django-fractions Documentation

Release 5.0.0

Justin Michalicek

Jan 09, 2023

Contents

Contents:

CHAPTER 1

django-fractions

pypi package 3.1.0

Fraction display and form fields for Django

1.1 Documentation

The full documentation is at https://django-fractions.readthedocs.org.

1.2 Quickstart

Install django-fractions:

```
pip install django-fractions
```

Add `djfractions` to `settings.INSTALLED_APPS`

Then use it in a project:

```
import djfractions
```

In templates:

```
{% load fractions %}
{% display_fraction 1.25 %}
```

In Forms:

```
from djfractions.forms import DecimalFractionField
from django import forms

class MyForm(forms.Form):
    a_fraction = DecimalFractionField()
```

1.3 Features

- Template tag for displaying float and Decimal values as fractions including mixed numbers
- DecimalFractionField form field which handles input such as "1/4", "1 1/2", "1 and 1/2", and converts to a decimal.Decimal instance

1.4 TODO

- Add unicode_fraction template tag to display the unicode fraction entity if available
- forms.FloatDecimalField to return a float rather than Decimal
- forms.SplitFractionWidget for having separate numerator and denominator form fields
- forms.SplitMixedFractionWidget for handling mixed number fractions with separate fields
- models.DecimalBackedFractionField() to store a Decimal value but return/accept it as a fraction
- models.FloatBackedFractionField() to store a Decimal value but return/accept it as a fraction

1.5 Cookiecutter Tools Used in Making This Package

- cookiecutter
- cookiecutter-djangopackage

CHAPTER 2

Installation

At the command line:

```
$ easy_install django-fractions
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv django-fractions
$ pip install django-fractions
```

After installation add `djfractions` to your `settings.INSTALLED_APPS`

CHAPTER 3

Usage

Add `djfractions` to `settings.INSTALLED_APPS`

3.1 Model Fields

3.2 DecimalFractionField

```
djfractions.models.DecimalFractionField(verbose_name=None,
                                        name=None,
                                        max_digits=None,
                                        decimal_places=None,
                                        limit_denominator=None,
                                        coerce_thirds=True,
                                        **kwargs)
```

Takes a `fractions.Fraction` value, stores it as a decimal value, and then returns it as a `fractions.Fraction`. This field is highly based on Django's `models.DecimalField` implementation and so the *max_digits* and *decimal_places* arguments are required.

param str verbose_name The verbose name of the field

param str name Name of the field

param int max_digits Maximum number of digits to use for the Decimal representation

param int decimal_places Maximum number of decimal places to use for the Decimal representation

param int limit_denominator Limits the fraction's denominator to this value if it is set.

paraam bool coerce_thirds If True, then when values which appear to be Decimal values which started as 1/3 or 2/3 will be forced back to 1/3 or 2/3 when retrieved from the database.

3.3 Form Fields

3.3.1 FractionField

```
FractionField(max_value=None,
              min_value=None,
              coerce_thirds=True,
              limit_denominator=None,
              use_mixed_numbers=True)
```

Returns a `fractions.Fraction` instance. Takes a string formatted as a fraction such as 1/4, 1 1/4, 1-1/4, 1 and 1/4, or -1/4 as input in a form.

param Decimal max_value The maximum value allowed for this field

param Decimal min_value The minimum value allowed for this field

param int limit_denominator Limits the fraction's denominator to this value if it is set.

param bool coerce_thirds If True, then when values which appear to be Decimal values which started as 1/3 or 2/3 will be forced back to 1/3 or 2/3 when retrieved from the database.

param bool use_mixed_numbers If True initial values which are decimals and floats greater than 1 will be converted to a mixed number such as *1 1/2* in the form field's value. If False then improper fractions such as *3/2* will be created. Defaults to True.

Example:

```
from django import forms
from djfractions.forms import FractionField

class MyForm(forms.Form):
    a_fraction = FractionField()
```

3.3.2 DecimalFractionField

```
DecimalFractionField(max_value=None,
                     min_value=None,
                     coerce_thirds=True,
                     limit_denominator=None,
                     use_mixed_numbers=True,
                     max_digits=None,
                     decimal_places=None)
```

Returns a `decimal.Decimal` instance. Takes a string formatted as a fraction such as 1/4, 1 1/4, 1-1/4, 1 and 1/4, or -1/4 as input in a form.

param bool coerce_thirds Defaults to True. If True then .3 repeating is forced to 1/3 rather than 3/10, 33/100, etc. and .66 and .67 are forced to 2/3.

param int limit_denominator Set a maximum denominator to be used on fractions created from the field input.

param bool use_mixed_numbers If True initial values which are decimals and floats greater than 1 will be converted to a mixed number such as *1 1/2* in the form field's value. If False then improper fractions such as *3/2* will be created. Defaults to True.

param max_value The maximum value allowed

param min_value The minimum value allowed

param int decimal_places The maximum number of decimal places the resulting Decimal value may have

param int max_digits The maximum number of digits, including decimal places, the resulting Decimal may have.

Example:

```
from django import forms
from djfractions.forms import DecimalFractionField

class MyForm(forms.Form):
    a_fraction = DecimalFractionField()
```

3.4 Template Tags

3.4.1 display_fraction

`{% display_fraction value limit_denominator allow_mixed_numbers coerce_thirds %}`

The display_fraction tag displays a formatted fraction in an HTML template. It takes a value and optional parameters to limit the denominator, allow mixed numbers, and adjust decimal/float values which usually are the result of rounding thirds back to thirds based fractions.

The output of this tag can be changed by overriding the `djfractions/display_fraction.html` template. This is because there are a number of style choices you might make depending on needs. In some cases <sup> and <sub> tags may cause issues with screen readers. You may just want to add css classes for easier styling. The template context also includes a `unicode_entity` value which has the html entity for the unicode value of a fraction if one is available. The unicode html entity is preferred by some people, but only a small number of fractions are supported (particularly if you must support very old browsers) and the styling is frequently difficult to match up exactly with <sup> and <sub> tags.:

```
{% load fractions %}
{% display_fraction 1.5 %}
```

Would output:

```
1 <sup>1</sup>&frasl;<sub>2</sub>
```

The template context:

whole_number The whole number part of a fraction. If `allow_mixed_numbers` is False then this will always be 0.

numerator The numerator of a fraction. For values which are only a whole number the numerator will be 0.

denominator The denominator of a fraction. For values which are only a whole number the denominator will be 1 for a fraction of 0/1.

unicode_entity The unicode_entity is the html entity for the unicode fraction if one exists.

allow_mixed_numbers The value passed to the tag for `allow_mixed_numbers`. Knowing this can be useful in template display logic.

The following unicode fraction HTML entities are supported by django-fractions. They may not all be supported by your browser.

Entity	IE 11	Firefox 39	Chrome 44
&frac12;	Yes	Yes	Yes
&frac13;	Yes	Yes	Yes
&frac23;	Yes	Yes	Yes
&frac14;	Yes	Yes	Yes
&frac34;	Yes	Yes	Yes
&frac15;	Yes	Yes	Yes
&frac25;	Yes	Yes	Yes
&frac35;	Yes	Yes	Yes
&frac45;	Yes	Yes	Yes
&frac16;	Yes	Yes	Yes
&frac56;	Yes	Yes	Yes
&frac17;	No	No	Yes
&frac18;	Yes	Yes	Yes
&frac38;	Yes	Yes	Yes
&frac58;	Yes	Yes	Yes
&frac78;	Yes	Yes	Yes

3.4.2 display_improper_fraction

`{% display_improper_fraction value limit_denominator coerce_thirds %}`

The display_improper_fraction tag works the same as display_fraction with its allow_mixed_numbers set to False. It is just a shortcut for a common use case.:

```
{% load fractions %}
{% display_improper_fraction 1.5 %}
```

Would output:

```
<sup>3</sup>&frasl;<sub>2</sub>
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/jmichalicek/django-fractions/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

4.1.4 Write Documentation

django-fractions could always use more documentation, whether as part of the official django-fractions docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/jmichalicek/django-fractions/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *django-fractions* for local development.

These instructions are out of date. I have moved to developing in a docker container in a docker compose stack. I have not yet set that up to deal with multiple Python versions for tox.

1. Fork the *django-fractions* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/django-fractions.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv django-fractions
   $ cd django-fractions/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 djfractions tests
$ python setup.py test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.3, and 3.4. Check https://travis-ci.org/jmichalicek/django-fractions/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_djfractions
```

CHAPTER 5

Credits

5.1 Development Lead

- Justin Michalicek <jmichalicek@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 Current

6.2 5.0.0 (2023-01-08)

- Forgot about the 4.0.0 mess when I published 3.1.0, so starting fresh again. There are not really any backwards incompatible changes. See the 3.1.0 release notes.
- Fixing documentation and release history.

6.3 4.0.0 (2022-08-10)

- I'm dumb and goofed thing up and so this version exists on pypi

6.4 3.1.0 (2022-12-26)

- Add testing on python 3.11 by @jmichalicek in #30
- Fixed 'Polynomial regular expression used on uncontrolled data' codeQL issues which could be abused to impact performance.

6.5 3.0.0 (2022-08-10)

- Create codeql-analysis.yml by @jmichalicek in #21
- Add type checking by @jmichalicek in #22
- Update supported python versions to 3.7+ by @jmichalicek in #23

- fix: Django 4.0 deprecations by @GriceTurrble in #24
- Add testing of django 4.0 by @jmichalicek in #27
- Updated to support Django 4.1, drop old unsupported Django and Python versions by @jmichalicek in #29

6.6 2.0.0 (2020-09-13)

- Dropped support for Django 2.0 and lower and bump major version
- Dropped support for Python 3.4 and lower
- Updated DecimalFractionField to work on Django 2+ where from_db_field() does not take a context argument and where the field is expected to have a *context* attribute like DecimalField.
- Cleared out a bunch of Python 2 compatibility code such as how calls to super() are made, usage of six, and from __future__ imports.

6.7 1.1.0 (2017-06-04)

- add python 3.6 and django 1.11 to tox.ini - still a bit broken
- convert to matrix for environments in .travis.yml because tox only wants to test py3.6 when installed under 3.6 but will not test 3.5 when running with python 3.6 as the base.
- Remove invalid ROOT_URLCONF from test django config There is no urls.py for djfractions, don't tell it to use one. Older django versions were ok with this, but 1.11 is pickier about the correctness.
- add current changes to HISTORY.rst
- Adjust SILENCED_SYSTEM_CHECKS setting during tests Django 1.11 is stricter about system checks and will not even run the tests where there are some errors we specifically test for due to older django versions letting you make these mistakes.
- Added optional max_digits and decimal_places parameters to forms.DecimalFractionField so that returned Decimal objects have the desired max_digits and decimal_places when not directly tied to a models.DecimalField() on a ModelForm

6.8 1.0.0 (2016-12-31)

- Stop subclassing Django's DecimalField and duplicate small amounts of code as necessary for db backend compatibility. Too many things need to be handled differently. Main cause of major version bump.
- Update forms.FractionField to skip over max_digits and decimal_places kwargs which will get passed in by models.fields.DecimalFractionField
- Add models.fields.DecimalFractionField.formfield() so that a forms.FractionField will be used by default
- Fix quantity_to_decimal and quantity_to_fraction to strip leading and trailing spaces before pattern matching and converting to a decimal or fraction
- Allow for leading negative sign with forms.FractionField input values
- Fix is_fraction() to allow leading negative sign
- Add *max_digits* and *decimal_places* params to DecimalFractionField in test model
- Additional test cases for models.fields.DecimalFractionField

6.9 0.4.0 (2016-08-29)

- Added djfractions.models.DecimalFractionField which stores fractions.Fraction values as decimals in the dataase.
- Better usage of tox to test against different Python and Django versions
- Added testing against Django 1.10

6.10 0.3.2 (2015-08-28)

- Fixed boolean logic for when to coerce values to thirds in in forms.DecimalFractionField and get_fraction_parts()

6.11 0.3.1 (2015-08-12)

- HISTORY.rst typo fixes
- pypi release version fix

6.12 0.3.0 (2015-08-12)

- Added forms.FractionField which returns fractions.Fraction instances
- Refactoring of common code with new forms.FractionField
- Smarter checking for numeric types throughout the code
- forms.DecimalFractionField.to_python() handles fractions.Fraction values now
- Fixed bug handling negative numbers in quantity_to_decimal()
- Added min_value and max_value to forms.DecimalFractionField
- Made coerce_thirds, limit_denominator, and use_mixed_numbers params to DecimalFractionField proper named parameters and not just kwargs.

6.13 0.2.1 (2015-08-06)

- Fixed typo in usage docs

6.14 0.2.0 (2015-08-06)

- display_fraction template tag output is templated so that its formatting can be changed by users
- Added new display_improper_fraction template tag to simplify the common case of wanting to only use improper fractions with no whole numbers
- Added unicode_entity to template context for display_fraction and display_improper_fraction so that the html entity for common fractions may be used rather than <sup> and <sub> tags

- Refactored lots of code out into smaller, reusable functions
- Added a bunch of test cases

6.15 0.1.0 (2015-08-01)

- First release on PyPI.