The *pyluatex* package

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

v0.4.2 (2022/02/13)

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}
```

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

2. Compile using LuaLaTeX (shell escape is required)

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

Note: Running LaTeX with the shell escape option enabled 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

For more intricate use cases have a look at our tests in the folder `test`.

2 Installation

PyLuaTeX is available in TeX Live, MiKTeX, and on CTAN[1] as `pyluatex`.

To install PyLuaTeX in **TeX Live** run `tlmgr install pyluatex`.

[1] `https://ctan.org/pkg/pyluatex`

In **MiKTeX**, PyLuaTeX can be installed in the *MiKTeX Console*.

To install PyLuaTeX **manually**, do the following steps:

1. Locate your local *TEXMF* folder

   The location of this folder may vary. Typical defaults for TeX Live are `~/texmf` for Linux, `~/Library/texmf` for macOS, and `C:\Users\<user name>\texmf` for Windows. If you are lucky, the command `kpsewhich -var-value=TEXMFHOME` tells you the location. For MiKTeX, the folder can be found and configured in the *MiKTeX Console*.

2. Download the latest release[2] of PyLuaTeX

3. Put the downloaded files in the folder `TEXMF/tex/latex/pyluatex` (where `TEXMF` is the folder located in 1.)

   The final folder structure must be

   ```
   TEXMF/tex/latex/pyluatex/
   |-- pyluatex-interpreter.py
   |-- pyluatex-json.lua
   |-- pyluatex.lua
   |-- pyluatex.sty
   |-- ...
   ```

3 Reference

PyLuaTeX offers a simple set of options, macros and environments.

Most macros and environments are available as *quiet* versions as well. They have the suffix `q` in their name, e.g. `\pycq` or `\pyfileq`. The quiet versions suppress any output, 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.

3.1 Package Options

- `verbose`

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

  Example: `\usepackage[verbose]{pyluatex}`

- `executable`

  Specifies the path to the Python executable. (default: `python3`)

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

- `ignoreerrors`

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

  Example: `\usepackage[ignoreerrors]{pyluatex}`

[2] `https://github.com/tndrle/PyLuaTeX/releases/latest`

- `shutdown`

  Specifies when the Python process is shut down. (default: `veryveryend`)

  Options: `veryveryend`, `veryenddocument`, `off`

  PyLuaTeX uses the hooks of the package *atveryend* to shut down the Python interpreter when the compilation is done. With the option `veryveryend`, Python is shut down in the `\AtVeryVeryEnd` hook. With the option `veryenddocument`, Python is shut down in the `\AtVeryEndDocument` 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.

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

Package options (except for `executable` and `shutdown`) can be changed in the document with the `\pyoption` command, e.g. `\pyoption{verbose}{true}` or `\pyoption{ignoreerrors}{false}`.

3.2 Macros

- `\py{code}`

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

  Example: `\py{3 + 7}`

- `\pyq{code}`

  Executes (object-like) `code`. Any output is suppressed.

  Example: `\pyq{3 + 7}`

- `\pyc{code}`

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

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

- `\pycq{code}`

  Executes `code`. Any output is suppressed.

  Example: `\pycq{x = 5}`

- `\pyfile{path}`

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

  Example: `\pyfile{main.py}`

- `\pyfileq{path}`

  Executes the Python file specified by `path`. Any output is suppressed.

  Example: `\pyfileq{main.py}`

- `\pysession{session}`

  Selects `session` as Python session for subsequent Python code.

  The session that is active at the beginning is `default`.

  Example: `\pysession{main}`

- `\pyoption{option}{value}`

Assigns `value` to the package option `option` anywhere in the document. For more information consider the Package Options section.

Example: `\pyoption{verbose}{true}`

3.3 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.

  Example:

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

- `pythonq`

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

- `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 section Typesetting Code or as shown in the example `repl.tex` in the folder `example`.

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

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

4 Requirements

- LuaLaTeX
- Python 3
- Linux, macOS or Windows

Our automated tests currently use TeX Live 2021 and Python 3.7+ on Ubuntu 20.04, macOS Big Sur 11 and Windows Server 2019.

5 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 macros 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 macro 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[3] (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{pythonq}
greeting = 'Hello PyLuaTeX!'
print(greeting)
\end{pythonq}
\pytypeset

\end{document}
```

[3] https://www.lua.org/pil/2.5.html

Notice that we use the `pythonq` environment, which suppresses any output. After that, the custom macro `\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.

To emulate an interactive Python console/REPL, the `pythonrepl` environment can be used.

6 How It Works

PyLuaTeX runs a Python `InteractiveInterpreter`[4] (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.

7 License

LPPL 1.3c[5] for LaTeX code and MIT license[6] for Python and Lua code.

We use the great json.lua[7] library under the terms of the MIT license[8].

[4] https://docs.python.org/3/library/code.html#code.InteractiveInterpreter
[5] http://www.latex-project.org/lppl.txt
[6] https://opensource.org/licenses/MIT
[7] https://github.com/rxi/json.lua
[8] https://opensource.org/licenses/MIT