, action=".", method="post", extra_classes=None, role="form", form_type="basic", horizontal_columns=('lg', 2, 10), enctype=None, button_map={}, id="")*

- By default the action is to post the form to the same route
- Can be changed to a different route using action attribute

Source: https://pythonhosted.org/Flask-Bootstrap/forms.html

Initialize Bootstrap

```
from flask import Flask
from flask_bootstrap import Bootstrap


def create_app():
        app=Flask(__name__)
        app.debug=True
        app.secret_key='thisisasecretkey122'

        boostrap = Bootstrap(app)

        from .views import mainbp
        app.register_blueprint(mainbp)


        return app
```

Creating form, template and view function

- Template to render the form
- Bootstrap support will be used (requires pip install Flask-Bootstrap)

```
{% include "base.html" %}
<!-- need to include this for bootstrap to render-->
{% import "bootstrap/wtf.html" as wtf %}


{% block body %}

<div class="page-header">
    <h4>{{heading}}</h4>
</div>


<!-- Form rendering is simple and easy if using bootstrap -->
<div class="col-md-6">
{{wtf.quick_form(form)}}
</div>
```

Connecting the dots

- Multiple functions/statements required for a solution
  - HTML update
  - Flask view function update
  - Connect them together

- Code walkthrough after the lectures
  - Understand how different code blocks come together

Summary

- Generating dynamic content in HTML
- Reusing HTML with templates
  - Template inheritance
- Introduction to Flask forms

Thank you!