

Web Application 2

Jinja2: What's going on with the {{ title }}?


Jinja2 is a template engine, based on Django's templating system. Built for Python, includes an expressive language to better integrate with Pythonic conventions.

So, what does this mean with the double curly brackets?

These Jinja tags allow us to generate variables from Python code, which it will render for our site. This has limitations, as outlined below, but for our needs, it is an incredibly fast and useful tool.

Frequently Asked Questions - Jinja Documentation (3.0.x)

The default syntax of Jinja matches Django syntax in many ways. However this similarity doesn't mean that you can use a Django template unmodified in Jinja. For example filter arguments use a function call syntax rather than a colon to separate filter name and arguments.

 <https://jinja.palletsprojects.com/en/3.0.x/faq/>

So, let's create the recommendations.html template for our recs page. Since we aren't generating this with an f-string in Python, let's replace those variables with the Jinja tags {{ first}} and {{ second }}.

Remember how we named these variables as arguments, let's update our recs section in app.py:

```
def recs():
    results = random_recommender()
    return render_template('recommendations.html', first = results[0], second = results[1])
```

Alternatively, if you're a fan of for-loops (or if want to later add a functionality to change how many results you get):

```
def recs():
    results = random_recommender()
    return render_template('recommendations.html', movies = results)
```

In this case we would make this change to recommendations.html:


```
<ul>
  {% for m in movies %}
  <li> {{ m }} </li>
  {% endfor %}
</ul>
```

In addition to accepting variables within double curly brackets, Jinja will also accept *some* statements within the `{% ... %}` bounds. We can't run list comprehensions or some more complex operations, but for-loops and if-conditionals are available.

Here is a more thorough breakdown from the Jinja documentation:

Template Designer Documentation - Jinja Documentation (3.0.x)

This document describes the syntax and semantics of the template engine and will be most useful as reference to those creating Jinja templates. As the template engine is very flexible, the configuration from the application can be slightly different from the code presented here in terms of delimiters and behavior of undefined values.

 <https://jinja.palletsprojects.com/en/3.0.x/templates/#list-of-control-structures>

Forms

The easiest way to get end-user input is a form. Let's write one to capture a user's opinion of three movies, and then have the submit button take us to the recommendations page.

```
<form action = "/recs">
  <p>Please rate the following movies on a scale of 0 to 5:</p>
  Movie 1:
  <input type = "text" name = "movie1" placeholder = "0 - 5"><br>
  Movie 2:
  <input type = "text" name = "movie2" placeholder = "0 - 5"><br>
  Movie 3:
  <input type = "text" name = "movie3" placeholder = "0 - 5"><br>
  <input type = "submit">
</form>
```

HTML has a built-in form tag with all kinds of useful elements beyond the text fields and submit buttons we use above, including radio buttons, drop-downs, and checkboxes. In

the above example, the end user won't see the name field, but we will be able to use it when parsing the information later.

So where does this information go when we hit submit? Look at the URL and you'll see an addition that should look pretty familiar from many other sites.

How do we use this? Let's take a look at another Flask object.

Request

Flask's Request object has a ton of methods and attributes to help us out, ranging from capturing cookie information to testing for JSON transmission.

flask.Request - Flask API

The Referer[sic] request-header field allows the client to specify, for the server's benefit, the address (URI) of the resource from which the Request-URI was obtained (the "referrer", although the header field is misspelled).

 <https://tedboy.github.io/flask/generated/generated/flask.Request.html>

For this purpose, Request's args attribute is what we will use. args parses the parameters contained in the URL. So let's update `app.py` to make use of this:

```
def recs():
    form_data = dict(request.args)
    results = random_recommendations()
    return render_template("recommendations.html", movies = results, form = form_data)
```

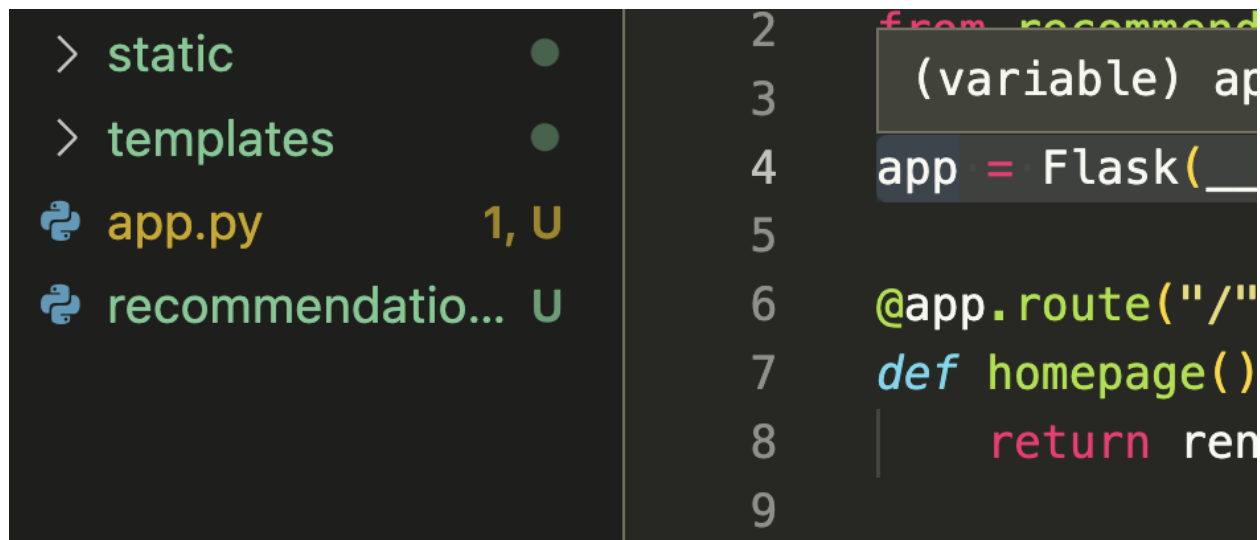
Bonus: Images

Embedding images in Flask can be a little tricky because of where it looks for information, so here's how we can add images to your pages.

First, you need to make a quick change to how you instantiated your Flask object:

```
app = Flask(__name__, static_url_path = '/static')
```

Along with this, you need to add a static folder inside of the parent directory. This means you will have two folders, templates and static.



Once you've done this you can go to where you want the image to be and add a modified version of the usual HTML.

```
<img src = "{{ url_for('static', filename='code_quality.png') }}">
```

The `url_for` method directs Flask to look in the static directory that you assigned before, and then for the filename you chose. If you have subfolders within it (like different pictures for each page or whatever), you can change the filename to include it. For example:

```
<img src = "{{ url_for('static', filename='folder/code_quality.png') }}">
```