py-moneyed

Release 3.0

Kai Wu

Nov 27, 2022

CONTENTS:

These are the docs for py-moneyed 3.0. Check the *Change log* for significant changes.

CHAPTER

ONE

INSTALLATION

py-moneyed can be installed with pip:

```
pip install py-moneyed
```

CHAPTER TWO

USAGE

The Money class is instantiated with:

- An amount which can be of type int, string, float, or Decimal. It will be converted to a Decimal internally. Therefore, it is best to avoid float objects, since they do not convert losslessly to Decimal.
- A currency, as a Currency object, or as a string which is a three-capital-letters ISO currency code (e.g. 'USD', 'EUR' etc), which will be converted to a Currency object.

For example,

```
from moneyed import Money
sale_price_today = Money(amount='99.99', currency='USD')
```

You then use Money instances as a normal number. The Money class provides operators with type checking, matching currency checking, and sensible dimensional behavior, e.g. you cannot multiply two Money instances, nor can you add a Money instance to a non-Money number; dividing a Money instance by another results in a Decimal value, etc.

The Currency class is also provided. All ISO 4217 currencies are available by importing from the moneyed module by their 3-letter code, as pre-built Currency objects.

You can also pass in the arguments to Money as positional arguments. So you can also write:

```
>>> from moneyed import Money, USD
>>> price = Money('19.50', USD)
>>> price
Money('19.50', 'USD')

>>> price.amount
Decimal('19.50')

>>> price.currency
USD

>>> price.currency.code
'USD'
```

If you want to get the amount in sub units (ISO 4127 compatible) you can do:

```
>>> from moneyed import Money, USD
>>> price = Money('19.50', USD)
>>> price.get_amount_in_sub_unit()
1950

```

(continues on next page)

(continued from previous page)

```
>>> price = Money('123.456', USD)
>>> price.get_amount_in_sub_unit()
12345
```

Currency instances have a `zero` property for convenience. It returns a cached `Money` instance of the currency. This can be helpful for instance when summing up a list of money instances using the builtin `sum()`.

```
>>> from moneyed import Money, USD
>>> currency = USD
>>> items = (Money('19.99', currency), Money('25.00', currency))

>>> sum(items, currency.zero)
Money('44.99', 'USD')

>>> sum((), currency.zero)
Money('0', 'USD')
```

2.1 Search by Country Code

In order to find the ISO code associated with a country, the function `get_currencies_of_country()` can be used. This function takes the ISO country code (case insensitive) as the argument and returns the associated currency object(s) in a list. If a country with the given name is not found the function returns an empty list. The code below demonstrates this:

```
>>> from moneyed import get_currencies_of_country
>>> get_currencies_of_country("IN")
[INR]
>>> get_currencies_of_country("BO")
[BOB, BOV]
>>> get_currencies_of_country("XX")
[]
```

2.2 Get country names

`Currency.country_codes` returns a list of ISO 3166 country codes. You can convert these to names using the function `get_country_name`, which must be passed a ISO 2-letter code and a locale code:

```
>>> from moneyed import ZMW, get_country_name
>>> ZMW.country_codes
['ZM']
>>> get_country_name('ZM', 'en')
'Zambia'
```

2.3 List all currencies

You can get all installed currencies as below:

```
>>> from moneyed import list_all_currencies
>>> list_all_currencies()
[ADP, AED, AFA, ...]
```

The result is a list of `Currency` objects, sorted by ISO code.

CHAPTER THREE

FORMATTING

You can print Money object as follows:

```
>>> from moneyed.l10n import format_money
>>> format_money(Money(10, 'USD'), locale='en_US')
'$10.00'
```

Note that you need to specify `locale` or you will get the system default, which will probably not be what you want. For this reason, it is recommended to always provide the `locale` argument, and you may well want to add your own wrappers around this function to supply your project specific defaults.

This function is a thin wrapper around babel.numbers.format_currency. See those docs for other arguments that can be specified to control the formatting of the number. By default, Babel will apply definitions of how to format currencies that have been derived from the large CLDR database.

If you do `str()` on a `Money` object, you will get the same behaviour as `format_money()`, but with no options supplied, so you will get the system default locale.

CHAPTER

FOUR

CONTRIBUTING TO PY-MONEYED

If you would like to contribute to py-moneyed, the recommended workflow is:

1. First raise an issue on GitHub for any proposal or idea that might be controversial and get feedback from the maintainers. If you have something that is an obvious bug (a typo in the docs, for example), you can skip this step.
2. Fork the project and checkout your fork onto your development machine.
3. Create a virtualenv of some kind for development (venv, virtualenv or virtualenvwrapper) and install py-moneyed into it:

   ```
   python setup.py develop
   ```

4. Optional, but highly recommended to save time later - install pre-commit hooks:

   ```
   pip install pre-commit
   pre-commit install
   ```

5. Create a git branch for your changes, starting from `master`
6. Fix the bug or implement your changes, being sure to:
   1. Add tests and docs
   2. Run the test suite (below)
7. Push your changes to your GitHub repo and submit a pull request.

4.1 Testing

To run the test suite, first install tox (into your virtualenv):

```
pip install tox
```

Run the tests using tox:

```
tox -e py310
```

You can run the test suite on all supported environments using tox (recommended). If you do not have all versions of Python that are used in testing, you can use pyenv to install them, and you may benefit from the additional plugin pyenv-implict.

The py-moneyed package is tested against Python 3.7 - 3.11 and PyPy 3.

CHAPTER FIVE

CHANGE LOG

Significant or incompatible changes listed here.

5.1 Unreleased - TBA

5.2 3.0 (2022-11-27)

- Added `SLE` & `VED` currencies.
- Removed support for Python 3.6.
- Added support for Python 3.10 & 3.11.

5.3 2.0 (2021-05-26)

- Dropped support for Python 2.7 and 3.5 and PyPy 2.
- Added pyupgrade pre-commit hook.
- Added black pre-commit hook and reformatted codebase.
- Updated pre-commit hooks.
- Replaced custom flake8, isort and check-manifest Github Action jobs with a generic pre-commit job.
- Dropped the `moneyed.localization` module that was deprecated and announced for removal in 1.0.
- Added type hints along with a mypy pre-commit hook.
- Added action for building and publishing releases, along with the check-github-workflows pre-commit hook for validating Github Action workflow files.
- Removed undocumented `DEFAULT_CURRENCY` and `DEFAULT_CURRENCY_CODE` constants, and change to make instantiating `Money` without providing a currency a type error. This used to result in an object with a made-up `"XYZ"` currency, which could lead to surprising behaviors and bugs.
- Added `zero` property to `Currency` to conveniently access the zero value of a given currency.
- Moved to use setuptool's declarative packaging config and PEP 517 isolated builds.
- Removed requirements files and instead specified test requirements using extras.

5.4 1.2 (2021-02-23)

- `Money.__add__` returns `NotImplemented` instead of raising an exception when another operand has unsupported type.

5.5 1.1 (2021-01-15)

- Changed the `numeric` attribute values to `None` for currencies that don't have assigned ISO numeric codes: `IMP`, `TVD`, `XFO`, `XFU`.
- Restored the previous definition for the `XXX` currency, including its `name` and `countries` attributes.
- Fixed `get_currency` returning obsolete currencies.

5.6 1.0 (2021-01-09)

- Dropped official support for Python 2.6, 3.2, 3.3, 3.4 (mainly because our test tools don't support them any more).
- Added support for getting amount in sub units (fixed point)
- Format `Money` instances using CLDR and Babel. This is a large change with lots of parts. Many thanks to @pooyamb for all the hard work that went into this and other related changes.
  - Added new `moneyed.l10n` module, containing a new `format_money` function. This is a very thin wrapper around babel.numbers.format_currency and has all the same options. This allows us to get the official CLDR formats for currencies, in all the different locales.

    See docs in README.

    Note especially that you need to specify `locale` (e.g. `locale="en_US"`), or you will get the `LC_NUMERIC` default.
  - Deprecated the `format_money` function in `moneyed.localization`. There is no immediate plan to remove, but it should not be relied on. Also, this function relies on our own manually entered data for formatting of currencies in different locales. This data is very incomplete and will not be updated any more.

    So you need to use `moneyed.l10n.format_money` instead now.

    If you were relying on the `decimal_places` argument to the old function, there is no exact equivalent in the new `format_money` function, but see the `decimal_quantization` option (documented in babel.numbers.format_currency)
  - `Money.__str__` (`Money.__unicode__` on Python 2) now uses new `format_money` with the default locale `LC_NUMERIC`, which can produce different results from the old function. Use the new `format_money` to control output.
  - On Python 2, `Money.__str__` (bytestring) output has changed to be more basic. You should use the new `format_money` function to control output.
- Get currency names from Babel data. Several changes, including:
  - For all built-in currencies, `Currency.name` now comes from Babel ("en_US" locale). This means there have been various corrections to currency names.

    If you pass a non-None `name` to the `Currency` constructor, you can still specify any name you want.
  - `Currency.get_name(locale)` has been added.

- Get currency 'countries' from Babel data. Several changes, including:
  - `Currency.countries` now sources from Babel, so some names may be different.
  - `Currency.country_codes` has been added.
  - `Currency.countries` is deprecated, because it is not the most useful form for the data (e.g. upper cased strings, and names in US English only). It is recommended to use `Currency.country_codes` and convert to names using `get_country_name`.
- Changed the repr of `Money` so that `eval(repr(money_object) == money_object` (at least in some environments, and most of the typical ones). See Python docs on __repr__ for rationale. Thanks @davidtvs. This could be backwards incompatible if you were relying on the old output of `repr()`.
- Added `list_all_currencies()` utility function.

5.7 0.8 (2018-11-19)

- `Money.round([ndigits])` added. Uses `decimal.ROUND_HALF_EVEN` by default, but this can be overridden by setting `rounding` in the `decimal` context before calling `Money.round()`.
- Various fixes/additions for different locales
- Division support on Python 2
- DEFAULT locale is now used as a fallback to return a currency symbol if your chosen locale has no symbol set for that currency, rather than just returning the currency code.

5.8 0.7 (2017-05-08)

- `Money.__str__` changed under Python 2 to use only ASCII characters. This means that currency codes, rather than symbols, are used.
- Lots of additional locales supported out of the box.
- Python 3.5 supported
- Fixed #70 - format_money error when the locale is not in the formatting definitions: the default is not used.
- Various other bug fixes

5.9 0.6 and earlier

- See VCS logs.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search