# tri.token Documentation

***Release 1.1.0***

**[u'Anders Hovm6ller', u'Johan L']**

**Sep 26, 2018**

# Contents

CHAPTER 1

# tri.token

tri.token provides enriched enum functionality. tri.token enum structures are declared using:

- TokenAttribute: overridable attribute definitions with support for dynamic values.
- Token: holds TokenAttributes objects.
- TokenContainer: holds Token objects.

In other words: a Token is an enum which has TokenInstance members. Token instances are declared within TokenContainer(s).

## 1.1 Basic usage

```
from tri.token import Token, TokenAttribute, TokenContainer, PRESENT


class Taste(Token):
    name = TokenAttribute()
    display_name = TokenAttribute(value=lambda **kwargs: kwargs['name'].upper() + '!!
↪')
    opinion = TokenAttribute()


class Tastes(TokenContainer):
    vanilla = Taste()
    pecan_nut = Taste(display_name="pecan nutz", opinion="Tasty")


# A TokenContainer is a collection of Token objects.
assert Tastes.vanilla in Tastes

# The order of Token objects in a TokenContainer is by order of declaration.
assert list(Tastes) == [Tastes.vanilla, Tastes.pecan_nut]
```

(continues on next page)

(continued from previous page)

```
assert list(Tastes) != [Tastes.pecan_nut, Tastes.vanilla]

# Magic for 'name' TokenAttribute. It is set automatically from the token declaration␣
↪within it's container.
assert Tastes.vanilla.name == "vanilla"

# A TokenAttribute will have a None value if not set during Token instantiation.
assert Tastes.vanilla.opinion is None

# A TokenAttribute can have a dynamic value, derived from the invocation to the␣
↪callable
# set as 'value' in the TokenAttribute definition
# (see declaration of 'display_name' TokenAttribute further up in the code).

# The real value of the token attribute will be the return value of an invocation to␣
↪said callable.
# The invocation will receive the values of all other token attributes passed as␣
↪keyword arguments.
assert Tastes.vanilla.display_name == "VANILLA!!"

# TokenAttribute dynamic value behavior is overridden/not used if value is set␣
↪explicitly during Token instantiation.
assert Tastes.pecan_nut.display_name == "pecan nutz"

# A TokenContainer can be rendered as csv, excel, rst etc
assert """\
+--------------+---------+
| display_name | opinion |
+==============+=========+
|  VANILLA!!   |         |
+--------------+---------+
|  pecan nutz  |  Tasty  |
+--------------+---------+\
""" == Tastes.to_rst(['display_name', 'opinion'])
```

## 1.2 Optional token attributes

```
# A TokenAttribute may be declared as having optional dynamic values.
# That is, we want these dynamic attributes to be evaluated sometimes, but not always.
# In the example below, we want some superheroes to have homes, but not others.

SUPERHERO_HOMES = {'superman': 'Fortress of Solitude',
                   'batman': 'Batcave'}


class Superhero(Token):
    name = TokenAttribute()
    home = TokenAttribute(optional_value=lambda name, **_: SUPERHERO_HOMES[name])


# The PRESENT special value is used during Token instantiation to decide what
# optional token attributes should be evaluated.
class Superheroes(TokenContainer):
    batman = Superhero(home=PRESENT)
```

(continues on next page)

(continued from previous page)

```
    hawkman = Superhero()
    wonder_woman = Superhero(home='Themyscira')

# Batman has a home, but poor Hawkman does not.
assert Superheroes.batman.home == 'Batcave'
assert Superheroes.hawkman.home is None

# Just as with dynamic attributes, the logic for TokenAttribute optional dynamic␣
↪values is overridden
if value is set explicitly during Token instantiation.
assert Superheroes.wonder_woman.home = 'Themyscira'

# As a shortcut, PRESENT for specific optional token attributes may be assigned to
# variables, and used in declarations, for enhanced readability.
# This is useful when one has tokens with many attributes declared using dynamic␣
↪values,
# but we don't want all of them to be evaluated in all tokens.
home = PRESENT('home')

class Superheroes(TokenContainer):
    batman = Superhero(home)
    hawkman = Superhero()

# Again, Batman has a home, but poor Hawkman does not.
assert Superheroes.batman.home == 'Batcave'
assert Superheroes.hawkman.home is None
```

## 1.3 TokenAttribute inheritance

```
class FooToken(Token):
    foo = TokenAttribute(value=lambda **kwargs: 'foo_value')

class BarToken(Token):
    bar = TokenAttribute()

class FieToken(FooToken, BarToken):
    fie = TokenAttribute()

class FooBarFieTokenContainer(TokenContainer):
    t = FieToken(fie=3)

assert dict(FooBarFieTokenContainer.t) == {'foo': 'foo_value', 'bar': None, 'name': 't
↪', 'fie': 3}
```

## 1.4 TokenAttribute container inheritance

```
class MyToken(Token):

    name = TokenAttribute()
    stuff = TokenAttribute()
```

(continues on next page)

(continued from previous page)

```
class MyTokens(TokenContainer):

    foo = MyToken(stuff='Hello')
    bar = MyToken(stuff='World')

assert MyTokens.foo in MyTokens

class MoreTokens(MyTokens):
    boink = MyToken(stuff='Other Stuff')

assert MyTokens.foo in MoreTokens

assert list(MoreTokens) == [MyTokens.foo, MyTokens.bar, MoreTokens.boink]
assert MoreTokens.foo is MyTokens.foo
```

For more tri.token examples, please have a look at the contents of tests/test_tokens.py in the installation directory.

## 1.5 Running tests

You need tox installed then just *make test*.

## 1.6 License

BSD

## 1.7 Documentation

http://tritoken.readthedocs.org.

CHAPTER 2

Contents:

## 2.1 Installation

At the command line:

```
$ pip install tri.token
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv tri.token
$ pip install tri.token
```

## 2.2 Usage

You need to add *tri.token* to installed apps or copy the templates to your own template directory.

## 2.3 API documentation

**class** tri.token.**Token**(**args*, ***kwargs*)

**__init__**(**args*, ***kwargs*)
x.__init__(...) initializes x; see help(type(x)) for signature

**__lt__**(*other*)
x.__lt__(y) <==> x<y

**__repr__**() <==> *repr(x)*

**__str__**() <==> *str(x)*

**__weakref__**
list of weak references to the object (if defined)

**classmethod get_declared**(*parameter='members'*)
Get the `OrderedDict` value of the parameter collected by the `@declarative` class decorator. This is the same value that would be submitted to the `__init__` invocation in the `members` argument (or another name if overridden by the `parameter` specification) @type cls: class @type parameter: str @return OrderedDict

**classmethod set_declared**(*value*, *parameter='members'*)
@type cls: class @type value: OrderedDict @type parameter: str

**class** `tri.token.`**TokenAttribute**(**args*, ***kwargs*)

**__init__** (**args*, ***kwargs*)
x.__init__(...) initializes x; see help(type(x)) for signature

**__lt__** (*other*)
x.__lt__(y) <==> x<y

**__weakref__**
list of weak references to the object (if defined)

`tri.token.`**with_metaclass**(*meta*, **bases*)
Create a base class with a metaclass.

## 2.4 History

### 2.4.1 1.1.0 (2018-09-26)

- Added requirement to set *__override__=True* parameter for token that shadow token from base class.

#### 1.0.1 (2017-04-05)

- Changed metaclass handling to make PyCharm understand it

#### 1.0.0 (2016-09-27)

- First released version on github
- Added documentation
- Cleanup of build machinery

#### 0.6.0 (2016-04-11)

- Added documentation generation sort customisation
- Added python 3 support

#### 0.5.0 (2016-02-01)

- Move dependency tri.cache.memoize from requirements.txt to test_requirements.txt. It is only used for regression testing.

## 2.5 Credits

- Johan Lübcke <johan.lubcke@trioptima.com>
- Anders Hovmöller <anders.hovmoller@trioptima.com>

## 2.6 Contributing

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

Issues, feature requests, etc are handled on github.

CHAPTER 3

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## t

# Index

## Symbols



## G



## S



## T



## W