The *pyluatex* package

Tobias Enderle
https://github.com/tndrle/PyLuaTeX

v0.7.0 (2026/04/17)

Execute Python code on the fly in your LaTeX documents

PyLuaTeX allows you to execute Python code and to include the resulting output in your LaTeX documents in a *single compilation run*. LaTeX documents must be compiled with LuaLaTeX for this to work.

1 Example

1. LaTeX document `example.tex`

```
\documentclass{article}

\usepackage{pyluatex}

\begin{python}
import math
import random

random.seed(0)

greeting = 'Hello PyLuaTeX!'
\end{python}

\newcommand{\randint}[2]{\py{random.randint(#1, #2)}}

\begin{document}
\py{greeting}

$\sqrt{371} = \py{math.sqrt(371)}$

\randint{2}{5}
\end{document}
```

2. Compile using LuaLaTeX (shell escape is required)

```
lualatex --shell-escape example.tex
```

Note: PyLuaTeX starts Python 3 using the command `python3` by default. If `python3` does not start Python 3 on your system, find the correct command and extend `\usepackage{pyluatex}` to `\usepackage[executable=<your python`

`command>]{pyluatex}`. For example,
`\usepackage[executable=python.exe]{pyluatex}`.

Security note: Running LaTeX with the `--shell-escape` option allows arbitrary code to be executed. For this reason, it is recommended to **compile trusted documents only**.

1.1 Further Examples

The folder `example` contains additional example documents:

- `beamer.tex`

  Demonstrates the use of PyLuaTeX environments and typesetting in *BEAMER* presentations. In particular, the `fragile` option for frames is highlighted.

- `data-visualization.tex`

  Demonstrates the visualization of data using *pgfplots* and *pandas*

- `matplotlib-external.tex`

  Demonstrates how *matplotlib* plots can be generated and included in a document

- `matplotlib-pgf.tex`

  Demonstrates how *matplotlib* plots can be generated and included in a document using *PGF*

- `readme-example.tex`

  The example above

- `repl.tex`

  Demonstrates how a Python console/REPL can be run and typeset

- `sessions.tex`

  Demonstrates the use of different Python sessions in a document

- `typesetting-example.tex`

  The code typesetting example below

- `typesetting-listings.tex`

  A detailed example for typesetting code and output with the *listings* package

- `typesetting-minted.tex`

  A detailed example for typesetting code and output with the *minted* package

2 Reference

2.1 Commands

Most of the following commands accept key-value options. See Options for more details.

- **`\py[<options>]{<code>}`**

  Executes (object-like) `<code>` and writes its string representation to the document.

  Allowed options: `ignoreerrors`, `quiet`, `repl`, `session`, `store`, `verbose`

Example: `\py{3 + 7}`

- **`\pyc[<options>]{<code>}`**

  Executes `<code>`. Output (e.g. from a call to `print()`) is written to the document.

  Allowed options: `ignoreerrors`, `quiet`, `repl`, `session`, `store`, `verbose`

  Examples: `\pyc{x = 5}`, `\pyc{print('hello')}`

- **`\pyfile[<options>]{<path>}`**

  Executes the Python file specified by `<path>`. Output (e.g. from a call to `print()`) is written to the document.

  Allowed options: `ignoreerrors`, `quiet`, `repl`, `session`, `store`, `verbose`

  Example: `\pyfile{main.py}`

- **`\pyif[<options>]{<test>}{<then clause>}{<else clause>}`**

  Evaluates the Python boolean expression `<test>`, and then executes either the LaTeX code in `<then clause>` or the LaTeX code in `<else clause>`.

  Allowed options: `session`, `verbose`

  Example: `\pyif{a == 1}{$a = 1$}{$a \neq 1$}`

- **`\pyoptions{<options>}`**

  Sets options globally. For more information see the Options section.

  Allowed options: `ignoreerrors`, `quiet`, `repl`, `session`, `store`, `verbose`

  Example: `\pyoptions{verbose,session=main}`

The following commands exist as shortcuts and for backward compatibility with previous versions of PyLuaTeX:

- **`\pyq[<options>]{<code>}`**

  Executes (object-like) `<code>`. Any output to the document is suppressed.

  This is equivalent to `\py[quiet,<options>]{<code>}`.

  Allowed options: `ignoreerrors`, `repl`, `session`, `store`, `verbose`

  Example: `\pyq{3 + 7}`

- **`\pycq[<options>]{<code>}`**

  Executes `<code>`. Any output to the document is suppressed.

  This is equivalent to `\pyc[quiet,<options>]{<code>}`.

  Allowed options: `ignoreerrors`, `repl`, `session`, `store`, `verbose`

  Example: `\pycq{x = 5}`

- **`\pyfileq[<options>]{<path>}`**

  Executes the Python file specified by `<path>`. Any output to the document is suppressed.

  This is equivalent to `\pyfile[quiet,<options>]{<path>}`.

  Allowed options: `ignoreerrors`, `repl`, `session`, `store`, `verbose`

  Example: `\pyfileq{main.py}`

- **\pysession{<name>}**

  Sets <name> globally as Python session for subsequent Python code. The session that is active at the beginning is default.

  This is equivalent to \pyoptions{session=<name>}.

  Example: \pysession{main}

- **\pyoption{<option>}{<value>}**

  Assigns <value> to the option <option> globally. For more information see the Options section.

  This is equivalent to \pyoptions{<option>=<value>}.

  Allowed options: ignoreerrors, quiet, repl, session, store, verbose

  Example: \pyoption{verbose}{true}

2.2 Environments

- **python**

  Executes the provided block of Python code. The environment handles characters like _, #, %, \, etc. Code on the same line as \begin{python} is ignored, i.e., code must start on the next line. If leading spaces are present, they are gobbled automatically up to the first level of indentation.

  Allowed options: ignoreerrors, quiet, repl, session, store, verbose

  Example:

```
\begin{python}
  x = 'Hello PyLuaTeX'
  print(x)
\end{python}
```

Like commands, environments accept key-value options, e.g.

```
\begin{python}[ignoreerrors]
  print(1 / 0)
\end{python}
```

The opening tag [must **directly follow** the \begin{python}. Spaces or line breaks between \begin{python} and [are not allowed.

The following environments exist as shortcuts and for backward compatibility with previous versions of PyLuaTeX:

- **pythonq**

  Same as the python environment, but any output to the document is suppressed.

  This is equivalent to \begin{python}[quiet,<options>].

  Allowed options: ignoreerrors, repl, session, store, verbose

- **pythonrepl**

Executes the provided block of Python code in an interactive console/REPL. Code and output are stored together in the output buffer and can be typeset as explained in the section Typesetting Code or as shown in the example `repl.tex` in the folder `example`.

This is equivalent to `\begin{python}[quiet,repl,<options>]`.

Allowed options: `ignoreerrors`, `session`, `store`, `verbose`

2.3 Custom Environments

You can create your own environments based on the `python`, `pythonq` and `pythonrepl` environments. However, since those are verbatim environments, you have to use the command `\PyLTVerbatimEnv` in your environment definition, e.g.

```
\NewDocumentEnvironment{custompy}{}
{\PyLTVerbatimEnv\begin{python}}
{\end{python}}
```

2.4 Options

Options marked with *package option only* are only valid as package options in `\usepackage[...]{pyluatex}`. All other options can be used as package options and throughout the document. Options can be set globally using `\pyoptions` or locally for the various environments and commands. If a value contains commas, the entire value must be enclosed in quotation marks.

- **`executable`** `string` *default:* `python3` *package option only*

  Specifies the path to the Python executable.

  Example: `\usepackage[executable=/usr/local/bin/python3]{pyluatex}`

- **`ignoreerrors`** `boolean` *default:* `false`

  By default, PyLuaTeX aborts the compilation process when Python reports an error. If the `ignoreerrors` option is set, the compilation process is not aborted.

  Alias: `ignore errors`

  Examples: `\usepackage[ignoreerrors]{pyluatex}`, `\py[ignore errors]{1 / 0}`

- **`localimports`** `boolean` *default:* `true` *package option only*

  If this option is set, the folder containing the LaTeX input file is added to the Python path. This allows local Python packages to be imported.

  Alias: `local imports`

  Example: `\usepackage[localimports=false]{pyluatex}`

- **`quiet`** `boolean` *default:* `false`

  If this option is set, any output to the document is suppressed, even if the Python code explicitly calls `print()`. This is helpful if you want to process code or output further and do your own typesetting. For an example, see the Typesetting Code section.

  Alias: `q`

  Examples: `\py[quiet]{7 + 4}`, `\py[q]{'Hello'}`

- **repl** boolean *default:* false

  If this option is set, Python code is executed in an interactive console/REPL. Code and output are stored together in the output buffer and can be typeset as explained in the section Typesetting Code or as shown in the example repl.tex in the folder example. The use of quiet together with repl is recommended.

- **session** boolean *default:* default

  Sessions provide a way to structure and separate code. Variables, function definitions, etc. of one session are only accessible by that session. This can be helpful in long documents with a lot of code.

  Alias: s

  Examples: \pyc[session=main]{x = 5}, \py[s=main]{x}

- **shutdown** choice *default:* veryveryend *package option only*

  Specifies when the Python process is shut down.

  Possible values: veryveryend, veryenddocument, off

  PyLuaTeX shuts down the Python interpreter when the compilation is done. With the option veryveryend, Python is shut down in the enddocument/end hook. With the option veryenddocument, Python is shut down in the enddocument/afteraux hook. With the option off, Python is not shut down explicitly. However, the Python process will shut down when the LuaTeX process finishes even if off is selected. Using off on Windows might lead to problems with SyncTeX, though (https://github.com/tndrle/PyLuaTeX/issues/8).

  Before v0.6.2, PyLuaTeX used the hooks \AtVeryVeryEnd and \AtVeryEndDocument of the package *atveryend*. The new hooks enddocument/end and enddocument/afteraux are equivalent to those of the *atveryend* package.

  Example: \usepackage[shutdown=veryenddocument]{pyluatex}

- **store** boolean *default:* true

  If this option is set, code and output are stored in buffers. See Typesetting Code for more details.

  Example: \pyc[store=false]{x = 5}

- **verbose** boolean *default:* false

  If this option is set, Python input and output is written to the LaTeX log file.

  Examples: \usepackage[verbose]{pyluatex}, \py[verbose]{7 + 4}

2.5 Logging from Python

```
tex.log(*objects, sep=' ', end='\n')
```

Writes objects to the LaTeX log, separated by sep and followed by end. All elements in objects are converted to strings using str(). Both sep and end must be strings.

Example:

```
\begin{python}
  tex.log('This text goes to the LaTeX log.')
```

```
\end{python}
```

3 Requirements

- LuaLaTeX
- Python 3
- Linux, macOS or Windows

4 Typesetting Code

Sometimes, in addition to having Python code executed and the output written to your document, you also want to show the code itself in your document. PyLuaTeX does not offer any commands or environments that directly typeset code. However, PyLuaTeX has a **code and output buffer** which you can use to create your own typesetting functionality. This provides a lot of flexibility for your typesetting.

After a PyLuaTeX command or environment has been executed, the corresponding Python code and output can be accessed via the Lua functions `pyluatex.get_last_code()` and `pyluatex.get_last_output()`, respectively. Both functions return a Lua table (basically an array) where each table item corresponds to a line of code or output.

A simple example for typesetting code and output using the *listings* package would be:

```
\documentclass{article}

\usepackage{pyluatex}
\usepackage{listings}
\usepackage{luacode}

\begin{luacode}
function pytypeset()
  tex.print("\\begin{lstlisting}[language=Python]")
  tex.print(pyluatex.get_last_code())
  tex.print("\\end{lstlisting}")
  tex.print("") -- ensure newline
end
\end{luacode}

\newcommand*{\pytypeset}{%
  \noindent\textbf{Input:}
  \directlua{pytypeset()}
  \textbf{Output:}
  \begin{center}
    \directlua{tex.print(pyluatex.get_last_output())}
  \end{center}
}

\begin{document}

\begin{python}[quiet]
```

```
  greeting = 'Hello PyLuaTeX!'
  print(greeting)
\end{python}
\pytypeset

\end{document}
```

Notice that we use the `python` environment with the `quiet` option, which suppresses any output. After that, the custom command `\pytypeset` is responsible for typesetting the code and its output.

Using a different code listings package like *minted*, or typesetting inline code is very easy. You can also define your own environments that combine Python code and typesetting. See the `typesetting-*.tex` examples in the `example` folder.

Use the option `repl`, to emulate an interactive Python console/REPL.

5 How It Works

PyLuaTeX runs a Python `InteractiveInterpreter` (actually several if you use different sessions) in the background for on-the-fly code execution. Python code from your LaTeX file is sent to the background interpreter through a TCP socket. This approach allows your Python code to be executed and the output to be integrated in your LaTeX file in a single compilation run. No additional processing steps are needed. No intermediate files have to be written. No placeholders have to be inserted.

6 License

LPPL 1.3c for LaTeX code and MIT license for Python and Lua code and other files.