django-pygmentify Documentation

Release 0.1

Richard Cornish

Oct 06, 2018

Contents

```python
from django.db import models
from django.core.urlresolvers import reverse

class Post(models.Model):
    title = models.CharField(max_length=255)
    slug = models.SlugField(unique=True)
    body = models.TextField()
    pub_date = models.DateTimeField("Pub date")

    class Meta:
        ordering = ["-pub_date"]

    def __str__(self):
        return self.title

    def get_absolute_url(self):
        return reverse('blog:post_detail', args=[self.slu
```

Django Pygmentify is a Django template tag application to highlight code with Pygments.

It is an alternative to Django Pygments, which hasn't been updated in several years.

- Package distribution
- Code repository
- Documentation
- Tests

CHAPTER 1

Install

```
$ pip install django-pygmentify
```

Add to `settings.py`.

```
INSTALLED_APPS = [
    # ...
    'pygmentify',
]
```

CHAPTER 2

Usage

```
{% load pygmentify_tags %}

<link rel="stylesheet" href="{% pygmentify_css %}">

{% pygmentify %}
<pre class="python">
print('Hello, world!')
</pre>
{% endpygmentify %}
```

Result:

```
<link rel="stylesheet" href="/static/pygmentify/css/default.min.css">

<div class="highlight"><pre class="python"><span></span><span class="k">print</span>
↪<span class="p">(</span><span class="s2">&quot;Hello, world!&quot;</span><span␣
↪class="p">)</span>
</pre></div>
```

CHAPTER 3

Contents

3.1 Install

Install with the pip package manager.

```
$ mkvirtualenv myvenv -p python3
$ pip install django
$ pip install django-pygmentify
```

After creating a project, add `pygmentify` to `INSTALLED_APPS` in `settings.py`.

```
INSTALLED_APPS = [
    # ...
    'pygmentify',
]
```

Remember to update your `requirements.txt` file. In your project directory:

```
$ pip freeze > requirements.txt
```

3.2 Usage

3.2.1 HTML

Load the template tags.

```
{% load pygmentify_tags %}
```

Use the `{% pygmentify %}` template tag to convert HTML into Pygments HTML.

```
{% pygmentify %}
<pre class="python">
print('Hello, world!')
</pre>
{% endpygmentify %}
```

Result:

```
<div class="highlight"><pre class="python"><span></span><span class="k">print</span>
↪<span class="p">(</span><span class="s2">&quot;Hello, world!&quot;</span><span␣
↪class="p">)</span>
</pre></div>
```

The `{% pygmentify %}` template tag expects an opening `<pre>` tag with a `class` attribute of the programming language that you are using that matches against the short name of the corresponding Pygment lexer. For example, `<pre class="python">` uses the Python lexer, and `<pre class="html">` uses the HTML lexer to highlight code.

If no CSS class is present, the template tag makes a best guess using heuristics of the code between the `<pre>` tags. If multiple CSS classes are present, the classes are collected into a list and are iterated against all available lexers.

Pygments's default behavior strips your customized `<pre class="...">` tag and replaces it with a plain `<pre>` tag. The template tag undoes this behavior and preserves the customized `<pre class="...">` tag.

See the Pygments documentation for all language short names. There's even a Django template one.

3.2.2 CSS

Use the `{% pygmentify_css %}` template tag to output the URL of the CSS file.

```
<link rel="stylesheet" href="{% pygmentify_css %}">
```

Result:

```
<link rel="stylesheet" href="/static/pygmentify/css/default.min.css">
```

The default output of the style file is the minified version.

The way that Pygments generates CSS is awkward. Rather than provide CSS files, Pygments abstracts a more generalized style language into Python classes to create styles that can be used with formatters other than HTML. Therefore, the template tag provides exports of the default styles using the `pygmentize` command and `clean-css` library.

```
$ pygmentize -S <style> -f html > <style>.css
```

```
$ cleancss <style>.css -o <style>.min.css
```

Please remember to put the `<link>` tag in the `<head>` of your document.

3.2.3 Minimalist example

The bare minimum to highlight your code is to use the `{% pygmentify %}` tag and to load a corresponding style with the `{% pygmentify_css %}` tag.

```
{% load pygmentify_tags %}

<link rel="stylesheet" href="{% pygmentify_css %}">

{% pygmentify %}
<pre class="python">
print('Hello, world!')
</pre>
{% endpygmentify %}
```

The `default.min.css` style file will be used in this example.

Please ensure that the code to highlight contains HTML either natively or by conversion (by, say Markdown) because the template tag will look for fully rendered HTML.

3.2.4 Customize with positional arguments

Customize the behavior of the `{% pygmentify_css %}` and `{% pygmentify %}` tags by passing the name of a style as a positional argument.

```
{% load pygmentify_tags %}

<link rel="stylesheet" href="{% pygmentify_css 'monokai' %}">

{% pygmentify 'monokai' %}
<pre class="python">
print('Hello, world!')
</pre>
{% endpygmentify %}
```

The `monokai.min.css` style file will be used in this example.

The name of a style is the only possible positional argument available to `{% pygmentify_css %}` and `{% pygmentify %}`.

If you customize the style, please ensure you pass the same argument, e.g. `'monokai'`, to *both* the `{% pygmentify_css %}` and `{% pygmentify %}` tags. You might see unexpected behavior otherwise because "not all lexers might support every style," meaning styles are guaranteed to work fully only when the lexer assigns to tokens HTML classes that correspond to the class selectors in the CSS file. Therefore, you're probably better off customizing the style by changing the *Settings* of the project. Template tag arguments take precedence over settings. Also see *Settings* for creating your own styles.

3.2.5 Customize with keyword arguments

Additionally customize the behavior of the `{% pygmentify_css %}` and `{% pygmentify %}` tags with keyword arguments.

The `{% pygmentify_css %}` can accept the `style` and `minify` keyword arguments.

```
{% pygmentify_css style='monokai' minify='false' %}
```

The `monokai.css` style file will be used in this example.

Note that because Django's template language is not Python, the `{% pygmentify_css %}` "keyword arguments" are expected to be strings. Therefore, most notably, use `'true'` or `'false'` for the `minify` keyword argument.

You will probably want the minified file, so use `'true'`–or even better omit the keyword argument all together because the default style file to use is the minified file.

Therefore:

```
{% pygmentify_css style='default' minify='true' %}
```

is equivalent to...

```
{% pygmentify_css 'default' %}
```

which is also equivalent to...

```
{% pygmentify_css %}
```

Additionally, the `{% pygmentify %}` tag accepts all available options of Pygments's `HtmlFormatter` class, such as `style` and `linenos`, as keyword arguments.

```
{% pygmentify style='monokai' cssclass='bettercssclass' linenos='true' linenostart=0
↪%}
<pre class="python">
print('Hello, world!')
</pre>
{% endpygmentify %}
```

Again, because Django's template language is not Python, template tags expect either a string or a number as a keyword argument. Therefore, in instances when Pygments's `HtmlFormatter` constructor expects a Python data type, such as a string, number, boolean, or list, the value of the keyword argument should be the equivalent string or number. For example, pass `'true'` as the equivalent of `True` or `'[...]'` as the equivalent of `[...]`. Numbers can be left as is. All keyword arguments are later coerced into Python data types.

See Pygments's documentation on the `HtmlFormatter` class for all available keyword arguments.

3.3 Settings

The template tag offers two settings. By default, they are:

```
PYGMENTIFY = {
    'style': 'default',
    'cssclass': 'highlight'
}

PYGMENTIFY_MINIFY = True
```

3.3.1 PYGMENTIFY

A dictionary corresponding to the keyword argument options of Pygments's `HtmlFormatterClass`, which means you're free to use any of the 20+ options. The default options are:

- `style` is a string indicating the Pygments style class to use.
- `cssclass` is a string indicating the class of the `<div>` element that wraps the highlighted code.

The up-to-date list of styles is in the Pygments repository, but generally speaking, the styles from which to choose are:

- `algol`

- algol_nu
- arduino
- autumn
- borland
- bw
- colorful
- default
- emacs
- friendly
- fruity
- igor
- lovelace
- manni
- monokai
- murphy
- native
- paraiso-dark
- paraiso-light
- pastie
- perldoc
- rrt
- tango
- trac
- vim
- vs
- xcode

Preview these styles by visiting any of the Pygments demo entries.

This setting is also available on a per-template basis, but by setting the value of the `style` key of `PYGMENTIFY` once, the template tag automatically sets the correct Pygments HTML output *and* corresponding CSS to use. See examples in *Usage* for details.

If you want to create your own style, follow the Pygments documentation by creating a `Style` class, registering it as a plugin, and passing its `name` attribute to the value of the `style` key of the `PYGMENTIFY` setting.

3.3.2 PYGMENTIFY_MINIFY

- A boolean indicating the serving of a minified CSS file. The app serves the minified file by default.

3.4 Documentation

Full documentation is available online.

However, you can also build the documentation from source. Enter your virtual environment.

```
$ workon myvenv
```

Clone the code repository.

```
$ git clone git@github.com:richardcornish/django-pygmentify.git
$ cd django-pygmentify/
```

Install Sphinx, `sphinx-autobuild`, and `sphinx_rtd_theme`.

```
$ pip install sphinx sphinx-autobuild sphinx_rtd_theme
```

Create an HTML build.

```
$ (cd docs/ && make html)
```

Or use `sphinx-autobuild` to watch for live changes.

```
$ sphinx-autobuild docs/ docs/_build_html
```

Open 127.0.0.1:8000.

3.5 Tests

Continuous integration test results are available online.

However, you can also test the source code.

```
$ workon myvenv
$ django-admin test pygmentify.tests --settings="pygmentify.tests.settings"

Creating test database for alias 'default'...
.........
----------------------------------------------------------------------
Ran 9 tests in 0.326s

OK
Destroying test database for alias 'default'...
```

A bundled settings file allows you to test the code without even creating a Django project.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

P