PyLaTeX Documentation

Release 1.0.0

Jelte Fennema

March 23, 2016

Contents

PyLaTeX is a Python library for creating and compiling LaTeX files. The goal of this library is to be an easy, but extensible interface between Python and LaTeX.

PyLaTeX has two quite different usages: generating full pdfs and generating LaTeX snippets. Generating full pdfs is mostly useful when all the text that pdf should contain is generated by python, for instance exporting the data from a database. Snippets are useful when some text still needs to be written by hand, but some stuff can be automatically generated, for instance writing a report with a couple of matplotlib plots.

CHAPTER 1

Installation

PyLaTeX works on Python 2.7 and 3.3+ and it is simply installed using pip:

```
pip install pylatex
```

Some of the features require other libraries as well. This is mostly the case when converting a datatype of that library to LaTeX. For instance, generating LaTeX matrices requires Numpy. The dependencies for these extra features can simply be installed like this:

```
pip install pylatex[matrices]
```

The features that require aditional libraries are:

- matrices
- matplotlib
- quantities

CHAPTER 2

Code

To see the some code in action, please take a look at the examples/full, which generates the pdf below. To understand how the code works, please look at the *Library usage*.

1 The simple stuff

Some regular text and some *italic text.*

Also some crazy characters: $&#{}

1.1 Math that is incorrect

$$2 * 3 = 9$$

1.2 Table of something

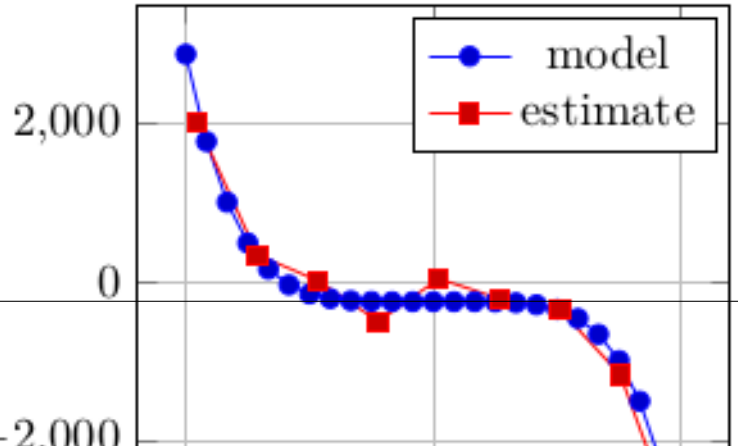
1	2	3	4
4	5	6	7

2 The fancy stuff

2.1 Correct matrix equations

$$\begin{pmatrix} 2 & 3 & 4 \\ 0 & 0 & 1 \\ 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} 100 \\ 10 \\ 20 \end{pmatrix} = \begin{pmatrix} 310 \\ 20 \\ 40 \end{pmatrix}$$

2.2 Beautiful graphs



CHAPTER 3

Support

This library is being developed in and for Python 3. Because of a conversion script the current version also works in Python 2.7. For future versions, no such promise will be made. Python 3 features that are useful but incompatible with Python 2 will be used. If you find a bug for Python 2 and it is fixable without ugly hacks feel free to send a pull request.

This library is developed for Linux. I have no intention to write fixes or test for platform specific bugs with every update, especially since I have no other operating systems to test it on. Pull requests that fix those issues are always welcome though. Issues have been fixed for Windows and it seems that compiling to pdf is currently working.

CHAPTER 4

Contributing

Read the *How to contribute* page for tips and rules when you want to contribute. To just see the source code, you should go to the Github repository.

4.1 Library usage

4.1.1 Understanding PyLaTeX

PyLaTeX is structured around two main tasks: Generating LaTeX code, and compiling LaTeX documents. The package is flexible, and can either work with your pre-existing code or generate new code with its system of classes. In turn, LaTeX code can be used to generate a document, or can be exported as-is.

4.1.2 The Classes

PyLaTeX uses a set of classes to turn LaTeX document generation into a set of pythonic components. For example, a `Document` might be comprised of `Section` objects, which in turn might have `List` objects, `Figure` objects, or custom `Command` objects.

Classes can be part of a single document, or can act as pieces on their own. With the `dumps` method, most classes can return their LaTeX-formatted code, and with the `generate_tex` method, this code can be written to a file.

Containers / Documents

A `Container` is an object that groups other LaTeX classes. Containers function like lists; they can be indexed and appended to.

One of the most important container classes is the `Document` class. Documents create a full LaTeX document that can create a PDF file with `generate_pdf` . Unless you are only generating LaTeX snippets, you will likely want to enclose your code inside a Document.

Additionally, a number of `section` containers are available, which correspond to the standard `\section` commands of LaTeX. As with documents, these can be appended to. A `Section` can further include a `Subsection` or a `Subsubsection` object.

Tables, Images, Math, etc.

PyLaTeX has a number of classes that are useful in generating difficult-to-format LaTeX code. See the API documentation and code examples for information on a specific environment.

Commands, Options, and Arguments

Although PyLaTeX has implemented many useful commands, it is easy to create a custom command with the `Command` class. Commands can be supplied with `{}` arguments or `[]` options, with either a single option as a string, or multiple options in a list.

Additionally, Options and Arguments can be placed in an `Options` object or a `Arguments` object.

Formatting Strings

A number of functions are available in `utils` that are helpful in formatting text. For example, the functions `bold` and `italic` exist to format text appropriately.

4.1.3 Extending PyLaTeX

Because of all the base classes supplied by PyLaTeX, it is very easy to extend its support in LaTeX features. Just pick one of the existing (base) classes that fits best and extend that with the needed functionality.

All LaTeX objects come from `LatexObject` , but it is probably more useful to extend one of the other base subclasses, like `Environment` or `CommandBase`. Consult the API documentation to see the variety of base classes available for use.

4.1.4 Plain LaTeX Strings

Although PyLaTeX contains classes and functions to make generating LaTeX formatted text easy, at its core it is a nice wrapper around string manipulations. This is why all of them also accept raw LaTeX strings. That way you can just use regular LaTeX strings when something is not supported directly by the library.

Unescaping Strings

Using regular LaTeX strings may not be as simple as is seems though, because by default almost all strings are escaped. This is done for security reasons and to make sure valid LaTeX code will be generated at all times. However, there are cases where raw LaTeX strings should just be used directly in the document. This is why the `NoEscape` string type exists. This is just a subclass of `str`, but it will not be escaped. One important thing to note about this class is that appending a `NoEscape` type string to a regular string results in a regular string, since one type has to be chosen and the most conservative approach is taken.

Another way to make sure strings are not escaped is by setting the `escape` attribute to `False` on the class that is the container of the string. Keep in mind though that any strings that are added to that object will not be escaped when doing this. So, only use this method for objects that don't contain possibly unsafe strings.

4.2 Examples

The examples below show some of the different uses of PyLaTeX. They show how to use some of the modules. For all the uses and modules please see the *API reference*.

4.3 API reference

This section shows all of classes and functions this library exposes. The most important thing to remember when looking at the documentation, is that this library uses subclassing extensively. That is why you should always look at the parent classes if it seems like the class you are looking at is missing methods.

4.4 Change Log

All notable changes to this project will be documented on this page. This project adheres to Semantic Versioning.

4.4.1 Unreleased - docs

See these docs for changes that have not yet been released and are only present in the development version. This version might not be stable, but to install it use:

```
pip install git+https://github.com/JelteF/PyLaTeX.git
```

Changed

- Allow overriding of the default numbering of `Section` class.
- `Parameters` now unpacks a dict as keyword arguments when passed a single dictionary as argument.

Added

- Add the `textcomp` package by default. This way some special glyphs, like the Euro (€) Symbol can be used in the source.
- `Quantity` got a new `options` keyword argument and learned to handle uncertain quantities.

Fixed

- Setting the `lmodern` keyword argument of `Document` to false will not cause invalid LaTeX code anymore.

4.4.2 1.0.0 - docs - 2015-11-25

This realease brings some great changes. The whole package has been refactored and actual documentation has been added. Because of this, things have been moved an renamed. One of the most notable changes is that all normal text is now escaped by default.

Changed

- The base_classes submodule has been split into multiple sub-submodules.
- The old baseclasses have been renamed as well. They now have easier names that better show their purpose.
- The command and parameters submodules have been merged into one command submodule in the base_classes submodule.
- The numpy classes have been moved to the math submodule.

- For all of the previous changes the old submodules and names should still work during the transition period, but they will be removed before the final release.
- The `Plt` class has been merged with the `Figure` class. Its `add_plot` method also doesn't take a plt argument anymore. The plt module is now imported when the `add_plot` method is used. This also allows for adding plots in the `SubFigure` class.
- Compiling is more secure now and it doesn't show output unless an error occurs or explicitly specified.
- The internal method `propegate_packages` has been spelled correctly and made "internal" by adding an underscore in front of the name, resulting in `_propagate_packages`
- The default allignment of a multicolumn is not `c` instead of `|c|`, since vertical lines in tables are ugly most of the time.
- Make the list method of `Parameters` a private method.
- Make the `get_table_width` function private.
- Make `width` and `placement` keyword only arguments for the `add_plot` method.
- The old `Table` class is renamed to `Tabular`. A new `Table` class has been created that represents the `table` LaTeX environment, which can be used to create a floating table.
- Fixed a bug in the `Document` class, that lead to an error if a filepath without basename was provided.
- Fixed the testall.sh script such that sphinx and nosetests get called with the correct python version.
- The graphics submodule has been renamed to figure.
- The pgfplots submodule has been renamed to tikz.
- Rename the `seperate_paragraph` keyword argument to the correctly spelled `separate_paragraph`.
- The `container_name` attribute has been changed to `latex_name` so it can be used more than containers. By default it is still the lowercase version of the classname. To change the default for a class you should set `_latex_name`
- Made `Document.select_filepath` private.
- `Container` now has a `dumps_content` method, which dumps it content instead of a dumps method. This allows to override just that method when subclassing `Environment` so you can do dump in some special inside the environment, while still keeping the `\begin` and `\end` stuff provided by `Environment`.
- When subclassing a class and special LaTeX packages are needed, you now have to specify the packages class attribute instead of passing packages along with the `__init__` method.
- Content of subclasses of `Container` is now automatically escaped. Content of `Arguments` or `Options` is not escaped by default.
- Made `separate_paragraph`, `begin_paragraph` and `end_paragraph` class attributes instead of instance attributes.
- The default of the `filepath` argument for the `Document.generate_pdf` and `Document.generate_tex` have been changed to `None`. The response to the default is not changed, so this is a fairly invisible change.
- Moved `separate_paragraph`, `begin_paragraph` and `end_paragraph` attributes to `LatexObject`.
- Use `latexmk` to compile to pdf when available, otherwise fallback to `pdflatex`.
- Change the order of arguments of the `Axis` constructor.
- Tables like `Tabular` now raise an exception when rows with wrong size are added

- Made lots of keyword arguments keyword only arguments. This was needed to make it easy to keep the API the same in the future.
- Removed the submodules `pylatex.parameters`, `pylatex.command` and `pylatex.numpy`. The content of the first two was moved to `pylatex.base_classes.command` and the content of the last one was moved to `pylatex.math`.

Removed

- The add `add_multicolumn` and `add_multirow` methods on tabular classes are removed in favor of the much more robust and easier to use `MultiRow` and `MultiColumn` classes.
- Removed unused `name` argument of the `Matrix` class.
- Removed base keyword argument of the `Package` class. `Command` should be used when changing of the base is needed.
- Removed the `title`, `author`, `date` and `maketitle` arguments from the `Document` constructor. They were from a time when it was not possible to change the preamble, which is now very easy. They are not so commonly used that they should be part of the main `Document` object.
- Removed useless list class constructor arguments for list_spec and pos. These were probably copied from the `Tabular` class.

Added

- Lots of documentation!!!!!
- A float environment base class.
- An unfinished Quantity class that can be used in conjunction with the quantitities package. https://pythonhosted.org/quantities/
- Allow supplying a mapper function to dumps_list and the add_row method for tabular like objects.
- An `extra_arguments` argument to `Command`. See docs for description.
- Add `CommandBase`, which can be easily subclassed for a command that is used more than once.
- Add `NoEscape` string class, which can be used to make sure a raw LaTeX string is not escaped.
- A `__repr__` method, so printing LaTeX objects gives more useful information now.

4.4.3 0.8.0 - 2015-05-23

Added

- List classes (enumerate, itemize, description)
- Arguments for plt.savefig
- SubFigure class for use with subcaption package
- Command line argument for ./testall.sh to supply a custom python command
- The generate_tex method is now usable in every class, this makes making snippets even easier.
- MultiColumn and MultiRow classes for generalized table layouts.

Changed

- BaseLaTeXNamedContainer now uses the name of the class as the default container_name
- The `Table` object is going to be deprecated in favor of the better named `Tabular` object. This will take a couple of releases.
- Allow the data keyword argument of containers to be a single item instead of a list. If this is the case it will be wrapped in a list on initialization.

Fixed

- Propagate packages recursively add packages of sub containers
- Make cleanup of files Windows compatible
- Filenames can be paths (`foo/bar/my_pdf`).
- Replace `filename` by `filepath` in the names of the arguments.
- Matplotlib support now uses the tmpfile module, this fixes permission issues with the badly previously badly located tmp directory.
- The temp directory is only removed in generate_pdf when cleaning is enabled

4.4.4 0.7.1 - 2015-03-21

Added

- Contributing guidelines.

Changed

- The non keyword argument for filename is now called path instead of filename to show it can also be used with paths.
- Travis now checks for Flake8 errors.

Fixed

- Fix a bug in Plt and one in fix_filename that caused an error when using them with some filenames (dots in directories and a file without an extension)

4.4.5 0.7.0 - 2015-03-17

Added

- Matplotlib support
- Quite a bit of basic docstrings

Changed

- Filenames should now be specified to the `generate_pdf`/`generate_tex` methods of document. If this is not done the `default_filename` attribute will be used.

Fixed

- Fix a lot of bugs in the `escape_latex` function

4.4.6 0.6.1 - 2015-01-11

Added

- Travis tests

Fixed

- Bug in VectorName

4.4.7 0.6 - 2015-01-07

Added

- Figure class
- Command and Parameter classes
- `with` statement support

4.4.8 0.5 - 2014-06-02

Added

- Python 2.7 support

4.4.9 0.4.2 - 2014-03-18

Added

- More table types

4.4.10 0.4.1 - 2014-01-29

Added

- Partial experimental support for multicol/multirow

Fixed

- Fix package delegation with duplicate packages

4.5 How to contribute

First of all, if anything is incorrect or something is missing on this page (or any other for that matter), please send in a pull request. It is important that setting up the development environment is as painless as possible.

4.5.1 Setting up the development environment

Unfortunately there are quite some steps involved in setting up a development environment. If you don't want to do this and know how Vagrant works, see the bottom of this section on how to use that instead.

OS specific dependencies

Some dependencies are OS specific. Ofcourse you need to have LaTeX installed, but that also comes in some different packages on most systems.

For Ubuntu and other Debian based systems:

```
sudo apt-get install python3 python3-dev virtualenv \
    texlive-pictures texlive-science texlive-latex-extra \
    imagemagick
```

Getting the source code

You need your own fork of the Github repository by using the Github fork button. You will then need to clone your version of the repo using the normal way, something like this:

```
git clone git@github.com:YourUserName/pylatex
cd pylatex
```

Make your own branch for your specific feature or fix (don't do this just on master):

```
git checkout -b your-nice-feature
```

Python environment setup

This method will use a virtual environment, this is the easiest way to get all the dependencies.

1. Create a virtualenv by running:

```
    virtualenv venv -p python3
```

2. Activate it by running (you should do this whenever you start working on your changes):

```
    . venv/bin/activate
```

3. Install all the development dependencies inside the virtual environment by running:

```
    pip install -r dev_requirements.txt
```

Vagrant support

This might be an easier way to obtain a development environment, but the script is not very well maintained and might not work anymore. If everything goes as planned Vagrant will launch and configure a small virtual machine with all necessary tools for you, so that you can start working with PyLaTeX right away.

With Vagrant already installed, you can start the virtual machine with `$ vagrant up` and then use `$ vagrant ssh` to ssh into it. Your source files will be located under `/vagrant`. To run all unit tests and build the documentation run `$ ./testall.sh -p python3 -c` from that directory.

You can download or read more about Vagrant on https://www.vagrantup.com/.

4.5.2 Some tips before starting

1. Look at the code that is already there when creating something new, for instance the classes for tables.
2. To learn how to squash commits, read this blog. Ignore the word of caution, since that only applies to main repositories on which people base their own work. You can do this when you have a couple of commits that could be merged together. This mostly happens when you have commits that fix a typo or bug you made in a pull request and you fix that in a new commit.

4.5.3 Some rules

There are two things that are needed for every pull request:

1. Run the `testall.sh` script before making a pull request to check if you didn't break anything.
2. Follow the **PEP8** style guide and make sure it passes pyflakes (this is also tested with the `testall.sh` script).

These are also tested for by Travis, but please test them yourself as well.

Depending on your type of changes some other things are needed as well.

1. If you add new arguments, function or classes, add them to `tests/args.py` without forgetting to name the arguments. That way it is easy to see when the external API is changed in the future.
2. Change docstrings when necessary. For instance when adding new arguments or changing behaviour.
3. If you fix something, add a **test** so it won't break again.
4. If your change is user facing, add it to the **changelog** so it will be mentioned in the next release. Its location is at `docs/source/changelog.rst`.
5. If you add something new, show it off with an **example**. If you don't do this, I will probably still merge your pull request, but it is always nice to have examples of features.

CHAPTER 5

Indices

- genindex
- modindex