Django

Documentation

Coding style

Please follow these coding standards when writing code for inclusion in Django.

Python style

- Unless otherwise specified, follow **PEP 8**.

  Use flake8 to check for problems in this area. Note that our **`setup.cfg`** file contains some excluded files (deprecated modules we don't care about cleaning up and some third-party code that Django vendors) as well as some excluded errors that we don't consider as gross violations. Remember that **PEP 8** is only a guide, so respect the style of the surrounding code as a primary goal.

  An exception to **PEP 8** is our rules on line lengths. Don't limit lines of code to 79 characters if it means the code looks significantly uglier or is harder to read. We allow up to 119 characters as this is the width of GitHub code review; anything longer requires horizontal scrolling which makes review more difficult. This check is included when you run **`flake8`**. Documentation, comments, and docstrings should be wrapped at 79 characters, even though **PEP 8** suggests 72.

- Use four spaces for indentation.

- Use underscores, not camelCase, for variable, function and method names (i.e. **`poll.get_unique_voters()`**, not **`poll.getUniqueVoters`**).

- Use **`InitialCaps`** for class names (or for factory functions that return classes).

- Use convenience imports whenever available. For example, do this:

  ```
  from django.views.generic import View
  ```

  Don't do this:

  ```
  from django.views.generic.base import View
  ```

- In docstrings, use "action words" such as:

  ```
  def foo():
      """
      Calculates something and returns the result.
      """
      pass
  ```

  Here's an example of what not to do:

```
def foo():
    """
    Calculate something and return the result.
    """
    pass
```

Template style

- In Django template code, put one (and only one) space between the curly brackets and the tag contents.

  Do this:

  ```
  {{ foo }}
  ```

  Don’t do this:

  ```
  {{foo}}
  ```

View style

- In Django views, the first parameter in a view function should be called **`request`**.

  Do this:

  ```
  def my_view(request, foo):
      # ...
  ```

  Don’t do this:

  ```
  def my_view(req, foo):
      # ...
  ```

Model style

- Field names should be all lowercase, using underscores instead of camelCase.

  Do this:

```
class Person(models.Model):
    first_name = models.CharField(max_length=20)
    last_name = models.CharField(max_length=40)
```

Don’t do this:

```
class Person(models.Model):
    FirstName = models.CharField(max_length=20)
    Last_Name = models.CharField(max_length=40)
```

- The **`class Meta`** should appear *after* the fields are defined, with a single blank line separating the fields and the class definition.

  Do this:

```
class Person(models.Model):
    first_name = models.CharField(max_length=20)
    last_name = models.CharField(max_length=40)

    class Meta:
        verbose_name_plural = 'people'
```

  Don’t do this:

```
class Person(models.Model):
    first_name = models.CharField(max_length=20)
    last_name = models.CharField(max_length=40)
    class Meta:
        verbose_name_plural = 'people'
```

  Don’t do this, either:

```
class Person(models.Model):
    class Meta:
        verbose_name_plural = 'people'

    first_name = models.CharField(max_length=20)
    last_name = models.CharField(max_length=40)
```

- If you define a **`__str__`** method (previously **`__unicode__`** before Python 3 was supported), decorate the model class with **`python_2_unicode_compatible()`**.
- The order of model inner classes and standard methods should be as follows (noting that these are not all required):
  - All database fields

    - Custom manager attributes
    - **class Meta**
    - **def __str__()**
    - **def save()**
    - **def get_absolute_url()**
    - Any custom methods
- If **choices** is defined for a given model field, define each choice as a tuple of tuples, with an all-uppercase name as a class attribute on the model. Example:

```
class MyModel(models.Model):
    DIRECTION_UP = 'U'
    DIRECTION_DOWN = 'D'
    DIRECTION_CHOICES = (
        (DIRECTION_UP, 'Up'),
        (DIRECTION_DOWN, 'Down'),
    )
```

Use of **django.conf.settings**

Modules should not in general use settings stored in **django.conf.settings** at the top level (i.e. evaluated when the module is imported). The explanation for this is as follows:

Manual configuration of settings (i.e. not relying on the **DJANGO_SETTINGS_MODULE** environment variable) is allowed and possible as follows:

```
from django.conf import settings

settings.configure({}, SOME_SETTING='foo')
```

However, if any setting is accessed before the **settings.configure** line, this will not work. (Internally, **settings** is a **LazyObject** which configures itself automatically when the settings are accessed if it has not already been configured).

So, if there is a module containing some code as follows:

```
from django.conf import settings
from django.core.urlresolvers import get_callable

default_foo_view = get_callable(settings.FOO_VIEW)
```

...then importing this module will cause the settings object to be configured. That means that the ability for third parties to import the module at the top level is incompatible with the ability to configure the settings object manually, or makes it very difficult in some circumstances.

Instead of the above code, a level of laziness or indirection must be used, such as **`django.utils.functional.LazyObject`**, **`django.utils.functional.lazy()`** or **`lambda`**.

Miscellaneous

- Mark all strings for internationalization; see the i18n documentation for details.
- Remove **`import`** statements that are no longer used when you change code. flake8 will identify these imports for you. If an unused import needs to remain for backwards-compatibility, mark the end of with **`# NOQA`** to silence the flake8 warning.
- Systematically remove all trailing whitespaces from your code as those add unnecessary bytes, add visual clutter to the patches and can also occasionally cause unnecessary merge conflicts. Some IDE's can be configured to automatically remove them and most VCS tools can be set to highlight them in diff outputs.
- Please don't put your name in the code you contribute. Our policy is to keep contributors' names in the **`AUTHORS`** file distributed with Django – not scattered throughout the codebase itself. Feel free to include a change to the **`AUTHORS`** file in your patch if you make more than a single trivial change.

Writing code

Unit tests