

HTML



Full Stack Web Development



Flask

By The Lowell Dev Club Team

{DEV}

Web Pages / Frontend

Creating the visible part of a website also known as the frontend of your website

We will use Hyper Text Markup Language (HTML) to add content and Cascading Style Sheets (CSS) to edit and style html content

Past slides

- Intro

The Workshop

- Go to <https://www.lowelldev.club>
- Check out /workshop or the link on our home page

Full Stack Web Development

1. Repl.it

2. <https://lowelldev.club/workshop/hack1>

3. <https://lowelldev.club/workshop/hack2>

WORKSHOP SLIDES

WORKSHOP CODE

SOURCE CODE

Past slides

- Workshop page
- Intro

Sign up for REPL.IT

- Repl.it is an online code processor
- Sign in if you already have an account

☐

I'm a teacher

or log in

Sign up


By continuing, you agree to Repl.it's [Terms of Service](#) and [Privacy Policy](#), and to receiving emails with updates.


Past slides

- Signup for repl.it
 - Workshop page
 - Intro
- Create a new repl
- Choose Python (Make sure its not Python 2.7)
 - Import the workshop from this link:
<https://github.com/lowell-dev-club/flask-workshop>

×


Create a new repl


 Python

 [https://github.com/lowell-dev-club/flask-w](https://github.com/lowell-dev-club/flask-workshop)

flask-workshop

Describe your repl



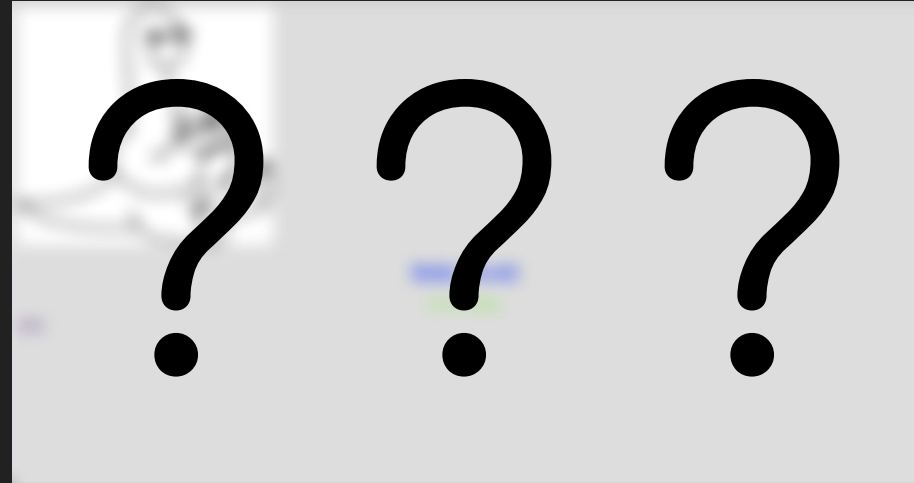
 Upgrade your account for private repls

☐ public

Import from GitHub

Past slides

- Import
repo to
repl
 - Signup for repl.it
 - Workshop
page
 - Intro
- Run the program!
- Run the Code! Repl.it will display your webpage in a side pane it also provides a link you can use to view your whole site.



Past slides

- Running the program
- Import repo to repl
- Signup for repl.it
- Workshop page
- Intro

Customize the index.html

- Head to <https://www.lowelldev.club/workshop/hack1>
- You can use css (stylesheets) and or add other html tags to create your own custom webpage.
- Feel free to ask any leaders for help

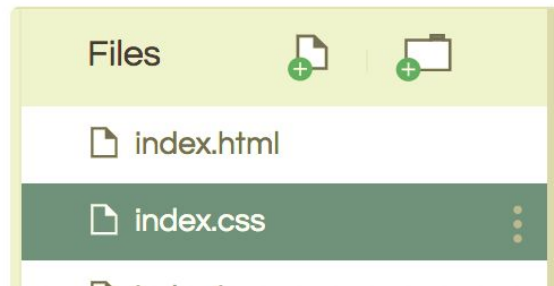
Part III: The CSS File

So what is CSS? CSS, also known as Cascading Style Sheets, is a language used for styling the tags (or “elements”) on a web page.

While HTML oversees the content and the way it’s structured, CSS is how you’ll specify how you’d like your content to look --- with it you can set things like colors, spacing, and more.

1) Using CSS

We already have an `style.css` in the file tree and this is called an external style sheet because the CSS file is external to the HTML file (i.e., the stylesheet is not inside the HTML file).



Flask / Backend

Creating the hidden part of a website also known as the backend of your website

We will use Flask a python framework to create the web server that hosts your website and allows for dynamic web pages

We will use Jinja a template builder for flask that allows for dynamic web pages

Past slides

- Customizing html
- Running the program
- Import repo to repl
- Signup for repl.it
- Workshop page
- Intro

Flask Create Routes

- Import needed variables and functions
- Create routes (URLS) `@app.route("/whatever-you-want")` and choose any html method you would like to allow for the route
- Create functions below flask route decorators and return rendered html files or Jinja

```
1 # Imports
2 from workshop import app
3 from flask import render_template, redirect, url_for
4
5 # Views
6 # Creating a route always requires a "/". You can assign
7 # Assign request method to routes to control if users can
8 @app.route("/", methods=['GET'])
9 @app.route("/home", methods=['GET'])
10 def index():
11     # Render template renders webpage with data creating
12     return render_template('index.html')
13
14 # python array containing any data a developer would like
15 array = ['Hello', 'World', "Dev Club", "Lowell!", "Free f
16
17 @app.route("/advanced", methods=['GET'])
18 def advanced():
19     # Using Jinja and flask's ability to create dynamic we
20     return render_template('advanced.html', array=array)
21
22 # Error Handlers
23 # You can create custom error pages for any html error co
24 # 404 error handler returns 404 page
25 @app.errorhandler(404)
26 def page_not_found(e):
27     return render_template('404.html')
```

Past slides

- Creating Flask Routes
- Customizing html
- Running the program
- Import repo to repl
- Signup for repl.it
- Workshop page
- Intro

Creating Dynamic Pages (Flask)

- Create variable containing any data. Lists, Dictionaries, Strings, etc.
- Pass in the variable with the data into the render template function so Jinja can render it in the template create your dynamic page

```
14 # python array containing any data a developer would like
15 array = ['Hello', 'World', "Dev Club", "Lowell!", "Free food!", "CODE", "PYTHON IS SUPERIOR"]
16
17 @app.route("/advanced", methods=['GET'])
18 def advanced():
19     # Using Jinja and flask's ability to create dynamic webpages we pass in the array in to the Jin
20     return render_template('advanced.html', array=array)
21
```

Past slides

- Creating Dynamic Pages (Flask)
- Creating Flask Routes
- Customizing html
- Running the program
- Import repo to repl
- Signup for repl.it

Jinja preparations

- SEO and meta data setup in base.html
- Create custom blocks and link and needed files

```
46     {% block head_css %}
47     <link rel="stylesheet" type="text/css" href={{ url_for('static', filename='footer.css') }}>
48     {% endblock %}
49
50     {% block head_js %}
51     {% endblock %}
52
53 </head>
54 <body>
55
56     {% block content %}
57     {% endblock %}
58
59     {% block trailing_js %}
60     {% endblock %}
61
62     {% include "footer.html" %}
63
64 </body>
65 </html>
```

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4
5      <!-- Title -->
6      <title>{% block title %}{% endblock %}</title>
7
8      <!-- Meta Data -->
9      <meta charset="utf-8">
10     <meta name="robots" content="index, follow">
11     <meta name="theme-color" content="#ffffff">
12     <meta name="viewport" content="width=device-width, initial-scale=1">
13     <meta name="keywords" content="Lowell High School Dev Club Hack Club">
14     <meta name="description" content="Website for Lowell High School's Dev Club.">
15     <meta name="msapplication-TileColor" content="#da532c">
16
17     <!-- Favicon -->
18     <link rel="/static/img/apple-touch-icon" sizes="180x180" href="/apple-touch-icon.png">
19     <link rel="icon" type="image/png" sizes="32x32" href="/static/img/favicon-32x32.png">
20     <link rel="icon" type="image/png" sizes="16x16" href="/static/img/favicon-16x16.png">
21     <link rel="manifest" href="/static/img/site.webmanifest">
22     <link rel="mask-icon" href="/static/img/safari-pinned-tab.svg" color="#5bbad5">
```

Past slides

- Prepare Jinja
- Creating Dynamic Pages (Flask)
- Creating Flask Routes
- Customizing html
- Running the program
- Import repo to repl

Creating Dynamic Pages (Jinja)

- Extend base.html file
- Use custom Jinja blocks created in base.html
- Use Jinja to create a for loop to put the data passed by Flask into html tags
- Link stylesheet using flask url_for function and inside of custom Jinja block

```
1  {% extends "base.html" %}
2  <!--Extend from base.html and use Jinja block system and allowing
3
4  <!--Jinja custom block allowing for easy title changes-->
5  {% block title %}Advanced Flask!{% endblock %}
6
7  {% block head_css%}
8      <link rel="stylesheet" type="text/css" href={{ url_for('s
9  {% endblock %}
10
11 <!--Content block from base.html-->
12 {% block content %}
13
14     <!--Ordered list of all the item in the python array-->
15     <ol>
16
17     <!--Loop through every item in array-->
18     {% for items in array %}
19
20         <!--Add each item to ordered list-->
21         <li>{{ items }}</li>
22
23     <!--Jinja end loop-->
24     {% endfor %}
25     </ol>
26
27 {% endblock %}
```

Past slides

- Dynamic Pages (Jinja)
- Prepare Jinja
- Creating Dynamic Pages (Flask)
- Creating Flask Routes
- Customizing html
- Running the program

Customize the index.html

- Head to <https://www.lowelldev.club/workshop/hack2>
- Check out Flask docs or ask club leaders and we can help teach specific flask things you may wanna know
- Ask club leaders of any questions you have!

Variable Rules

You can add variable sections to a URL by marking sections with `<variable_name>`. Your function then receives the `<variable_name>` as a keyword argument. Optionally, you can use a converter to specify the type of the argument like `<converter:variable_name>`.

```
@app.route('/user/<username>')
def show_user_profile(username):
    # show the user profile for that user
    return 'User %s' % escape(username)

@app.route('/post/<int:post_id>')
def show_post(post_id):
    # show the post with the given id, the id is an integer
    return 'Post %d' % post_id

@app.route('/path/<path:subpath>')
def show_subpath(subpath):
    # show the subpath after /path/
    return 'Subpath %s' % escape(subpath)
```

Converter types:

<code>string</code>	(default) accepts any text without a slash
<code>int</code>	accepts positive integers
<code>float</code>	accepts positive floating point values
<code>path</code>	like <code>string</code> but also accepts slashes
<code>uuid</code>	accepts UUID strings

The End

This is the end of our work shop this week

We hope you enjoyed learning about Flask and Frontend code

If you want to learn more come back next week!

If you want to refer to the workshop go to <https://www.lowelldev.club/workshop/old>

Our website will be updated over the school year with new and old workshops as well as any resources we have to offer and Club info