

Unit Conversion in Python

Asked 14 years, 11 months ago Modified 1 year, 11 months ago

Viewed 79k times



60



I'm working on a project that lets users track different data types over time. Part of the base idea is that a user should be able to enter data using any units that they need to. I've been looking at both units:

<http://pypi.python.org/pypi/units/>

and quantities:

<http://pypi.python.org/pypi/quantities/>

However I'm not sure the best way to go. From what I can tell, quantities is more complex, but includes a better initial list of units.

python

units-of-measurement

Share

Improve this question

Follow

edited Mar 28, 2010 at 22:01



Jon Seigel

12.4k ● 8 ● 60 ● 93

asked Jan 23, 2010 at 23:02



Chris Drackett

1,327 ● 2 ● 10 ● 22

5 I think you need to more specific about what your question is, to receive valid answers. – [Jakob Borg](#) Jan 23, 2010 at 23:38

I assume you'll be storing the data normalised to SI units, so really this is a parsing problem on input, and possibly a conversion problem on output. Use whichever library has the best parsing/conversion - you could use different ones on the way in and out. – [Malcolm Box](#) Jan 28, 2010 at 22:08

2 There are a lot of solutions for this in python, unfortunately. pint.readthedocs.org/en/latest/... – [endolith](#) May 22, 2014 at 15:11

This question appears to be seeking open-ended comparison of two libraries. That definitely does not meet current standards. – [Karl Knechtel](#) Jul 28, 2023 at 2:33

13 Answers

Sorted by:

Highest score (default)



39



I applaud use of explicit units in scientific computing applications. Using explicit units is analogous brushing your teeth. It adds some tedium up front, but the type safety you get can save a lot of trouble in the long run. Like, say, [not crashing \\$125 million orbiters into planets](#).



You should also probably check out these two other python unit/quantity packages:

[Unum](#)



[Scientific.Physics.PhysicalQuantity](#)

I once investigated Scientific.Physics.PhysicalQuantity. It did not quite meet my needs, but might satisfy yours. It's hard to tell what features you need from your brief description.

I ended up writing my own python package for unit conversion and dimensional analysis, but it is not properly packaged for release yet. We are using my unit system in the python bindings for our OpenMM system for GPU accelerated molecular mechanics. You can browse the svn repository of my python units code at:

[SimTK python units](#)

Eventually I intend to package it for distribution. If you find it interesting, please let me know. That might motivate me to package it up sooner. The features I was looking for when I was designing the SimTK python units system included the following:

1. Units are NOT necessarily stored in terms of SI units internally. This is very important for me, because one important application area for us is at the molecular scale. Using SI units internally can lead to exponent overflow in commonly used molecular force calculations. Internally, all unit systems are equally fundamental in SimTK.
2. I wanted similar power and flexibility to the [Boost.Units](#) system in C++. Both because I am familiar with that system, and because it was designed under the scrutiny of a large group of

brilliant engineers. Boost.Units is a well crafted second generation dimensional analysis system. Thus I might argue that the SimTK units system is a third generation system :). Be aware that while Boost.Units is a "zero overhead" system with no runtime cost, all python quantity implementations, including SimTK units, probably exact a runtime cost.

3. I want dimensioned Quantities that are compatible with numpy arrays, but do not necessarily require the python numpy package. In other words, Quantities can be based on either numpy arrays or on built in python types.

What features are important to you?

Share Improve this answer

Follow

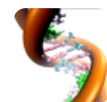
edited Nov 25, 2019 at 18:46



nixin72

342 ● 6 ● 16

answered Feb 5, 2010 at 17:49



Christopher Bruns

9,490 ● 7 ● 47 ● 63



28



[Pint](#) has recently come onto the field. Anybody care to share their experiences? Looks good. FYI: It looks like [Pint will be integrated with Uncertainties](#) in the near future.



Share Improve this answer

edited Dec 20, 2013 at 16:59

Follow

answered Jun 11, 2013 at 0:06



Mark Mikofski

20.1k ● 2 ● 60 ● 92

-
- 3 I am evaluating Pint at I write this and but one big +1 is that it is able to use Decimal which means no funky float rounding errors - all the mentioned libraries mentioned seem to only work in float type - python-in-the-lab.blogspot.ca/2013/01/... – Daniel Sokolowski Aug 13, 2013 at 16:37
-
- 2 I am no longer as big +1 for Pint - in the last two days I have submitted so far two bug fixes - I am debating if I really need the extra functionality/complexity and instead could just use something straight forward like code.activestate.com/recipes/... – Daniel Sokolowski Aug 16, 2013 at 20:03
-
- 1 @user90855 Take a look at the update of the post. The bug was a regression in the development branch that was corrected before it landed into master. – Hernan Jun 17, 2014 at 0:29
-

Pint doesn't actually support decimals throughout, unfortunately; internally, constants are represented as float and casted into decimals from that, which causes imprecisions. – Clément Apr 17, 2019 at 18:48



11

There is another package called [unyt](#) from the yt-project. The authors of unyt acknowledge the existence of Pint and astropy.units. Conversions from and to these other packages is supported.



The selling point of unyt is *speed*. It is faster than the other two. The unit packages are compared in several benchmarks in this [paper](#).



The benchmarks are disappointing for anyone obsessed with performance. :-(The slowdown of calculations with any of these unit systems is large. The **slowdown factor is 6-10** for arrays with 1000 entries (worse for smaller arrays).

Disclaimer: I am not affiliated with unyt, I just want to share what I learned about unit systems.

Share Improve this answer

answered Jun 12, 2018 at 15:07

Follow



olq_plo

1,232 ● 1 ● 13 ● 19



Note that `quantities` has very bad support for temperature:

9



```
>>> (100 * pq.degC).rescale(pq.degF)
array(179.99999999999997) * degF
>>> (0 * pq.degC).rescale(pq.degF)
array(0.0) * degF
```



0 degrees Celsius isn't 0 degrees Fahrenheit. Their framework doesn't support any kind of conversion that isn't just multiplying by a factor.

Share Improve this answer

answered Feb 14, 2012 at 9:02

Follow



ubershmekel

12.8k ● 10 ● 80 ● 98

-
- 5 Temperature is tricky because there are two types - absolute/thermodynamic temperature and relative/temperature difference. In absolute, everyone knows $0\text{ }^{\circ}\text{F} \neq 0\text{ }^{\circ}\text{C}$. When talking about a temperature difference (ΔT), $0\text{ }^{\circ}\text{F} == 0\text{ }^{\circ}\text{C}$. – [Mr Anderson](#) Aug 29, 2017 at 15:15 ✎
-

As of February 21st, 2020, the $0\text{ }^{\circ}\text{C} \rightarrow 0\text{ }^{\circ}\text{F}$ behavior still happens in quantities==0.12.4.

– [Intrastellar Explorer](#) Feb 21, 2020 at 23:09



9



I am surprised that nobody mentioned SymPy yet. [SymPy](#) is a mature and well-maintained symbolic mathematics library for Python that is moreover a [NumFOCUS-sponsored](#) project.

It has a [Physics](#) module with many useful classes and functions for "solving problems in physics". Most relevant for you, it has a [Unit](#) sub-module that contains everything you need, I think; just read the excellent documentation.

Share Improve this answer

edited Jul 7, 2017 at 12:18

Follow

answered May 30, 2017 at 16:34



okartal

2,986 ● 1 ● 16 ● 11



6

You may want to look at a new package called [natu](#). It addresses the three issues that @ChristopherBruns listed. It's available in [PyPI](#).



I'm the author of that package, and I'd appreciate any comments or suggestions.



Share Improve this answer

answered Aug 8, 2014 at 19:06

Follow



[kdavies4](#)

178 ● 2 ● 6



5

It looks like another package has come out for doing this as well, written by Massimo DiPierro of web2py fame, called [Buckingham](#).



Also of note, [Brian](#) has had something like this for some time.



Share Improve this answer

edited Feb 25, 2016 at 17:49

Follow



[Dag Høidahl](#)

8,315 ● 8 ● 55 ● 70


answered Oct 20, 2010 at 23:46



[James Snyder](#)

4,072 ● 2 ● 21 ● 16

Buckingham seems incomplete and can't convert from gram to pounds for example: `>>> (Number(100, dims='gram')).convert('pound').value` results in `RuntimeError: Incompatible Dimensions`
– [Daniel Sokolowski](#) Aug 19, 2013 at 19:57

- 2 Daniel, The particular error you mention there is because the internal definition for pound is as a unit of force rather than mass. Additionally, it certainly doesn't have an exhaustive list of supported units. – [James Snyder](#) Aug 19, 2013 at 22:16 

Ahh that would make sense, to add pound mass support add `'lb': (453.592, 0, 0, 1, 0, 0, 0), # lb` to the `UNITS` list. – [Daniel Sokolowski](#) Aug 20, 2013 at 13:53



Thought to mention the [units](#) package which is part of the Astropy package.

3



It's well maintained, easy to use, and has all the basic units (as well as astrophysics-related units). It provides tools for both units and quantities. And there's also a module for [physical constants](#).



Share Improve this answer

[edited Feb 18, 2020 at 6:23](#)

Follow

answered Feb 12, 2020 at 8:34



[Elinor Medezinski](#)

51 ● 5



I'd like to point to a separate library for dealing with units: Barril

2



<https://github.com/ESSS/barril>

Docs at: <https://barril.readthedocs.io/en/latest/>



While it does have support for creating "random" units from computation (such as Pint, unum, etc), it's more tailored to having a database of units (which the library has by default -- see:

<https://barril.readthedocs.io/en/latest/units.html> and the

implementation:

<https://github.com/ESSS/barril/blob/master/src/barril/units/posc.py>) and then you can query and transform based on the related units.

One thing it supports that does a lot of difference in that regard is dealing with unit conversions which would be "dimensionless" -- such as m^3/m^3 (i.e.: volume per volume) and then converting to cm^3/m^3 and keeping the dimension.

i.e.: in pint:

```
>>> import pint
>>> ureg = pint.UnitRegistry()
>>> m = ureg.meter
>>> v = 1 \* (m\*3)/(m\*3)
>>> v
<Quantity(1.0, 'dimensionless')>
```

And then, after that (as far as I know), it's not really possible to do additional unit conversions properly knowing that it was m^3/m^3 .

In barril:

```
>>> from barril.units import Scalar
>>> a = Scalar(3, 'm3/m3')
```

```
>>> a.GetValue('cm3/m3')
3000000.0
>>> a.category
'volume per volume'
>>> a.unit
'm3/m3'
```

and something as `a.GetValue('m3')` (with an invalid value) would give an error saying that the conversion is actually invalid.

The unit database (which was initially based on the POSC Units of Measure Dictionary) is a bit more tailored for the Oil & Gas field, but should be usable outside of it too.

Share Improve this answer

answered Sep 4, 2021 at 13:41

Follow



Fabio Zadrozny

25.3k ● 5 ● 67 ● 79



I think you should use quantities, because a quantity has some units associated with it.

1



Pressure, for example, will be a quantity that could be entered and converted in and to different units (Pa, psi, atm, etc). Probably you could create new quantities specifics for your application.



Share Improve this answer

answered Jan 24, 2010 at 15:20

Follow



Pedro Ghilardi

1,944 ● 1 ● 17 ● 19



1



My preferred package is [QuantiPhy](#). It takes a different approach than most other packages. With QuantiPhy the units are simply strings, and the package is largely used when reading or writing quantities. As such, it is much easier to incorporate into your software. QuantiPhy supports unit and scale factor conversion both when creating quantities and when rendering them to strings. Here is an example that reads and then writes a table of times and temperatures, converting from minutes/°F to seconds/K on the way in and back to the original units on the way out:

```
>>> from quantiphy import Quantity

>>> rawdata = '0 450, 10 400, 20 360'
>>> data = []
>>> for pair in rawdata.split(','):
...     time, temp = pair.split()
...     time = Quantity(time, 'min', scale='s')
...     temp = Quantity(temp, '°F', scale='K')
...     data += [(time, temp)]

>>> for time, temp in data:
...     print(f'{time:9q} {temp:9q}')
      0 s   505.37 K
     600 s   477.59 K
    1.2 ks   455.37 K

>>> for time, temp in data:
...     print(f"{time:<7smin} {temp:s°F}")
0 min   450 °F
10 min  400 °F
20 min  360 °F
```



0



I find the units packages to be more than what want. It doesn't take much code to start building your own functions that refer back to the very few basic fundamental numbers. Also, It forces you to do the dimensional analysis to prevent errors.



```
def FtoC(Tf):  
    return (Tf-32)*5/9  
def CtoF(Tc):  
    return 9*Tc/5+32  
def CtoK(Tc):  
    return Tc+273.15  
def INCHtoCM(Inch):  
    return 2.54 * Inch  
def CMtoINCH(cm):  
    return cm / INCHtoCM(1)  
def INCHtoMETER(inch):  
    return .01*INCHtoCM(inch)  
def FOOTtoMETER(foot):  
    return INCHtoMETER(12*foot)  
def METERtoINCH(Meter):  
    return CMtoINCH(100 * Meter)  
def METERtoFOOT(Meter):  
    return METERtoINCH(Meter)/12  
def M3toINCH3(M3):  
    return (METERtoINCH(M3))**3  
def INCH3toGALLON(Inch3):  
    return Inch3 / 231  
def M3toGALLON(M3):  
    return INCH3toGALLON(M3toINCH3(M3))  
def KG_M3toLB_GALLON(KGperM3):  
    return KGtoLBM(KGperM3) / M3toGALLON(1)  
def BARtoPASCAL(bar):  
    return 100000 * bar  
def KGtoLBM(kilogram):  
    return kilogram * 2.20462262185
```

```

def LBMtoKG(lbm):
    return lbm/KGtoLBM(1)
def NEWTONtoLBF(newton):
    return newton * KGtoLBM(1) * METERToFOOT(1) / STAN
def LBFtoNEWTON(lbf):
    return lbf * STANDARD_GRAVITY_IMPERIAL() * LBMtoKG
def STANDARD_GRAVITY_IMPERIAL():
    return 32.174049
def STANDARD_GRAVITY_SI():
    return 9.80665
def PASCALtoPSI(pascal):
    return pascal * NEWTONtoLBF(1) / METERToINCH(1)**2
def PSItoPASCAL(psi):
    return psi * LBFtoNEWTON(1) / INCHtoMETER(1)**2

```

Then let's say you want to plot the static head pressure of 1,3 Butadiene @44 F and use gauges in PSI because you live in the US but the density tables are in SI units as they should be.....

```

# butadiene temperature in Fahrenheit
Tf = 44

# DIPPR105 Equation Parameters (Density in kg/m3, T in
# valid in temperature 165 to 424 Kelvin
A=66.9883
B=0.272506
C=425.17
D=0.288139

# array of pressures in psi
Pscale = np.arange(0,5,.1, dtype=float)
Tk = CtoK(FtoC(44))
Density = A / (B**((1+(1-Tk/C)**D)) # KG/M3
Height = [PSItoPASCAL(P) / (Density * STANDARD_GRAVITY
Height_inch = METERToINCH(1) * np.array(Height, dtype

```

Share Improve this answer

Follow



David Filler

196 ● 7



0



Another package to mention is [Axiompy](#).

Installation: `pip install axiompy`

```
from axiompy import Units
units = Units()
print(units.unit_convert(3 * units.metre, units.foot))
>>> <Value (9.84251968503937 <Unit (foot)>)>
```

Share Improve this answer

Follow

answered Mar 26, 2022 at 1:12



anonymousse

1