

11. Web Frameworks and Forms

Today's Objectives

- Define web framework
- Build a simple Flask webpage
- Create a simple template
- Create a website form
- Validate user input with clear error messages

SEO Tech
Developer

Web Framework

Web Framework

- As we learned, a **framework** is structured code that makes it easier for a programmer or developer to create a desktop/mobile/web application; it usually includes a set of libraries to perform various tasks

LIBRARY VERSUS FRAMEWORK	
Library	Framework
Library is a set of reusable functions used by computer programs.	Framework is a piece of code that dictates the architecture of your project and aids in programs.
You are in full control when you call a method from a library and the control is then returned.	The code never calls into a framework, instead the framework calls you.
It's incorporated seamlessly into existing projects to add functionality that you can access using an API.	It cannot be seamlessly incorporated into an existing project. Instead it can be used when a new project is started.
They are important in program linking and binding process.	They provide a standard way to build and deploy applications
Example: jQuery is a JavaScript library that simplifies DOM manipulation.	Example: AngularJS is a JavaScript-based framework for dynamic web applications.

Web Framework

- As we learned, a **framework** is structured code that makes it easier for a programmer or developer to create a desktop/mobile/web application; it usually includes a set of libraries to perform various tasks
- A **web framework** is a framework for web development
 - Examples: Django, Flask, Bottle, Angular, etc.
- We will be using Flask !
 - It's to learn and highly flexible

Flask

- Is a web framework that uses Python
- Helps with routing and fetching, and HTML/CSS for layout
- Almost makes development too easy...
- “Hello, world” web app is about 10 lines of Python code, and nothing else!!
- Advertises itself as a “micro-framework”
- The micro- prefix here just means it abstracts over very little (meaning you can decide what to abstract)

Hello World in Flask (in .py file)

```
from flask import Flask

# gets name of the .py file so Flask knows it's name
app = Flask(__name__)

# tells you the URL the method below is related to
@app.route("/")
def hello_world():
    # prints HTML to the webpage
    return "<p>Hello, World!</p>"
```

Running Flask in Codio (option 1)

- Add the following to the end of your python file:
 - `if __name__ == '__main__':`
 `app.run(host="0.0.0.0")`
- Start the Flask server using normal python command:
 - `python3 file_name.py`
 - Stop server using CTRL + C
- Click "Flask Application" in the top menu bar to see the webpage
 - If you see 404, check that your server is running

Running Flask in Codio (option 2)

- Set an environment variable:
 - `export FLASK_APP=file_name_without_extension`
 - e.g. if your file is `hello.py`, run `export FLASK_APP=hello`
- Start the Flask server by running:
 - `flask run --host=0.0.0.0`
 - Stop server using CTRL + C
- Click "Flask Application" in the top menu bar to see the webpage
 - If you see 404, check that your server is running

Making Changes

- If you make a change with the server running and re-load the webpage, you'll notice the change isn't propagated. Either:
 - Re-start the server - stop it using CTRL+C and re-start it
 - Run your server in DEBUG mode
- To run your server in DEBUG, change the last line of code to:
 - `app.run(debug=True, host="0.0.0.0")`

Adding a page to your website

- Set the URL – for example website.com/about is normal:
 - `@app.route("/about")`
- Make function that does something on that page:
 - `def about():
 return "<p>About us!</p>"`
- View your new page by:
 - Starting your Flask server
 - Open your website
 - Add `/about` to the URL

DEMO Using Flask

SEO Tech
Developer

Templates

You can have HTML rendered this way...

```
import flask
import random
import os

app = flask.Flask(__name__)

@app.route('/')
def index():
    return '<html itemscope="" itemtype="http://schema.org/WebPage" lang="en"><head><meta content="Search the world's information, including webpages, images, videos and more. Google has many special features to help you find exactly what you're looking for." name="description"><meta content="noodp" name="robots"><meta content="/logos/doodles/2017/bessie-colemans-125th-birthday-5751652702224384-hp.gif" itemprop="image"><link href="/images/branding/product/ico/googleg_lodp.ico" rel="shortcut icon"><meta content="Bessie Coleman's 125th birthday! #GoogleDoodle" property="og:description">...'

app.run(
    port=int(os.getenv('PORT', 8080)),
    host=os.getenv('IP', '0.0.0.0')
)
```

Templates

- Templates enable you to put your HTML into nicely formatted files
 - No need to stuff all HTML into a crazy string
- Flask has a built-in templating language called Jinja2
 - This will be relevant when you Google for documentation!

HTML template files

- Instead of having to squeeze a bunch of HTML into your python file, you can use `templates` which are HTML files
- To use this, you **must** create a `templates` folder/directory (this is the folder that Flask looks for) to hold your HTML files.
- Let's say we create 3 files:
 - `layout.html`
 - `home.html`
 - `about.html`

Jinja

- Jinja is what powers templates in flask -- allowing us to do things like leave placeholders called [Blocks](#) to be filled later.
- Jinja also helps inject python into HTML pages. Some basic and useful things:
 - [Variables](#)
 - [if statements](#)
 - [for loops](#)

Creating your layout.html

- A layout page holds all the code you need for every page, so you don't need to copy-paste for every new page you make!

```
<!-- layout.html -->
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <h1>This shows up on every page</h1>
    {% block content %}{% endblock %}
  </body>
</html>
```



Jinja block

home.html and about.html

```
{% extends "layout.html" %}
{% block content %}
    <h2>{{ subtitle }}</h2>
{% endblock content %}
```

Notes about the above code:

1. `{% extends "layout.html" %}` - says this inherits from our layout.html file
2. `{% block content %}...{% endblock content %}` - specifies the part of the layout.html template we are filling in
3. `<h2>{{ subtitle }}</h2>` - says that we are expecting something called subtitle we want to print

Rendering templates

Update your python file in the following ways:

1. Import `render_template`
2. Call `render_template` instead of returning raw HTML

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route("/")
@app.route("/home")
def home():
    return render_template('home.html', subtitle='Home Page')
```

DEMO Using Templates

SEO Tech
Developer

Forms

WTForms

- WTForms is a python library for forms validation and rendering
- **Forms** are made up of **Fields** with **Validators**
 - Fields include basic form field types
 - **StringField** – field to input text
 - **PasswordField** – like StringField but value is not rendered back to browser
 - **SubmitField** - allows checking if a given submit button has been pressed
 - Validators check user input
 - **DataRequired** - sets the required [flag](#) on fields it is used on
 - **Length** - Validates the length of a string
 - **Email** - Validates an email address using email_validator package
 - **EqualTo** – Compares the values of two fields - used to facilitate the password change form
- Flask-WTF is a library that integrated WTForms with Flask

Create Forms.py

```
# forms.py
```

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField
from wtforms.validators import DataRequired, Length, Email, EqualTo

class RegistrationForm(FlaskForm):
    username = StringField('Username',
                           validators=[DataRequired(), Length(min=2, max=20)])
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    confirm_password = PasswordField('Confirm Password',
                                     validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Sign Up')
```


Add Form to Website (hello.py)

- Make sure your Flask app can see your form by importing it in `hello.py`:

```
from forms import RegistrationForm
```

- Add a new page to your site:

```
@app.route("/register", methods=['GET', 'POST'])
def register():
    form = RegistrationForm()
    return render_template('register.html', title='Register', form=form)
```

Simple register.html

```
{% extends "layout.html" %}
{% block content %}
    <div class="content-section">
        <form method="POST" action=""> <!--sends information to form -->
            {{ form.hidden_tag() }} <!-- passes your secret key ** -->
            <fieldset class="form-group">
                <legend class="border-bottom mb-4">Join Today</legend>
                <div class="form-group"> <!--add username field -->
                    {{ form.username.label(class="form-control-label") }}
                    {{ form.username(class="form-control form-control-lg") }}
                </div>
                <div class="form-group"> <!--add email field -->
                    {{ form.email.label(class="form-control-label") }}
                    {{ form.email(class="form-control form-control-lg") }}
                </div>
                <div class="form-group"> <!--add password field -->
                    {{ form.password.label(class="form-control-label") }}
                    {{ form.password(class="form-control form-control-lg") }}
                </div>
                <div class="form-group"> <!--add confirm password field -->
                    {{ form.confirm_password.label(class="form-control-label") }}
                    {{ form.confirm_password(class="form-control form-control-lg") }}
                </div>
            </fieldset>
            <div class="form-group">{{ form.submit(class="btn btn-outline-info") }} </div> <!--add button field -->
        </form></div>{% endblock content %}
```

Secrets Keys??

To protect our website form from bad cookies, we need to set a secret key.

1. Pop open a python interpreter and generate a 16 byte token -- copy it without the quotes

```
>>> import secrets  
>>> secrets.token_hex(16)
```

2. In your main `.py` file, after `app = Flask(__name__)` add...
`app.config['SECRET_KEY'] = 'the key you generated'`

Passing Success Message

- After you instantiate form (`form = RegistrationForm()`) in your main .py file:

```
if form.validate_on_submit():
    flash(f'Account created for {form.username.data}!', 'success')
    return redirect(url_for('home'))
```

- The f-string allows us to insert a variable -- in this case the data filled in by the user in the username field.
- The `success` part tells bootstrap the type of message we are sending so it style accordingly.
- Finally, because it was validated and the account was "created" we can send the user to the home screen.
- To get the re-direct to work properly in Codio, we need to add a little code.

```
from flask import Flask, render_template, url_for, flash, redirect
from flask_behind_proxy import FlaskBehindProxy
app = Flask(__name__)
proxied = FlaskBehindProxy(app)
```

Passing Success Message

```
<!-- layout.html -->
...
<h1>This shows up on every page</h1>
{% with messages = get_flashed_messages(with_categories=true) %}
  {% if messages %}
    {% for category, message in messages %}
      <div class="alert alert-{{ category }}">
        {{ message }}
      </div>
    {% endfor %}
  {% endif %}
{% endwith %}

{% block content %}{% endblock %}
```

Showing Validation Errors

```
<!-- register.html -->
...
{% extends "layout.html" %}
{% block content %}
    <div class="content-section">
        <form method="POST" action="">
            {{ form.hidden_tag() }}
            <fieldset class="form-group">
<div class="form-group">
    {{ form.username.label(class="form-control-label") }}
    {% if form.username.errors %}
        {{ form.username(class="form-control form-control-lg is-invalid") }}
        <div class="invalid-feedback">
            {% for error in form.username.errors%}
                <span>{{error}}</span>
            {% endfor %}
        </div>
    {% else %}
        {{ form.username(class="form-control form-control-lg") }}
    {% endif %}
</div>

```

Passes data validation errors to user

DEMO Using Forms

Reminders

- Check your Google Calendars for this week's events!
- Kelvin's - Review: Python Object Oriented Programming starts at 1pm EST!!!
 - <https://seousa.zoom.us/j/82930558105>
- There will be no sessions this Thursday (July 4)
- We will have session on Friday (July 5)


polleverywhere Q & A

Q & A

- * Flask
- * Templates
- * Jinja
- * Forms

Nobody has responded yet.

Hang tight! Responses are coming in.



Start the presentation to see live content. For screen share software, share the entire screen. Get help at pollev.com/app

SEO Tech
Developer

Thank you!