Test n°? – Answer sheet

Introduction to the theme of the test : you can add text, images, quotes, etc. The current "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 that will be addressed in several sub-parts (defined by the `subsection` command).

1. Adaptive document structure

One of the main features of this class is the use of conditionals to modify the document's structure.

1. To change the entire document and see the results for those questions, set the `answerSheet` toggle to `true` (located in the preamble of the `.tex` document, line 20).

   Answer

   This is how you do it! Notice that:

   - A margin appears on the right where the questions' points are displayed.
   - The reminder paragraph below the title disappears.
   - The title now includes "Answer sheet."

   All of this can be achieved with the flip of a switch!

2. How are points indicated when in `answerSheet` mode?

   Answer

   Points for a given question are assigned using the `pts` command. For example, by using `\pts{0.5}` at the end of this paragraph, the value of this item is displayed in the right margin. /0.5

   You can use this command multiple times in a single answer if the answer requires multiple parts graded individually. /0.5

   Please note that points only appear in `answerSheet` mode.

3. Can you guess how I integrated this class into my workflow to be efficient ?

   Answer

   My idea was to :

   - Write a draft test subject in this document without the answers.
   - Quickly try it out on paper to prevent any major oversights in the test's design.
   - Switch to `answerSheet` mode.
   - Correct typos in the questions *while* coding[a] the answer sheet.

That way, I can design a test *only by writing its answer sheet*.

This allows me to design a test by *only writing its answer sheet*, eliminating the risk of pasting content in the wrong document and ensuring no missing answer key.

Additionally, the entire document is automatically formatted, saving significant time.

To minimize the risk of errors, values are coded rather than typed, with the computer computing the results and displaying them. See section 2 for more details.

[a] Coding, not typing !

2. Other cosmetic settings

This section outlines various macros, environments, and pre-loaded packages designed to enhance the cosmetic aspects of your document. These elements are provided for convenience and serve as wrappers around specific packages:

Document 1: Cosmetic environments and macros

Preamble macros :

- `\title{Test n°?}` : Sets the title at the top of the sheet **and** in the top right corner of every page.
- `\duration{1 hour}` : Displays small text below the title, but it only appears on the test subject.
- `\class{CLASS}` : Places text on the top left corner of the page.
- `\date{Week n°??}` : Includes text on the bottom left corner of the page.
- `\colorlet{cPrim}{...}` and `\definecolor{cPrim}{...}` : Allows you to change the accent color. This is particularly useful for organizing content by grade level and for aesthetic purposes.

Provided environments :

- `\begin{doc}[Additionnal title]` : This creates a framed document for students to analyze, similar to the one you're currently reading. It's essentially an `mdframed` environment.
- `\begin{answer}` : This environment is an `mdframed` wrapper around an `environ` custom environment that's displayed exclusively in the answer sheet.
- `enumerate`, as modified by the `enumitem` package : This enhances enumerated lists by allowing enumeration to be interrupted and resumed using the optional argument `resume`. It also provides modifications for item labels.

Pre-loaded packages :

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

1. Since the `mdframed` package is preloaded, you can create custom frames, as demonstrated below:

Data sheet

Data sheets are essential in physics. We often deal with values and formulas, but memorizing them is not always necessary.

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

Warning !

This information is crucial; students should not overlook it.

2. Notice how, despite exiting the previous `enumerate` environment, the numbering resumes. This is made possible by the `resume` argument provided by the `enumitem` package.

Exercise 2: PythonTeX and its wrappers

1. Context

The second significant feature of this class is its *ability to compute answers*, rather than manually type them.

In high school physics, calculations often follow a consistent structure:

- 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 might write the following on their paper:

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

Manually copying this process into an answer sheet multiple times can lead to errors. Common mistakes include:

- Changing the value in the second step but not in the third.
- Forgetting to convert units between the second and third step (if an intermediate unit conversion is added).
- Changing the units of quantities in the question but forgetting to update them in the answer.
- Inconsistencies with significant figures, and more.

In summary, this is an almost automatic but error-prone process when done by hand. Why not automate it?

2. Python, PythonTeX

2.1 Why Python in LaTeX ?

To accomplish this, I decided to integrate Python into the TeX document using the PythonTeX library. I prefer this approach over a purely LaTeX-based solution for several reasons:

- I'm not comfortable with programming in LaTeX; I might have managed automatic value calculations, but certainly not unit conversions. I discovered a Python library, `physics.py`, authored by Georg Brandl, that already performed exactly what I needed. It was much easier to adapt this library to generate LaTeX code than to start from scratch.
- Programming in Python for physics aligns with the curriculum, so I already had a working Python distribution installed.

2.2 Quick example

To reuse the example from subsection 1, follow these steps (refer to the `.tex` document for the full process):

1. Define the quantities using a `pycode` block ;
2. Type the formula :
$$v = \frac{d}{\Delta t}$$
3. Re-enter the same formula, replacing every symbol whose value you want with the corresponding Python variable, encapsulated within the `\Py` macro :
$$v = \frac{30\,\mathrm{m}}{4\,\mathrm{s}}$$
4. Re-enter the same formula, *this time within the `\Py` macro, as if it were a Python operation* :
$$v = 7{,}5\,\mathrm{m}\cdot\mathrm{s}^{-1}$$
5. Compile the document following the PythonTeX documentation :
XeLaTeX → PythonTex → XeLaTeX again.

Now, try changing the values in the `pycode` block above and recompile the document (step 5). You will see that the values are automatically updated.

Note that, while the result is as expected, the significant digits of Δt are not correctly displayed. This issue stems from the way Python formats strings and currently has no solution other than a workaround (explained in doc. 3), which involves manually adding the significant zero using the optional argument with `\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}`

A wrapper around the PythonTeX `\pyc` command, which calls the Python `PhysicalQuantity.latex` method. It prints the string `\SI{value+additional}{unit}` and renders it using LaTeX.

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

`\PyGray[additionnal]{PhysicalQuantity}`

This macro is identical to `\Py`, except that it displays the unit in gray. According to the curriculum, writing the unit during calculations is optional. Graying out the unit helps convey this idea while reminding students of the unit's importance (and the existence of units 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 as `Q`) object, with two constructor calling patterns:

1. `PhysicalQuantity(value, unit)`: Here, `value` can be any number, and `unit` is a string defining the unit.
2. `PhysicalQuantity(value_with_unit)`: In this form, `value_with_unit` is a string containing both the value and the unit, e.g., `'1.5 m/s'`. This format is provided for more convenient interactive use.

`stdev` (Integer): Standard deviation or uncertainty. Currently, this is cosmetic only and 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 to display. It uses Python's general (g) string formatting method to display those significant digits.

```
>>> 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 : due to string formatting, if the value is too close to 1, scientific notation may not be used, and significant digits will be ignored. Example :

```
>>> 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 scientific notation (*e*) in string formatting 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}}
```