Test n°?

Duration : 1 hour

This text will only appear on the answer sheet.
Useful to remind students to be careful about their writing, ...

This "test" gives an overlook over the functionalities provided.

Exercise 1: Getting started – changing the formatting

As you can see, exercises are defined using the `section` command ; one exercise.

In practice, this paragraph could provide the context of a problem, which would be treated in several sub-parts (defined by the command `subsection`).

1. Adaptive document structure

One of the main features of this class is the use of conditionals to alter the *structure* of the document.

1. Change the `answerSheet` toggle (in the preamble of the `.tex` document, line 20) to `true` to change the entire document, and see the results to those questions.
2. How are points indicated, when in `answerSheet` mode ?
3. Can you guess how I integrated this class into my workflow to be efficient ?

2. Other cosmetic settings

Other macros and environment are provided for convenience ; they are only wrappers around packages :

Document 1: Cosmetic environments and macros

Preamble macros :

- `\title{Test n°?}` : Title at the top of the sheet, **and** on the top right corner of every page.
- `\duration{1 hour}` : small text below the title. Only appears on the test subject.
- `\class{CLASS}` : Text on the top left corner of the page.
- `\date{Week n°??}` : Text on the bottom left corner of the page.
- `\colorlet{cPrim}{...}` and `\definecolor{cPrim}{...}` : Changes the accent color. I had a color for each grade level, which was handy for personal organisation, in addition to being pretty.

Provided environments :

- `\begin{doc}[Additionnal title]` : A framed document for the student's to analyse, like this one ! Actually just a `mdframed` environment.
- `\begin{answer}` : an `mdframed` environment wrapped around an `environ` custom environment[a], that is displayed only in the answer sheet.
- `enumerate`, as modified by the `enumitem` package : improved enumerated lists, allowing enumeration to be broken then resumed with the optional argument `resume`, and modification of items labels.

Pre-loaded packages :

- `multicols`
- `TiKZ`
- `mdframed`
- `enumitem`
- `xcolor` with `dvipsnames`
- `etoolbox`
- `siunitx`

[a]meaning that verbatim environments will not work inside the `answer` env !

1. Since the **mdframed** package comes loaded, it is possible to create your own frames, like those below :

Data sheet

Data sheets are vital in physics ! We always have values and formulae to use, but sometimes knowing them by heart is not required.

- $v = \frac{d}{\Delta t}$
- The speed of light is $3{,}00 \times 10^8\,\mathrm{m \cdot s^{-1}}$. Note the use of `siunitx` for typesetting the value !

Warning !

This information is really important : students really ought not to miss it!

2. See how, despite having exited the previous `enumerate` environment, the numbering resumed ? That's thanks to the `resume` argument provided by the `enumitem` package.

Exercise 2: PythonTeX and its wrappers

1. Context

The second main feature of this class it to *be able to compute answers*, rather than type them.

High school physics calculations tend to always follow the same structure, being :

- Write down the formula ;
- Replace the terms with their values (and optionally their unit) ;
- Compute the final value, and the final unit.

For example, a student would write this on their paper :

$$\begin{aligned} v &= \frac{d}{\Delta t} \\ &= \frac{30\,\mathrm{m}}{4{,}0\,\mathrm{s}} \\ &= 7{,}5\,\mathrm{m}\cdot\mathrm{s}^{-1} \end{aligned}$$

Now, typing this about 20 times by hand into an answer sheet can easily lead to mistakes: I tend to, for example:

- Change the value in the second step, but not in the third;
- Forget to convert some units between the second and third step (if I add an intermediate unit conversion);
- Changing the units of quantities in the question; and forgetting to change it in the answer;
- Being inconsistent with the significant figures;
- and so on...

To sum up, this is an almost automatic process, usually done by hand, and prone to many errors...Why not automate it?

2. Python, PythonTEX

2.1 Why Python in LATEX ?

To do that, I decided to integrate Python into the TEX document using the PythonTeX library. I prefer this method to a pure LATEX solution for the following reasons:

- I'm not comfortable programming in LATEX; I might have gotten away with the automatic calculation of values, but certainly not with that of units;
- I came across a Python library that already did exactly what I needed: the `physics.py` library written by Georg Brandl. It was much easier to adapt this library to produce LATEX code than to rewrite everything from scratch.
- Programming in Python for physics is part of the curriculum; I had to have a working Python distribution installed already.

2.2 Quick example

To reuse the example above in subsection 1, it's possible to do the following (follow along in the `.tex` document !)

1. Define the quantities using a `pycode` block ;
2. Type the formula :
$$v = \frac{d}{\Delta t}$$
3. Retype the same formula, changing every symbols we want the value of with their Python variable, wrapped around the `\Py` macro :
$$v = \frac{30\,\mathrm{m}}{4\,\mathrm{s}}$$

4. Retype the same formula, *within the* \Py *macro, as if it was a Python operation* :

$$v = 7{,}5\,\mathrm{m}\cdot\mathrm{s}^{-1}$$

5. Compile the document, following the PythonTEX doc :
   XeLATEX → PythonTex → XeLATEX again.

Now, try to change the values in the `pycode` block above and recompile the document (step 5) : you will see the values are automatically changed !

Note that while the result is the expected one, the significant digits of Δt are not correctly displayed... This is a problem with the way Python formats strings, and has no solution yet, other than a hack (shown in doc. 3): which is to manually add the significant zero using the optional argument to `\Py` :

$$v = \frac{30\,\mathrm{m}}{4{,}0\,\mathrm{s}}$$

2.3 General usage

Below is a list of commands and macros that can be used in a document:

Document 2: LATEX macros

`\Py[additionnal]{PhysicalQuantity}`

Wrapper around the PythonTEX `\pyc` command.
Calls the Python `PhysicalQuantity.latex` method, which prints the string `\SI{value+additionnal}{unit}`, and renders it with LATEX.

The `additionnal` can be used to manually add missing significant digits, or uncertainties (which are not yet automatically computed).

`\PyGray[additionnal]{PhysicalQuantity}`

Identical to `\Py`, except that the unit is displayed in gray.

The curriculum states that writing the unit in the development of the calculation is optional. Thus, graying out the unit helps to convey this idea, while making it clear that using units is useful (and often times, remind the students that units exist at all...).

Document 3: Python commands (to use within a `pyblock`)

`Q(value, unit=None, stdev=None, prec=3, force_scientific=False)`

The `PhysicalQuantity` (aliased `Q`) object. There are two constructor calling patterns:

1. `PhysicalQuantity(value, unit)`, where value is any number and unit is a string defining the unit
2. `PhysicalQuantity(value_with_unit)`, where `value_with_unit` is a string that contains both the value and the unit, i.e. `'1.5 m/s'`. This form is provided for more convenient interactive use.

`stdev` Integer. Standard deviation, or uncertainty. At the moment, cosmetic only ; it is not automatically computed, unlike units.

```
>>> from physics import Q
>>> print(Q("150 m/s").latex())
\SI{1.50e+02}{\meter\second\tothe{-1}}
>>> print(Q("150 m/s", stdev=0.4).latex())
\SI{1.50(0.4)e+02}{\meter\second\tothe{-1}}
```

`prec` Integer. Number of digits that should be displayed. Uses the Python general (g) string formatting method to display those SD.

```
>>> print(Q("150 m/s").latex())         # Default : 3 digits
\SI{1.50e+02}{\meter\second\tothe{-1}}
>>> print(Q("150 m/s", prec=5).latex()) # 5 digits
\SI{1.5000e+02}{\meter\second\tothe{-1}}
```

Warning : Since string formatting is used, it inherits its faults ; mainly, if the value is too close to 1, the scientific notation will not be used and significant digits will be ignored (see below).

```
>>> print(Q("1.5 m/s").latex())         # Expected value : 1.50
\SI{1.5}{\meter\second\tothe{-1}}
>>> print(Q("1.5 m/s", prec=5).latex()) # Expected value : 1.5000
\SI{1.5}{\meter\second\tothe{-1}}
```

Workaround : Additional 0s can be added manually from LaTeX for now, using `\Py`'s optional argument. To keep the same example :

```
      \Py{Q("1.5 m/s")} :   1,5 m · s^-1
   \Py[0]{Q("1.5 m/s")} :   1,50 m · s^-1
 \Py[000]{Q("1.5 m/s")} :   1,5000 m · s^-1
```

`force_scientific` Boolean. Forces the use of the scientific notation (e) string formatting method to display quantity values.

```
>>> print(Q("15 m/s").latex())
\SI{15}{\meter\second\tothe{-1}}
>>> print(Q("15 m/s", force_scientific=True).latex())
\SI{1.50e+01}{\meter\second\tothe{-1}}
```