solutions

February 23, 2016

1 Computing with units exercise notebook

1.1 Task 1

Import `UnitRegistry` from the pint package and import the `numpy` library as well. Then create an instance of `UnitRegistry` as explained in the presentation (check the *LiveCoding.ipynb* notebook on the desktop).

```
In [1]: from pint import UnitRegistry
        import numpy
        ureg = UnitRegistry()
```

Define a physical quantity with the Pint commands explained in the presentation. Specifically, create a variable that represents the mass m:

$$m = 11.4 \quad Kg$$

```
In [2]: m = 11.4 * ureg.kilogram
```

Now define another physical quantity, the acceleration a:

$$a = 12.3 \quad \frac{m}{s^2}$$

```
In [3]: a = 12.3 * (ureg.meter / ureg.second ** 2)
```

Familiarize with the outputs you can get from these quantities and print to the screen fro both variables a and m:

1. Print the quantity.
2. Print the magnitutde.
3. Print the unit(s).
4. Print the dimension.

```
In [4]: print(m)
        print(m.magnitude)
        print(m.units)
        print(m.dimensionality)
        print("\n")
        print(a)
        print(a.magnitude)
        print(a.units)
        print(a.dimensionality)
```

```
11.4 kilogram
11.4
kilogram
[mass]
```

```
12.3 meter / second ** 2
12.3
meter / second ** 2
[length] / [time] ** 2
```

Build a new variable F, force, from the 2 variables m and a employing the well known formula:

$$F = ma$$

```
In [5]: F = m * a
```

Then

1. Print the F variable.
2. Print its units.
3. Print its magnitude.
4. Print its dimension.

```
In [6]: print(F)
        print(F.magnitude)
        print(F.units)
        print(F.dimensionality)
```

```
140.22 kilogram * meter / second ** 2
140.22
kilogram * meter / second ** 2
[length] * [mass] / [time] ** 2
```

Now try to convert F in newton (*permanent conversion*) with the Pint command explained in the presentation. Print again F, does its magnitude changed? And its unit? Why?

```
In [7]: F.ito(ureg.newton) # Permanent conversion

        print(F)           # Doesn't change as Kg*m / s^2 is the unit definition of Newton
```

```
140.22 newton
```

Try to print only (so employing *on the fly conversion*) F in *Dyne*. To check if your variable is yet expressed in newton after you expressed in the new unit print both F in *Dyne* (*on the fly conversion*) and F.

```
In [8]: print(F.to(ureg.dyne)) # On the fly conversion
        print(F)
```

```
14022000.0 dyne
140.22 newton
```

Define 2 new (force quantities) variables $G1$ and $G2$ as follows:

$$G1 = 4.3 \qquad Dyne$$
$$G2 = -2.3 \qquad Kilogram - force$$

```
In [9]: G1 = 4.3 * ureg.dyne
        G2 = -2.3 * ureg.kilogram_force
```

Now try to sum F, $G1$ and $G2$ in these 3 diffrent orders:

$$S1 = F + G1 + G2$$

$$S2 = G1 + F + G2$$

$$S3 = G2 + G1 + F$$

```
In [10]: S1 = F + G1 + G2
         S2 = G1 + F + G2
         S3 = G2 + G1 + F

         print(S1)
         print(S2)
         print(S3)
```

```
117.664748 newton
11766474.8 dyne
11.9984651232 force_kilogram
```

The last exercise highlighted the fact that with Pint you can sum variables with different units but same dimensions together without converting them. However, you get 3 different results, in term of units, with the 3 different sums. Can you get the logic behind these difference and how to obtain directly the sum in the desired units?

To check if your guess is right, try to define these lengths:

$$\begin{aligned} l1 &= 1 \quad & m \\ l2 &= 2 \quad & cm \\ l3 &= 6 \quad & mm \end{aligned}$$

Sum these quantities in a way to get the result directly in *centimeters*.

```
In [11]: l1 = 1 * ureg.meter
         l2 = 2 * ureg.centimeter
         l3 = 6 * ureg.millimeter

         # The sum will be expressed in the unit of the first element
         print(l2 + l1 + l3)
```

```
102.6 centimeter
```

1.2 Task 2

With Pint is possible to define custom units of measure and also interact with the `numpy` library.

Let's pretend we want to produce some (really simple) statistical results regarding the alchol consumption of a group of PhD students in UK. Unfortunately, all the data we managed to collect are expressed in shots, a unit not implemented in Pint.

First, define a new unit *shot* equals to 30 *milliliters* as explained in the presentation with:

- Canonical name: *shot*
- Alias: *sh*
- Definition: $shot = 30 \quad ml$

```
In [12]: ureg.define('shot = 30 * milliliter = sh')
```

We got the data from an anonymous group of PhD students about the alchol consumption (in *shots*) for every day of the week:

$$[2, 3, 2, 5, 10, 12, 1] \quad shot$$

Build a single object containing the shot quantities showed above, employing `numpy.array` command as explained in the presentation.

Now try to:

1. Get the total number of shots consumed in that week employing `numpy.sum` command.
2. Get the week mean of the number of shots employing `numpy.mean`.
3. Convert (element wise) the quantities in *liters* with a `for` cycle. In this step you could use a `list` approach or a `numpy` approach. In both cases be aware that **quantities are objects**. Then if you decide to employ the `numpy` approach remeber that quantities cannot be stored in an array, in which you should store **the magnitudes** only. On the contrary, you can store quantities directly as `list` elements.
4. Get the total number and the mean of these quantities in *liters*.
5. Print to the screen all the quantities calculated.

```
In [13]: week_shots = numpy.array([2,3,2,5,10,12,1]) * ureg.shot
         total_shots = numpy.sum(week_shots)
         mean_shots = numpy.mean(week_shots)

         # Converting in liters - LIST VERSION
         days = len(week_shots)
         week_liters = []
         for day in range(days):
             week_liters.append(week_shots[day].to(ureg.liter))

         total_liters = sum(week_liters)
         mean_liters = total_liters / float(days)

         # # Converting in liters - NUMPY VERSION
         # days = len(week_shots)
         # liters_magnitude = numpy.zeros(days)
         # for day in range(days):
         #     liters_magnitude[day] = week_shots[day].to(ureg.liter).magnitude

         # week_liters = liters_magnitude * ureg.liter

         # total_liters = numpy.sum(week_liters)
         # mean_liters = numpy.mean(week_liters)


         print('The PhD students consumed {} in a week').format(total_shots)
         print('The PhD students consumed about {} per day').format(mean_shots)
         print('The PhD students consumed {} of alchol in a week').format(total_liters)
         print('The PhD students consumed about {} of alchol per day').format(mean_liters)
```

```
The PhD students consumed 35 shot in a week
The PhD students consumed about 5.0 shot per day
The PhD students consumed 1.05 liter of alchol in a week
The PhD students consumed about 0.15 liter of alchol per day
```

1.3 Task 3 (more an example)

A really useful feature of Pint is that it can read units also as strings. Let's pretend that a messy engineer wrote a document "**data.txt**" in which there are some lengths but expressed in different units (look at the file in the desktop).

Your task is to extract the quantities and convert all in meters and then print to the screen. Half of the work has been already done as in this way you can focus on Pint features and their interactions with Python instead of Python commands itself. However, if you are really confident with Python you can ignore our suggested code and try to get the data from the *.txt* file on your own.

Our suggested procedure is:

1. Open the "**data.txt**" file from the desktop and understand its structure (hint: "**data.txt**" has **10 lines**, you could do a `for` cycle for these 10 lines).
2. Then, try to extract single strings for each line of the file with the `open` (in read mode), `readline` (here do a `for` cycle for the **10 lines**) commands (look at this link if you don't remember how these commands work).
3. Then we suggest to use `split` command to separate the number from the unit into two separate string variables.
4. Build a physical quantities for each line with Pint commands (be aware of the type of the variables!).
5. Store these quantities in a list with `append` command.
6. Convert all elements in the list with Pint commands (hint: do a `for` cycle for each element of the list).
7. Print the obtained quantities.

```
In [14]: f = open('data.txt', 'r')
         number_lines = 10
         data = []
         for line in range(number_lines):
             quantity = f.readline()
             quantity = quantity.split() # Splitting magnitude and unit
             number = float(quantity[0]) # The magnitude is stored in the first column
             unit = quantity[1]          # The unit is stored in the second column
             data.append(number * ureg(unit))
         f.close()
```

```
In [15]: for element in range(len(data)):
             print(data[element])
```

```
12.1 meter
23.3 meter
0.01 centimeter
1.03 inch
12.3 kilometer
24.0 mile
92.1 meter
12.7 millimeter
22.0 meter
33.1 centimeter
```

All the quantities are now stored in `data` list. Try to convert them all in meters and print them to the screen.

```
In [16]: data_in_meters = []
         for element in range(len(data)):
             data_in_meters.append(data[element].to(ureg.meter))
             print(data_in_meters[element])
```

```
12.1 meter
23.3 meter
0.0001 meter
0.026162 meter
12300.0 meter
38624.256 meter
92.1 meter
0.0127 meter
22.0 meter
0.331 meter
```

1.4 Task 4 (for the braves)

Pint presents also some advanced features as the **Buckingham π theorem** feature explained in the presentation. Let's pretend you don't know much about physics and you have to solve a simple pendulum problem employing only the `pi_theorem` feature and the Pint library. The final goal is to calculate the period T of the system showed below.

First, import `pi_theorem` from Pint.

```
In [17]: from pint import pi_theorem
```

The **Buckingham π theorem** states that if you are dealing with an equation associated to a physical system involving:

- n numbers of physical variables (as Velocity, Acceleration, Force, ...)
- k numbers of independent fundamental quantities (as [time], [length], ...)

Then you can express the equation in terms of:

$$p = n - k$$

p dimensionless numbers.

In the system proposed above we have:

- T (time period), M (mass), l (length) and g (acceleration) as physical quantities. Then $n = 4$.
- [time], [mass], [length] as independent fundamental quantities (acceleration doesn't add any other fundamental units as its dimensions are $= [length]/[time]^2$). Then $k = 3$.

So, as $p = 4 - 3 = 1$ we can express our equation with one dimensionless number Π. Try to get the number Π employing the `ureg.pi_theorem` command as explained in the presentation.

```
In [18]: ureg.pi_theorem({'T' : '[time]',
                          'M' : '[mass]',
                          'l' : '[length]',
                          'g' : '[acceleration]'}) # Or also '[length]/[time]**2'
```

```
Out[18]: [{'T': 2.0, 'g': 1.0, 'l': -1.0}]
```

```
In [19]: # So the formula is ---> Pi = T^2 * g / l
```

The result should indicates that Π can be obtained multiplyng T, g, l elevated for given powers:

$$\Pi = T^{x_1} g^{x_2} l^{x_3}$$

From the output of the `ureg.pi_theorem` command you should get the coefficients x_1, x_2 and x_3. Then you have a formula to get the period of the pendulum T as a function of Π, g and l (write down on paper the formula).

Assuming that:

$$\Pi = 2\pi$$
$$g = 9.8 \quad \frac{m}{s^2}$$
$$l = 0.3 \quad m$$

Obtain the period of the pendulum T expressed in *minutes*.

```
In [20]: Pi = 2 * numpy.pi
         g = 9.8 * (ureg.meter / ureg.second ** 2)
         l = 0.3 * ureg.meter

         # Pi = T^2 * g / l ---> T = Pi * square_root(l / g)
         T = Pi * numpy.sqrt(l / g)

         # T now is expressed in seconds
         T.ito(ureg.minute) # Permanent conversion
         print(T)
```

```
0.0183221404309 minute
```