Django_templates

February 15, 2018

1 Django Templates

1.1 Template Inheritance

1.1.1 Teacher's Notes

Django Built-In Template Tags and Filters: You should bookmark this page, because you'll refer to it often as you work on Django projects. It's the entire list of built-in template tags and filters that you have access to when you use Django. In other words, it's a gold mine.

Refresher

{% for x in y %} - For loop in Django templates

{% extends "template.html" %} - Causes the current template to extend the quoted template so you can override blocks in the parent template.

{% block name %}{% endblock %} - Marks the start and end of a named block which can be replaced with inheritance.

{% load static from staticfiles %} - Loads the {% static %} tag from the staticfiles library.

{% static "/path/to/file.ext" %} - Generates the URL to the specified file.

1.2 CSS in Django

1.2.1 Teacher's Notes

Managing Static Files: The Django documentation goes into more detail about how Django serves static content in different environments. If you'd like more detail than what we've gone into here, I recommend checking out the docs. Always read the docs!

1.2.2 My Notes

- Best to store global static files in a directory called 'assests' inside your projects main directroy.
- Best to store app specific static files in a directroy structered like so: '///static//'.

1.2.3 Instructions:

Adding App specific CSS to projects

1. Inside /learning_site/courses/, add the follow directory structure and file: ./static/courses/css/courses.css
2. Inside courses.css add the following:

```
.card header a {
      color: #000080;
}
```

3. Inside the head tag, below 'layout.css' link tag of layout.html add the following:

```
{% block static %}{% endblock %}
```

4. Inside of course_list.html add the following:

```
{% load static from staticfiles %}

{% block static %}
<link rel='stylesheet' href="{% static 'courses/css/courses.css' %}">
{% endblock %}
```

1.3 Handy Dandy Filters

1.3.1 Teacher's Note

More on Humanize: The Django documentation on the `humanize` set of filters.

1.3.2 My Notes:

- Remember that a filter is applied to a variable that you pass into your template from your view, and uses the pipe notation.

1.3.3 Instructions:

1. Inside course_detail.html, add the following below `{{ course description }}`:

```
<p>There are {{ course.step_set.count }} step{{ course.step_set.count|pluralize }} in this cou

* `{{ course.step_set.all|length }}` would also work (in place of count), but it creates alot
* `{{ course.step_set.count }}` - Provides the number of steps for course.
* `{{ course.step_set.count|pluralizer }}` - Makes a word plural based on the count.
* `{{ course.step_set.all|join:", " }}` - Produces the nice comma seperated steps. Without joi
```

1.4 Using Template Libraries

1.4.1 Instructions:

Loading a template library

1. Inside /learning_site/learning_site/settings.py, update the INSTALLED_APPS list to the following:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.humanize',
    'courses',
]
```

2. Inside course_detail.html, add the following to the top of the file:

```
{% load humanize %}
```

3. Update the <p> we created earlier to the following:

```
<p>There are {{ course.step_set.count|apnumber }} step{{ course.step_set.count|pluralize }} in

* `apnumber` - For numbers 1-9, returns the number spelled out. Otherwise, returns the number.
```

Shortening variable names on the fly

4. Inside step_detail.html, change `{{ step.content|linebreaks }}` to following:

```
{% with con=step.content %}
    {{ con|linebreaks }}
{% endwith %}
```

1.4.2 Code Challenge

1. Add code that will display how many flavors of ice cream there are in the query set flavors using the count filter.
2. Finally, list all of the flavors of ice cream in the unordered list below. Use either a for tag to list all the flavors in <li> tags or the unordered_list filter.

```
<html>
    <head>
        <title>Ice Cream</title>
    </head>
    <body>
        <div>We have {{ flavors.count }} flavors today.</div>
        <ul>
            {{ flavors|unordered_list }}
        </ul>
    </body>
</html>
```

1.5 Built-in Tags and Features

Django Custom Date Filter

`wordcount` - Counts the words (defined by whitespace) in the variable.

`truncatewords:X` - Ends the variable after X words and appends an ellipsis if any content was cut off.

`urlize` - Converts HTTP(S) and email addresses into HTML anchor tags with appropriate links.

1.5.1 Instructions:

1. Insert and change the following the following into course_list.html:

```
{% block content %}
<div class="cards">
    {% for course in courses %}
    <div class="card">
        <header><a href="{% url 'courses:detail' pk=course.pk %}">{{ course.title }}</a></heade
        <div class="card-copy">
            {% if course.description|wordcount <= 5 %}
                {{ course.description }}
            {% else %}
                {{ course.description|truncatewords:5 }}
                <a href="{% url 'courses:detail' pk=course.pk %}">Read More</a>
            {% endif %}
            <div>Created on: {{ course.created_at|date:"F j, Y" }}</div>
        </div>
    </div>
    {% endfor %}
    <div>Have Questions? Contact Us! {{ email|urlize }}</div>
{% endblock %}
```

1.6 DIY Custom Tags

`template` is Django's module for all things template-related. We'll use this several times in the course.

`template.Library` is a class that lets us register new tags and filters through an instance of itself.

`register.simple_tag(tag_name)` or `@register.simple_tag` - Registers a function as a simple tag. Simple tags don't include new templates, don't have an end tag, and don't assign values to context variables.

1.6.1 Instructions:

1. In /learning_site/courses/ create a new directory called `templatetags` and add the following:

    - An empty `__init__.py` file.
    - `course_extras.py`

- Explaination: Template Tags must live in their own directory called `templatetags` within your app's main directory. For `templatetags` to be recognized as python package, an `__init__.py` file must be added, even if you plan to add nothing to it. `course_extras.py` will contain our custom built tags and filters.

2. In `course_extras.py` add the following:

   - "'python from django import template

   from courses.models import Course

   register = template.Library()

   @register.simple_tag def newest_course(): """" Gets the most recent course that was added to the library. """" return Course.objects.latest('created_at') "'

   ```
   * `created_at` is a column in our Course Model
   ```

3. In `layout.html` add the following:

   - "'html {% load course_extras %}

   Don't miss our latest course, {% newest_course %}!
   "'

4. Start Django Server to test changes.

1.7 Complex Template Tags

Django Documentation Simple Tag vs. Inclusion Tag
`register.inclusion_tag("tag_template.html")(tag_name)` or `@register.inclusion_tag("tag_template.html")` - Registers an inclusion tag. Inclusion tags render a template into wherever they're used.

1.7.1 Instructions for Inclusion tags:

1. In `course_extas.py`, add the following:

   - "'python # add below newest_course()

   @register.inclusion_tag('courses/course_nav.html') def nav_courses_list(): "'Returns dictionary of courses to display as navigation pane."'courses = Course.objects.all() return {'courses': courses}
   "'

2. In `/courses/templates/courses/` create `course_nav.html`. In that file add the following:

   - `python {% for course in courses %} <div><a href="{% url 'courses:detail' pk=course.pk %}">{{ course.title }}</a></div> {% endfor %}`

3. In `layout.html` add the following:

    - `html  <!-- under <div class="site-container"> -->  <div>{% nav_courses_list %}</div>`

4. Run server to check for errors, and that the new navigation menu for our courses is properly displaying.

1.8 Custom Time Filter

Django documentation on custom template filters

`pluralize` - A filter that, by default, returns an "s" when attached to a number that's not 1, and nothing when the number is 1. You can provide different options if needed. More information is available in the official documentation.

`register.filter("filter_name", filter_function) or @register.filter("filter_name")` - Registers a filter with the given name.

`{{ email|urlize }}` - Add a filter that will turn the email variable into a "mailto" link.

1.8.1 Instuctions for creating a filter:

1. In `course_extras.py`, add the following function.

    - `python  @register.filter('time_estimate')  def time_estimate(word_count):  ''' Estimates the number of minutes it will take to complete a step based on the current word count. '''   minutes = round(word_count/20)  return minutes`

2. In `step_detail.html`, add the following:

    - `html  {% load course_extras %}`

1.9 Chainging Filters

Filters are applied from left to right. For example `word|lower|capfirst`, takes a word, puts it in all lowercase, then uppercases the first letter.

Examples of chaining filters from Django Docs

1.9.1 Instructions:

1. In `step_detail.html`, add the following after `Content: {{{ con|wordcount }} words.`

    - `html  Estimated time to complete: {{ con|wordcount|time_estimate }} minute{{ con|wordcount|time_estimate|pluralize }}.`

1.10 Markdown -> HTML Filter

- Documentation for Markdown2 Python library
- Markdown2 on PyPI
- Markdown Syntax Documentation
- Markdown Basics on GitHub
- Documentation on Filters and Auto-escaping

`mark_safe(variable)` - Marks the variable as being safe to send directly to the browser without escaping or encoding the contents beforehand.

1.10.1 Instructions for transforming markdown to HTML w/ custom filter:

1. At the command line, type: `pip install markdown2`.
2. In `course_extras.py`, `import markdown2` and Runserver to check for import errors.
    - Now add the following function:

```
python  @register.filter('markdown_to_html')  def markdown_to_html(markdown_text):
''' Converts markdown text to html. '''      html_body =
markdown2.markdown(markdown_text)      return html_body
```

3. At the command line, type: `python manage.py createsuperuser`. Fill in the credentials as you choose.
    - Now type: `python manage.py runserver`
    - Add `/admin` to the end of local server's web address within your browser.
4. After filling in your credentials for the Django admin, go to the Python Testing Course, and change the course description to the following:

    ‘‘‘ ## Testing is Amazing!

    Learn to test your Python applications with unittest and doctests!

    Things you will learn:

    - unittest
    - doctests ‘‘‘
5. In `course_detail.html` add the following after `{{ course.description: html |markdown_to_html }}`

```
* You can add the `safe` filter after `markdown_to_html` to have it actually render the H
```

    - Add `{% load course_extras %}` as well.
6. In `course_extras.py`, type: `from django.utils.safestring import mark_safe`.
    - Now change the return statment in the `markdown_to_html` function to the following: `return mark_safe(html_body)`