pyUnitTypes Documentation

Release latest

Jul 26, 2019

Contents

1 Contents **3**
- 1.1 Overview . . . 3
- 1.2 Base Units . . . 5
- 1.3 Composite Units . . . 7
- 1.4 Creating your own UnitTypes . . . 7

This is a python package to work with different physical units as types and pythons type annotations.

Source code

https://github.com/flxdot/pyUnitTypes

Documentation

https://pyunittypes.readthedocs.io/en/latest/

CHAPTER 1

Contents

1.1 Overview

This package is designed to easily work with physical units. The main usage is to display different units and unit systems in FrontEnds. Typical use case would be: Your application make it's calculation in SI units and then you'll need to deploy your application in the US. So instead of display temperatures in °C you'll now have to display it in °F.

Warning: It should never be used to make precise calculations, since the accuracy is limited to 3-4 digits at the moment. Especially when converting SI units to imperial units.

1.1.1 Converting your first value

The following example illustrates how to convert from one unit to another in the same type of physical unit (e.g. lengths):

```
from pyUnitTypes.length import Meter, KiloMeter, MilliMeter, Inch

# define the value as KiloMeter
a_long_distance = KiloMeter(2.5)
a_short_distance = Inch(1)

# print the value as Meter
print(Meter(a_long_distance))
print(MilliMeter(a_short_distance))
```

The output will look like this:

1.1.2 Calculating with units

Calculation with unit values is as easy as normal calculation. Sofar following operators are implemented:

Arithmetic Operators

The following mathematical operators can be used to calculate with the units.

- **add (+)**: Works as within a UnitType package. Raises a TypeError if Units from different modules are used.

```
from pyUnitTypes.length import Meter, KiloMeter

two_kilometer = Meter(1000) + KiloMeter(1)
```

- **sub (-)**: Works as within a UnitType package. Raises a TypeError if Units from different modules are used.

```
from pyUnitTypes.length import Meter, KiloMeter

half_kilometer = KiloMeter(1) - Meter(500)
```

- **mul (*)**: Works when multiplied with `float` or `int`.

  Raises `pyUnitTypes.basics.UnknownUnitMultiplicationError` when multiplication of the two units has not been implemented

  Raises `TypeError` if multiplied with objects which are not inherited from `pUnitTypes.basics.BaseUnit`.

```
from pyUnitTypes.length import Meter, KiloMeter

five_kilometer = KiloMeter(2) * 2.5
```

- **div (*)**: Works when divided by `float` or `int`.

  Raises `pyUnitTypes.basics.UnknownUnitDivisionError` when division of the two units has not been implemented or if division of `float` or `int` by the unit is not implemented.

  Raises `TypeError` if multiplied with objects which are not inherited from `pUnitTypes.basics.BaseUnit`.

```
from pyUnitTypes.length import Meter, KiloMeter

four_kilometer = KiloMeter(10) / 2.5
```

Comparison Operators

Any pyUnitTypes object can be compared to another object from the same module. Comparing to a object of a different package or any other object will raise as `TypeError`.

```
from pyUnitTypes.length import Meter, KiloMeter

is_eq = Meter(1000) == KiloMeter(1)
is_ne = Meter(0) != KiloMeter(1)
is_lt = Meter(1) < KiloMeter(1)
is_gt = Meter(2000) > KiloMeter(1)
is_le = Meter(1000) >= KiloMeter(1)
is_ge = Meter(1000) <= KiloMeter(1)
```

Other numeric functionality

Besides the four basic arithmetic operators several other mathematical operations are supported:

- `round()`
- `math.ceil()`
- `math.floor()`
- `__neg__`: `Meter(-1)` is equal to `-Meter(1)`
- `__pos__`: `Meter(+1)` is equal to `+Meter(1)`

All pyUnitType objects can be converted to `int` or `float`.

1.2 Base Units

For this package Base Units are defined as units which are not able to be displayed by a multiplication or division of other units.

For Example meters and seconds are Base Units but the combination of both meters / seconds is a Composite Unit.

The following Base Units are currently available:

1.2.1 Length

Module: `pyUnitTypes.length`

ModuleSuperclass: `pyUnitTypes.length.Length`

BaseUnit: `pyUnitTypes.length.Meter`

Available Units

- `KiloMeter`: https://en.wikipedia.org/wiki/Meter#km
- `Meter`: https://en.wikipedia.org/wiki/Meter
- `DeciMeter`: https://en.wikipedia.org/wiki/Meter#dm
- `CentiMeter`: https://en.wikipedia.org/wiki/Meter#cm
- `MilliMeter`: https://en.wikipedia.org/wiki/Meter#mm
- `MicroMeter`: https://en.wikipedia.org/wiki/Meter#Mikrometer
- `NanoMeter`: https://en.wikipedia.org/wiki/Meter#nm
- `Mile`: https://en.wikipedia.org/wiki/Mile
- `Yard`: https://en.wikipedia.org/wiki/Yard
- `Feet`: https://en.wikipedia.org/wiki/Foot_(unit)
- `Inch`: https://en.wikipedia.org/wiki/Inch

1.2.2 Temperature

Module: `pyUnitTypes.temperature`

ModuleSuperclass: `pyUnitTypes.temperature.Temperature`

BaseUnit: `pyUnitTypes.temperature.Celsius`

Available Units

- `Celsius`: https://en.wikipedia.org/wiki/Celsius
- `Fahrenheit`: https://en.wikipedia.org/wiki/Fahrenheit
- `Kelvin`: https://en.wikipedia.org/wiki/Kelvin

1.2.3 Mass

Module: `pyUnitTypes.mass`

ModuleSuperclass: `pyUnitTypes.mass.Mass`

BaseUnit: `pyUnitTypes.mass.KiloGram`

Available Units

- `KiloGram`: https://en.wikipedia.org/wiki/Kilogram
- `Gram`: https://en.wikipedia.org/wiki/Gram
- `MilliGram`: https://en.wikipedia.org/wiki/Kilogram#SI_multiples
- `MicroGram`: https://en.wikipedia.org/wiki/Microgram
- `Pound`: https://en.wikipedia.org/wiki/Pound_(mass)
- `Ton`: https://en.wikipedia.org/wiki/Ton
- `Tonne`: https://en.wikipedia.org/wiki/Tonne
- `ShortTon`: https://en.wikipedia.org/wiki/Ton
- `Ounce`: https://en.wikipedia.org/wiki/Ounce

1.2.4 Time

Module: `pyUnitTypes.time`

ModuleSuperclass: `pyUnitTypes.time.Time`

BaseUnit: `pyUnitTypes.time.Day`

Available Units

- `Year`: https://en.wikipedia.org/wiki/Year
- `Week`: https://en.wikipedia.org/wiki/Week
- `Day`: https://en.wikipedia.org/wiki/Day
- `Hour`: https://en.wikipedia.org/wiki/Hour
- `Minute`: https://en.wikipedia.org/wiki/Minute
- `Second`: https://en.wikipedia.org/wiki/Second
- `MilliSecond`: https://en.wikipedia.org/wiki/Millisecond
- `MicroSecond`: https://en.wikipedia.org/wiki/Microsecond

1.3 Composite Units

For this package Composite Units are defined as units which are able to be displayed by a multiplication or division of other units.

For Example meters, kilograms and seconds are Base Units. But the combination of all meters / seconds creates a new unit Newton:

The following Base Units are currently available:

1.3.1 Area

1.3.2 Volume

1.3.3 Speed

1.3.4 Flow

1.3.5 Force

1.4 Creating your own UnitTypes

It is possible to build your own units based on the provided classes. If you want to extend a existing Base or Composite Unit you'll have to inherit the in your custom class

1.4.1 Extending a existing Unit type

Let's supose we want to create a custom length based unit `Marathon`. As you might know a Marathon is 42 km.

To get started let's create a new module `my-custom-units.py`.

```
from pyUnitTypes.basics import Conversion
from pyUnitTypes.length import Length

class Marathon(Length):
  """Nice."""

  def __init__(self, value=float()):
      """Create instance of the marathon class.

      :param value: (optional, int or float
      """

      super().__init__(name='Marathon', symbol='Marathon', to_
↪base=Conversion(factor=42000, offset=0), value=value)
```

Quite simple right? See the documentation of `pyUnitTypes.length.Length` and `pyUnitTypes.basics.BaseUnit` to understand the parameter used in the superclass constructor.

1.4.2 Creating your own Unit type

But what if we want to create a new base unit, because length, time, temperature is to boring for you.

So you have a lot of pets and you want to figure our old each of your pets is in human years? No problem.

First let's create a new module `age.py` with our super class of the unit type age:

```
from pyUnitTypes.basics import BaseUnit, Conversion

class Age(BaseUnit):
  """
  The Age class is the superclass of all age based unit classes. It provides the␣
↪magic method to calculate
  with the different length based units.
  """

  def __init__(self, name, symbol, to_base, value, from_base=None):
      """Constructor of the Age Superclass. Please don't use this class as standalone.

      :param name: (mandatory, string) name of the unit as word
      :param symbol: (mandatory, string) symbol of the unit
      :param to_base: (mandatory, pyUnitTypes.basics.Conversion) conversion object to␣
↪convert the value to the base
      value
      :param value: (mandatory, float, int or subclass of pyUnitTypes.length.Length)␣
↪The actual value of the class.
      :param from_base: (optional, pyUnitTypes.basics.Conversion) conversion object␣
↪to convert the value back from
      the base to the value of the actual class. Default: inversion of to_base
      """

      super().__init__(name=name, symbol=symbol, unit_type=Age, base_class=HumanYear,␣
↪to_base=to_base,
                       from_base=from_base)

      # store the value and calculate the value in the base class
      if isinstance(value, (float, int)):
          self.value = value
      elif issubclass(type(value), Age):
          self.value = self.from_base(value.base_value)
      else:
          raise TypeError('Can not create object of type {0} from object of type {1}'.
↪format(type(self).__name__,
                                                                                    ␣
↪       type(value).__name__))
```

Note how we set the unit_type, and base_class attribute. And how we allowed a conversion from different Age subclasses.

Now let's add some Units:

```
class HumanYear(Length):
  """The BaseClass of the age.py module"""

  def __init__(self, value=float()):
      """Create instance of the Age class.

      :param value: (optional, int or float
```

(continues on next page)

(continued from previous page)

```
        """

        super().__init__(name='HumanYear', symbol='Human Year(s)', to_base=Conversion(),
↪ value=value)

class DogYear(Length):
  """A dog year is generally know as 7 human years."""

  def __init__(self, value=float()):
        """Create instance of the DogYear class.

        :param value: (optional, int or float
        """

        super().__init__(name='DogYear', symbol='Dog Year(s)', to_
↪base=Conversion(factor=7), value=value)

class CatYear(Length):
  """Funny enough a cat year is also supposed to be 7 human years."""

  def __init__(self, value=float()):
        """Create instance of the DogYear class.

        :param value: (optional, int or float
        """

        super().__init__(name='CatYear', symbol='Cat Year(s)', to_
↪base=Conversion(factor=7), value=value)
```

That wasn't to hard right? So hold old is your 3.5 year old dog and your 4 year old cat? Let's assume your 24 ;-).

```
from age import HumanYear, DogYear, CatYear

# define the ages
my_age = HumanYear(24)
cat_age = CatYear(4)
dog_age = DogYear(3.5)

# check who's older
cat_is_older = cat_age > my_age
dog_is_older = dog_age > my_age

if cat_is_older and dog_is_older:
  print('It seems like your the youngest among your furry friends.')
else:
  if cat_is_older:
    print('Your cat is older. But at least your dog younger.')
  elif dog_is_older:
     print('Your dog is older. But at least your cat is younger.')
  else:
     print('Damn your an old fart.')
```