Choices in Django 3.0

History, design and implementation
A case study in adding a feature

2020 djangocon.eu porto

About me

- Shai Berger
- Django Security Team member
- Django Core, 2013 – 2018
- Python since 1998 (1.5.2)
- Professional Programmer since 1990
- Consultant and Free Software activist in Israel

קפלן ייעוץ
קוד פתוח

המקור
עמותה ישראלית לתוכנה
חופשית ולקוד מקור פתוח (ע"ר)

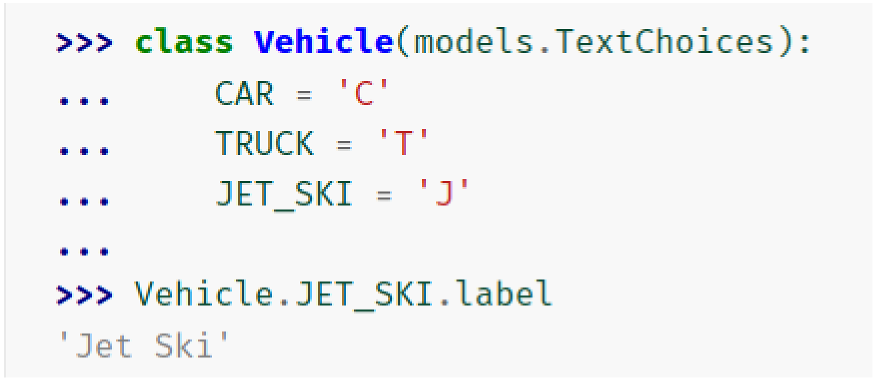
What are Django Choices

```python
class Student(models.Model):
    FRESHMAN = 'FR'
    SOPHOMORE = 'SO'
    JUNIOR = 'JR'
    SENIOR = 'SR'
    GRADUATE = 'GR'
    YEAR_IN_SCHOOL_CHOICES = [
        (FRESHMAN, 'Freshman'),
        (SOPHOMORE, 'Sophomore'),
        (JUNIOR, 'Junior'),
        (SENIOR, 'Senior'),
        (GRADUATE, 'Graduate'),
    ]
    year_in_school = models.CharField(
        max_length=2,
        choices=YEAR_IN_SCHOOL_CHOICES,
        default=FRESHMAN,
    )
```

```python
class Student(models.Model):

    class YearInSchool(models.TextChoices):
        FRESHMAN = 'FR', _('Freshman')
        SOPHOMORE = 'SO', _('Sophomore')
        JUNIOR = 'JR', _('Junior')
        SENIOR = 'SR', _('Senior')
        GRADUATE = 'GR', _('Graduate')

    year_in_school = models.CharField(
        max_length=2,
        choices=YearInSchool.choices,
        default=YearInSchool.FRESHMAN,
    )
```

What's more in Choices

- Automatic labels



```
>>> class Vehicle(models.TextChoices):
...     CAR = 'C'
...     TRUCK = 'T'
...     JET_SKI = 'J'
...
>>> Vehicle.JET_SKI.label
'Jet Ski'
```

- stdlib Enum features
  - Lookup with `[name]` and `(value)`
  - `.name` and `.value` (and `.label`) properties

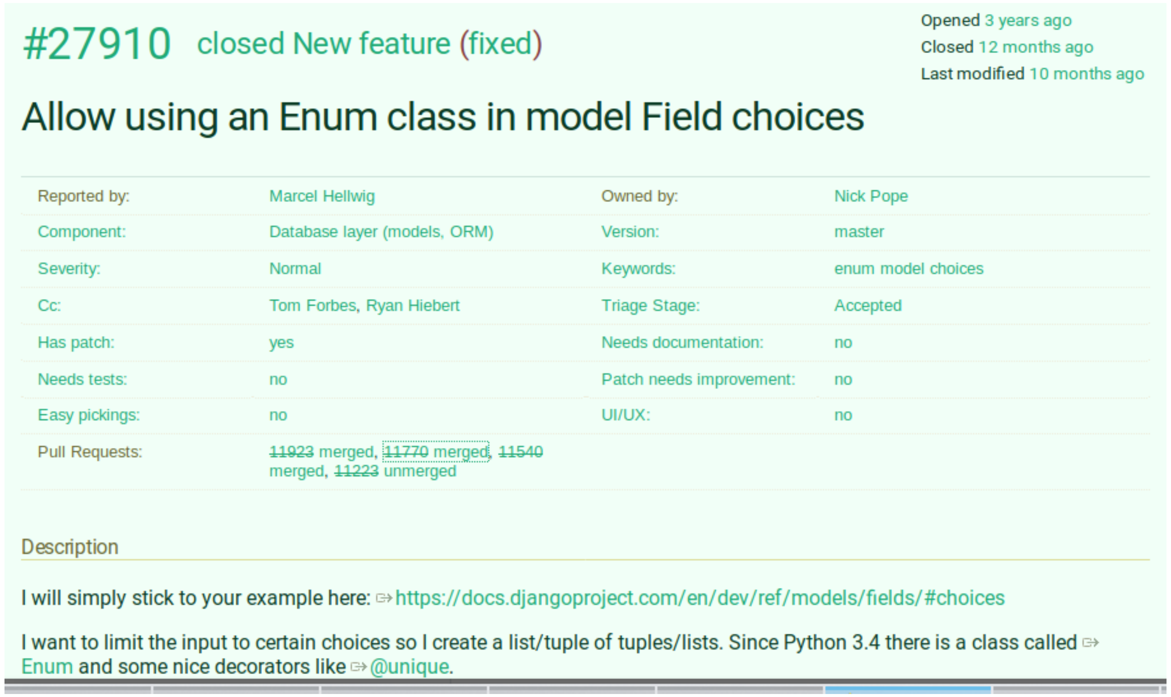
History

Grayscale Photography of Statues by Pixabay
https://www.pexels.com/photo/building-castle-figures-facade-36006/

Prehistory

- People built their own classes
- Important early example: Tom Forbes' django-choice-object (7/2013)
  - Not using stdlib enum
  - Predates Python 3.4
    (which added enum to stdlib, 3/2014)

History starts, 3/2017



#27910 closed New feature (fixed)

Opened 3 years ago
Closed 12 months ago
Last modified 10 months ago

Allow using an Enum class in model Field choices

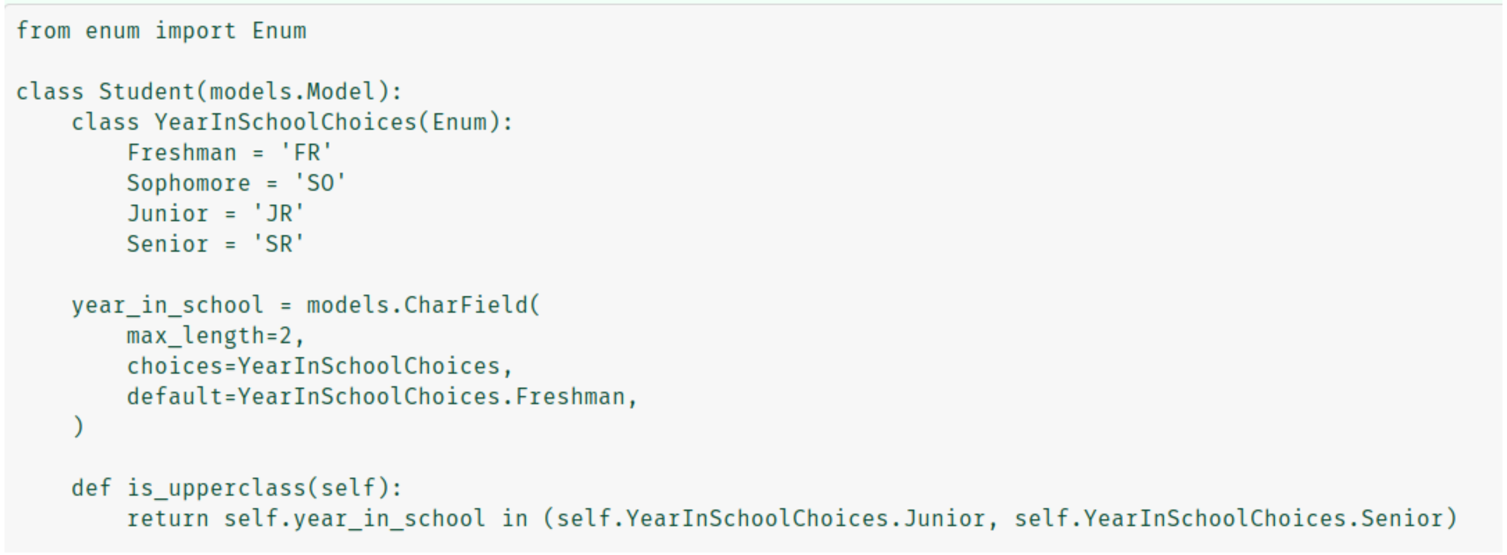
Reported by:	Marcel Hellwig	Owned by:	Nick Pope
Component:	Database layer (models, ORM)	Version:	master
Severity:	Normal	Keywords:	enum model choices
Cc:	Tom Forbes, Ryan Hiebert	Triage Stage:	Accepted
Has patch:	yes	Needs documentation:	no
Needs tests:	no	Patch needs improvement:	no
Easy pickings:	no	UI/UX:	no
Pull Requests:	~~11923~~ merged, ~~11770~~ merged, ~~11540~~ merged, ~~11223~~ unmerged		

Description

I will simply stick to your example here: https://docs.djangoproject.com/en/dev/ref/models/fields/#choices

I want to limit the input to certain choices so I create a list/tuple of tuples/lists. Since Python 3.4 there is a class called Enum and some nice decorators like @unique.

History starts, 3/2017



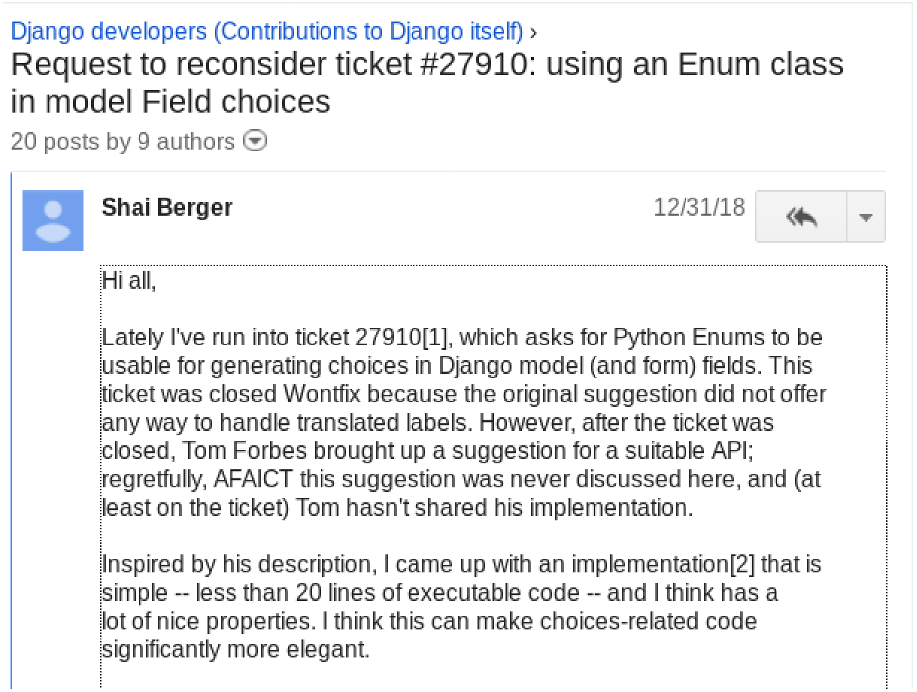
```python
from enum import Enum

class Student(models.Model):
    class YearInSchoolChoices(Enum):
        Freshman = 'FR'
        Sophomore = 'SO'
        Junior = 'JR'
        Senior = 'SR'

    year_in_school = models.CharField(
        max_length=2,
        choices=YearInSchoolChoices,
        default=YearInSchoolChoices.Freshman,
    )

    def is_upperclass(self):
        return self.year_in_school in (self.YearInSchoolChoices.Junior, self.YearInSchoolChoices.Senior)
```

Ticket 27910, take 1

- Using enum, with label from member name
- Closed “Wontfix” over difficulties with translation
- Five months later, Tom Forbes suggests his approach as an API
  - ...but does not link
  - The ticket is already closed, it mostly goes unnoticed

A second take, 12/2018

- A new look at the ticket leads to working POC code and a request to reopen
  - Reaction mostly positive
  - But some objections


Django developers (Contributions to Django itself) ›

Request to reconsider ticket #27910: using an Enum class in model Field choices

20 posts by 9 authors

Shai Berger 12/31/18

Hi all,

Lately I've run into ticket 27910[1], which asks for Python Enums to be usable for generating choices in Django model (and form) fields. This ticket was closed Wontfix because the original suggestion did not offer any way to handle translated labels. However, after the ticket was closed, Tom Forbes brought up a suggestion for a suitable API; regretfully, AFAICT this suggestion was never discussed here, and (at least on the ticket) Tom hasn't shared his implementation.

Inspired by his description, I came up with an implementation[2] that is simple -- less than 20 lines of executable code -- and I think has a lot of nice properties. I think this can make choices-related code significantly more elegant.

Progress at DjangoConEU

- Some discussions with Technical Team members and other devs
- Ticket re-opened
- POC fleshed out to an initial PR during sprints
  - Enums still considered key, so the base class is named ChoiceEnum

Others get involved

- PR gets reviews
- Following a call for help, Mariusz Felisiak and Nick Pope step in; Nick takes over in 7/2019
- More issues, code and documentation polishing
- Class name changed to “Choices”
  - with subclasses “IntegerChoices” and “TextChoices”
  - to de-emphasize the connection to Enum

Success!

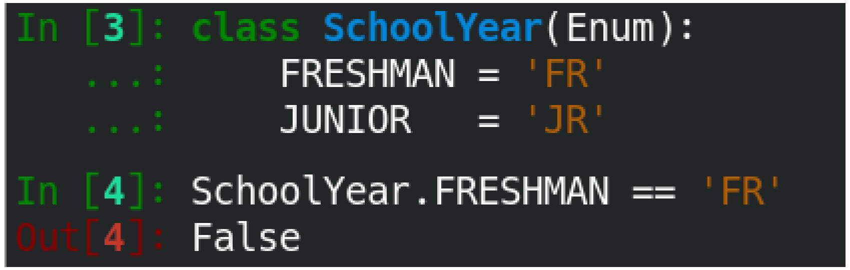
- Choices declared a major feature, and committed into Django's master branch on September 4th 2019
- Django 3.0 alpha1 released on September 10th
- One major bug fixed by Carlton Gibson before the final 3.0 release
- Django 3.0 released with Choices on December 2nd

Design

Photo by Christina Morillo from Pexels (Twitter: @wocintechchat)
https://www.pexels.com/photo/woman-wearing-gray-blazer-writing-on-dry-erase-board-1181534/

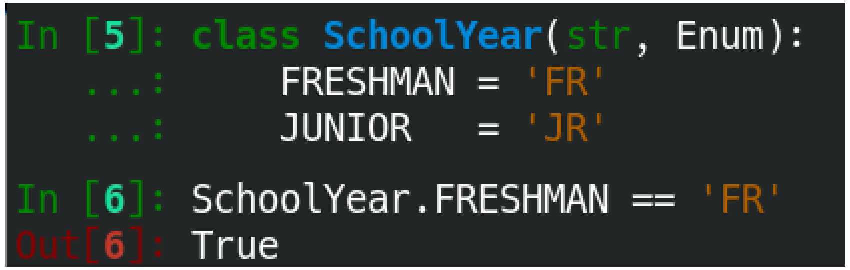
Use stdlib enums?

- Enums can be surprising:



```
In [3]: class SchoolYear(Enum):
   ...:     FRESHMAN = 'FR'
   ...:     JUNIOR   = 'JR'

In [4]: SchoolYear.FRESHMAN == 'FR'
Out[4]: False
```

- But you just need to be prepared:



```
In [5]: class SchoolYear(str, Enum):
   ...:     FRESHMAN = 'FR'
   ...:     JUNIOR   = 'JR'

In [6]: SchoolYear.FRESHMAN == 'FR'
Out[6]: True
```

Use stdlib enums.

- Enums don't always do what you expect
  - Our use case requires base types, generally discouraged
  - We had to change some details
- But they are familiar
  - We get some API (design and implementation) “for free”
- “[W]e are providing a `Choices` type that is enum-like and enum-backed, but has to make its own rules.”
  - *Nick Pope, on PR 11964*

The empty choice

- With list-of-pairs, specified by value `None`
  - Enum-with-base-type member must have value of type
- Options:
  - Designate specific value to mark empty
  - Make and use `NullableInt` instead of `int` etc
  - Preprocess `None` away
  - Use special name `__empty__` with label only

What should `__str__()` return?

- Options for TextChoices (`str` values):
  - Return label, the human-presentation version
  - Return value, the str we started with
    - Return `str(self.value)` for all Choices types
  - Raise exception, because both above are valid
  - Delete `TextChoices` to avoid the issue
    - (stdlib defines `IntEnum` but not `StrEnum`)

```
str(YIS.SENIOR)

"Senior"

"SR"
```

Implementation

Close-up of Telephone Booth by Pixabay
https://www.pexels.com/photo/close-up-of-telephone-booth-257736/

The Challenge

- Inherit `enums.Enum`
- … but make it "not notice" the label
- … while storing the label and accessing it

Subclass `enums.Enum`

- We want to modify the subclass creation process
- So we really need to inherit the metaclass

```python
class ChoicesMeta(enum.EnumMeta):
    """A metaclass for creating a enum choices."""

    def __new__(metacls, classname, bases, classdict):
```

```python
class Choices(enum.Enum, metaclass=ChoicesMeta):
    """Class for creating enumerated choices."""
```

Hide labels from parent

- Check value
- If fits pattern, remove and keep label
- Else make label
- Collect labels separately

```python
def __new__(metacls, classname, bases, classdict):
    labels = []
    for key in classdict._member_names:
        value = classdict[key]
        if (
            isinstance(value, (list, tuple)) and
            len(value) > 1 and
            isinstance(value[-1], (Promise, str))
        ):
            *value, label = value
            value = tuple(value)
        else:
            label = key.replace('_', ' ').title()
        labels.append(label)
        # Use dict.__setitem__() to suppress defenses against double
        # assignment in enum's classdict.
        dict.__setitem__(classdict, key, value)
    cls = super().__new__(metacls, classname, bases, classdict)
```

Make labels accessible

- `labels` is in order
- Enum gives us values in order via `cls._value2member_map_`
  - Set `label` attribute on each instance, or
  - Build `cls._value2label_map_` and access it in a property

2020 djangocon.eu porto

`label` property implementation

- We build the map in one line:

```
cls._value2label_map_ = dict(zip(cls._value2member_map_, labels))
```

- Define property (suggestion)

```
cls.label = property(cls._values2label_map_.get)
```

- Final version was less magical

```
cls.label = property(lambda self: cls._value2label_map_.get(self.value))
```

2020 djangocon.eu porto

End Matter

Dock Under Golden Hour by Pixabay
https://www.pexels.com/photo/beach-bench-boardwalk-bridge-276259/

Enhancements?

- Grouping in the control (supported with list-of-tuples)
- Categories (sets of options) as present in django-choice-object

Thanks!

- shai@platonix.com or shai@kaplanopensource.co.il
- Twitter: @shaib_il
- https://github.com/shaib and https://gitlab.com/shaib
- And a happy 5781 to you all!

All non-credited images are from Django project assets
(except for the blue elephant icon)