Choices Enum Documentation

Release 0.7.0

Fernando Macedo

Mar 01, 2022

CONTENTS

Contents:

CHAPTER

ONE

CHOICES ENUM

Python's Enum with extra powers to play nice with labels and choices fields.

- Free software: BSD license
- Documentation: https://python-choicesenum.readthedocs.io.

1.1 Installation

Install `choicesenum` using pip:

```
$ pip install choicesenum
```

1.2 Features

- An `ChoicesEnum` that can be used to create constant groups.
- `ChoicesEnum` can define labels to be used in *choices* fields.
- Django fields included: `EnumCharField` and `EnumIntegerField`.
- All `ChoicesEnum` types can be compared against their primitive values directly.
- Support (tested) for Python 2.7, 3.5, 3.6, 3.7 and 3.8.
- Support (tested) for Django 1.9, 1.10, 1.11, 2.0, 2.1, 2.2 and 3.0.

1.3 Usage examples

Example with `HttpStatuses`:

```
class HttpStatuses(ChoicesEnum):
    OK = 200
    BAD_REQUEST = 400
    UNAUTHORIZED = 401
    FORBIDDEN = 403
```

Example with `Colors`:

```
from choicesenum import ChoicesEnum

class Colors(ChoicesEnum):
    RED = '#f00', 'Vermelho'
    GREEN = '#0f0', 'Verde'
    BLUE = '#00f', 'Azul'
```

1.3.1 Comparison

All *Enum* types can be compared against their values:

```
assert HttpStatuses.OK == 200
assert HttpStatuses.BAD_REQUEST == 400
assert HttpStatuses.UNAUTHORIZED == 401
assert HttpStatuses.FORBIDDEN == 403

status_code = HttpStatuses.OK
assert 200 <= status_code <= 300

assert Colors.RED == '#f00'
assert Colors.GREEN == '#0f0'
assert Colors.BLUE == '#00f'
```

1.3.2 Label for free

All *Enum* types have by default a *display* derived from the enum identifier:

```
assert HttpStatuses.OK.display == 'Ok'
assert HttpStatuses.BAD_REQUEST.display == 'Bad request'
assert HttpStatuses.UNAUTHORIZED.display == 'Unauthorized'
assert HttpStatuses.FORBIDDEN.display == 'Forbidden'
```

You can easily define your own custom display for an *Enum* item using a tuple:

```
class HttpStatuses(ChoicesEnum):
    OK = 200, 'Everything is fine'
    BAD_REQUEST = 400, 'You did a mistake'
    UNAUTHORIZED = 401, 'I know your IP'
    FORBIDDEN = 403

assert HttpStatuses.OK.display == 'Everything is fine'
assert HttpStatuses.BAD_REQUEST.display == 'You did a mistake'
```

(continues on next page)

(continued from previous page)

```
assert HttpStatuses.UNAUTHORIZED.display == 'I know your IP'
assert HttpStatuses.FORBIDDEN.display == 'Forbidden'
```

1.3.3 Dynamic properties

For each enum item, a dynamic property `is_<enum_item>` is generated to allow quick boolean checks:

```
color = Colors.RED
assert color.is_red
assert not color.is_blue
assert not color.is_green
```

This feature is usefull to avoid comparing a received enum value against a know enum item.

For example, you can replace code like this:

```
# status = HttpStatuses.BAD_REQUEST

def check_status(status):
    if status == HttpStatuses.OK:
        print("Ok!")
```

To this:

```
def check_status(status):
    if status.is_ok:
        print("Ok!")
```

1.3.4 Custom methods and properties

You can declare custom properties and methods:

```
class HttpStatuses(ChoicesEnum):
    OK = 200, 'Everything is fine'
    BAD_REQUEST = 400, 'You did a mistake'
    UNAUTHORIZED = 401, 'I know your IP'
    FORBIDDEN = 403

    @property
    def is_error(self):
        return self >= self.BAD_REQUEST

assert HttpStatuses.OK.is_error is False
assert HttpStatuses.BAD_REQUEST.is_error is True
assert HttpStatuses.UNAUTHORIZED.is_error is True
```

1.3.5 Iteration

The enum type is iterable:

```
>>> for color in Colors:
...     print(repr(color))
Color('#f00').RED
Color('#0f0').GREEN
Color('#00f').BLUE
```

Order is guaranteed only for py3.4+. For fixed order in py2.7, you can implement a magic attribute `_order_`:

```
from choicesenum import ChoicesEnum

class Colors(ChoicesEnum):
    _order_ = 'RED GREEN BLUE'

    RED = '#f00', 'Vermelho'
    GREEN = '#0f0', 'Verde'
    BLUE = '#00f', 'Azul'
```

1.3.6 Choices

Use `.choices()` method to receive a list of tuples `(item, display)`:

```
assert list(Colors.choices()) == [
    ('#f00', 'Vermelho'),
    ('#0f0', 'Verde'),
    ('#00f', 'Azul'),
]
```

1.3.7 Values

Use `.values()` method to receive a list of the inner values:

```
assert Colors.values() == ['#f00', '#0f0', '#00f', ]
```

1.3.8 Options

Even if a `ChoicesEnum` class is an iterator by itself, you can use `.options()` to convert the enum items to a list:

```
assert Colors.options() == [Colors.RED, Colors.GREEN, Colors.BLUE]
```

1.3.9 A “dict like” get

Use `.get(value, default=None)` method to receive `default` if `value` is not an item of enum:

```
assert Colors.get(Colors.RED) == Colors.RED
assert Colors.get('#f00') == Colors.RED
assert Colors.get('undefined_color') is None
assert Colors.get('undefined_color', Colors.RED) == Colors.RED
```

1.3.10 Compatibility

The enum item can be used whenever the value is needed:

```
assert u'Currrent color is {c} ({c.display})'.format(c=color) ==\
       u'Currrent color is #f00 (Vermelho)'
```

Even in dicts and sets, as it shares the same *hash()* from his value:

```
d = {
    HttpStatuses.OK.value: "using value",
    HttpStatuses.BAD_REQUEST: "using enum",
    401: "from original value",
}
assert d[HttpStatuses.OK] == "using value"
assert d[HttpStatuses.BAD_REQUEST.value] == "using enum"
assert d[HttpStatuses.OK] == d[HttpStatuses.OK.value]
assert d[HttpStatuses.UNAUTHORIZED] == d[401]
```

There’s also optimistic casting of inner types:

```
assert int(HttpStatuses.OK) == 200
assert float(HttpStatuses.OK) == 200.0
assert str(HttpStatuses.BAD_REQUEST) == "400"
```

Check membership:

```
assert HttpStatuses.OK in HttpStatuses
assert 200 in HttpStatuses
assert 999 not in HttpStatuses
```

JSON

If you want json serialization, you have at least two options:

1. Patch the default serializer.
2. Write a custom JSONEncoder.

ChoicesEnum comes with a handy patch funtion, you need to add this code to somewhere at the top of everything to automagically add json serialization capabilities:

```
from choicesenum.patches import patch_json
patch_json()
```

Note: Eventually `__json__` will be added to the stdlib, see https://bugs.python.org/issue27362

1.4 Django

1.4.1 Fields

Usage with the custom Django fields:

```
from django.db import models
from choicesenum.django.fields import EnumCharField

class ColorModel(models.Model):
    color = EnumCharField(
        max_length=100,
        enum=Colors,
        default=Colors.GREEN,
    )

instance = ColorModel()
assert instance.color ==  Colors.GREEN
assert instance.color.is_green is True
assert instance.color.value == Colors.GREEN.value == '#0f0'
assert instance.color.display == Colors.GREEN.display

instance.color = '#f00'
assert instance.color == '#f00'
assert instance.color.value == '#f00'
assert instance.color.display == 'Vermelho'
```

Is guaranteed that the field value is *always* a *ChoicesEnum* item. Pay attention that the field will only accept valid values for the `Enum` in use, so if your field allow *null*, your enum should also:

```
from django.db import models
from choicesenum import ChoicesEnum
from choicesenum.django.fields import EnumIntegerField

class UserStatus(ChoicesEnum):
    UNDEFINED = None
    PENDING = 1
    ACTIVE = 2
    INACTIVE = 3
    DELETED = 4


class User(models.Model):
    status = EnumIntegerField(enum=UserStatus, null=True, )

instance = User()
assert instance.status.is_undefined is True
assert instance.status.value is None
assert instance.status == UserStatus.UNDEFINED
assert instance.status.display == 'Undefined'
```

(continues on next page)

(continued from previous page)

```
# again...
instance.status = None
assert instance.status.is_undefined is True
```

1.5 Graphene

Usage with Graphene Enums:

```
UserStatusEnum = graphene.Enum.from_enum(UserStatus)
```

1.6 Schematics

Usage with Schematics Enums:

```
from schematics.models import Model as SchematicModel
from schematics.types import StringType, DateTimeType
from choicesenum import ChoicesEnum
from choicesenum.schematics.types import ChoicesEnumType

class HttpStatus(ChoicesEnum):
    OK = 200
    BAD_REQUEST = 400
    UNAUTHORIZED = 401
    FORBIDDEN = 403

class CustomSchematicModel(SchematicModel):
    name = StringType(required=True, max_length=255)
    created = DateTimeType(required=True, formats=('%d/%m/%Y', ''))
    http = ChoicesEnumType(HttpStatuses, required=True)
```

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install Choices Enum, run this command in your terminal:

```
$ pip install choicesenum
```

This is the preferred method to install Choices Enum, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Choices Enum can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/loggi/python-choicesenum
```

Or download the tarball:

```
$ curl  -OL https://github.com/loggi/python-choicesenum/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER

THREE

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:

3.1 Types of Contributions

3.1.1 Report Bugs

Report bugs at https://github.com/loggi/python-choicesenum/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.

3.1.2 Fix Bugs

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

3.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

3.1.4 Write Documentation

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

3.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/loggi/python-choicesenum/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 :)

3.2 Get Started!

Ready to contribute? Here's how to set up *choicesenum* for local development.

1. Fork the *choicesenum* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/choicesenum.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 choicesenum
$ cd choicesenum/
$ 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 choicesenum tests
$ 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.

3.3 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 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/loggi/python-choicesenum/pull_requests and make sure that the tests pass for all supported Python versions.

3.4 Tips

To run a subset of tests:

```
$ py.test tests.test_choicesenum
```

CHAPTER

FOUR

CREDITS

4.1 Development Lead

- Fernando Macedo <fgmacedo@gmail.com>

4.2 Contributors

- Kristian Klette <klette@klette.us>
- Gabriela Surita <gabsurita@gmail.com>
- Leandro Gomes <leandrogs99@gmail.com>
- Tomas Fagerbekk <tomas.a.fagerbekk@gmail.com>
- Lefteris Karapetsas <lefteris@refu.co>
- Maiky Guanaes <maiky.guanaes@gmail.com>

4.3 Tools

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER

FIVE

HISTORY

5.1 0.7.0 (2020-08-02)

- Add support Django 3.0 (tks @klette).
- Drop support for Python 3.4.
- Fix issue when using Django EnumIntegerField on Admin.

5.2 0.6.0 (2019-09-05)

- Adding schematics contrib type ChoicesEnumType.
- Drop support for Django 1.6, 1.7, 1.8.

5.3 0.5.3 (2019-02-06)

- Fix Django migrations with default values for Django 1.7+.

5.4 0.5.2 (2019-01-18)

- Optimize member check and dynamic creation of *is_<name>* properties.

5.5 0.5.1 (2019-01-04)

- Fix readme RST (requires new Pypi upload).

5.6 0.5.0 (2019-01-04)

- Membership test (item in Enum) returning valid results for primitive values.
- New dict-like *.get* method able to return a default value (thanks @leandrogs).
- Django: Support Postgres array functions and queries (thanks @tomfa).
- Django: Support for deferring an enum field using *queryset.defer()* (thanks @noamkush).
- Django: 2.1 support.

5.7 0.4.0 (2018-07-13)

- Optimistic casting of inner types (thanks @gabisurita).
- Optional stdlib patch to automagic json serialization support.
- Add Python3.7 to the test matrix.

5.8 0.3.0 (2018-06-22)

- Official Django 2.0 support (0.2.2 just works fine too).
- `ChoicesEnum` sharing the same hash() as his value. Can be used to retrieve/restore items in dicts (*d[enum] == d[enum.value]*).

5.9 0.2.2 (2017-12-01)

- Fix: Support queries through *select_related* with no *None* value defined (thanks @klette).

5.10 0.2.1 (2017-09-30)

- Fix South migrations for Django 1.6.

5.11 0.2.0 (2017-09-11)

- `ChoicesEnum` items are comparable by their values (==, !=, >, >=, <, <=) (thanks @jodal).
- +``ChoicesEnum.values``: Returns all the Enum's raw values (eq: `[x.value for x in Enum]`).

5.12 0.1.7 (2017-09-10)

- Fix: `ChoicesEnum` is now hashable (thanks @jodal).

5.13 0.1.6 (2017-09-08)

- Fix: Proxy `__len__` calls to the inner enum value.

5.14 0.1.5 (2017-09-05)

- +ChoicesEnum.description: Alias for *label*, allow enum descriptors to be used by Graphene.

5.15 0.1.4 (2017-08-28)

- Fix South migrations for Django 1.6.
- `ChoicesEnum` repr can be used to reconstruct an instance (`item == eval(repr(item))`).

5.16 0.1.3 (2017-08-28)

- Fix sdist not including sub-modules (django contrib).

5.17 0.1.2 (2017-08-27)

- README fixes and improvements.

5.18 0.1.0 (2017-08-27)

- First release on PyPI.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search