The pyjupyter package

Bright Bara
barabright62@gmail.com

March 11, 2026

Contents

1 Introduction

The pyjupyter package provides a lightweight environment for typesetting Python code in LaTeX documents with a visual style inspired by Jupyter notebooks.

The package combines the syntax highlighting capabilities of listings with the layout and framing features of tcolorbox. The result is a structured and readable presentation of Python code blocks suitable for:

- scientific reports
- programming assignments
- lecture notes
- reproducible research documents

A key motivation for this package is to provide a clean alternative to standard Jupyter-to-LaTeX conversions. Typical conversion tools generate extremely verbose code, rely on heavy Pygments dependencies, and clutter the preamble with numerous renewcommand definitions, making the source file difficult to read and maintain. pyjupyter avoids this complexity by offering a native, minimalist approach that remains fully compatible with standard LaTeX workflows.

2 Installation

If you are using a modern LaTeX distribution (such as TeX Live or MiKTeX), the package can be installed automatically through your package manager.

Alternatively, you can manually install the package by following these steps:

1. Download the package from: https://mirrors.ctan.org/macros/latex/contrib/pyjupyter

2. Place the file pyjupyter.sty in the working directory of your project or in your local texmf tree.

3. Run texhash (if necessary) to update your database.

3 Loading the package

Load the package in the preamble:

```
\usepackage{pyjupyter}
```

The package automatically loads the required dependencies.

4 The `jupyter` environment

The package defines a single environment called `jupyter`.

```
\begin{jupyter}[]
# Python code
print("Hello")
\end{jupyter}
```

Example

```python
# Example Python code
def square(x):
    return x ** 2

print(square(4))
```

The environment produces a framed code block with Python syntax highlighting.

5 Important notes

5.1 About the brackets

The `jupyter` environment must always be invoked with brackets immediately after `\begin{jupyter}`:

```
\begin{jupyter}[]
    ...
\end{jupyter}
```

This applies even when no options are specified.

The reason for this is that Python comments start with the character #. In TEX the character # is also used internally to denote macro parameters.

If the environment is started without the optional argument brackets and the first line of code begins with a Python comment, TEX may interpret the character incorrectly during argument parsing.

Providing the optional brackets ensures that the environment is fully initialized before the code content is processed by the `listings` engine. This prevents compilation errors when the first line of the code block is a Python comment.

5.2 About highlighting of operators and spaces

The current version of `pyjupyter` relies on the `listings` engine for syntax highlighting. Due to the way this engine parses characters, mathematical and logical operators (such

as +, -, =, *, etc.) are highlighted in **violet** only when they are surrounded by spaces.

If the code is written in a compact way (e.g., `x=5+2`), the operators will remain in the default text color (black).

- **Correct highlighting** : `x = 5 + 2`
- **No highlighting** : `x=5+2`

While this is a technical limitation of the underlying engine, it also promotes the **PEP 8** style guidelines, which recommend surrounding operators with spaces for better readability.

```
# Operators are highlighted with spaces:
a = 10
b = 5
result =  a + b
# Operators remain black without spaces:
result=a+b
```

6 Line numbering

Line numbering can be activated using the `numbered` option.

```
\begin{jupyter}[numbered]
for i in range(5):
    print(i)
\end{jupyter}
```

```
for i in range(5):
    print(i)
```

The option internally configures the settings of the `listings` engine.

7 Font and Size Configuration

By default, `pyjupyter` uses the `sourcecodepro` family at `small` size. These settings can be customized globally through package options to suit your document's aesthetic.

7.1 Font Families

You can globally customize the font for all your Jupyter blocks by passing the `font` option when loading the package. The following families are supported:

- `sourcecodepro` (default)
- `inconsolata`
- `beramono`
- `lmtt`
- `courier`
- `txtt`

7.2 Code Size

For the code size, you can specify the LaTeX font size command **without the backslash**. The most common values are:

- `tiny`, `scriptsize` or `footnotesize`
- `small` (default)
- `normalsize`

Note: While larger options like `large`, `Large`, *or* `LARGE` *are technically supported, they are generally not recommended for code listings.*

7.3 Usage Example

To use `inconsolata` with a smaller font size (`footnotesize`) throughout your entire document, write this in the preamble:

```
\usepackage[font=inconsolata, codesize=footnotesize]{pyjupyter}
```

8 Customization

8.1 Passing options to `tcolorbox`

The `jupyter` environment allows further customization by passing options to the underlying `tcolorbox` environment.

```
\begin{jupyter}[colback=blue!3,colframe=Navy,title=Example]
print("Custom box")
\end{jupyter}
```

Example

```
print("Custom box")
```

8.2 Passing options to `listings`

Advanced users may pass options directly to the `listings` engine using the key `listing options`.

```
\begin{jupyter}[listing options={basicstyle=\ttfamily\small}]
print("Custom listings configuration")
\end{jupyter}
```

```
print("Custom listings configuration")
```

9 Embedding LaTeX code

It is possible to insert LaTeX code inside a code block using the escape delimiters defined in the listings configuration.

The escape characters are two @:

```
@ ... @
```

Everything between @ delimiters is processed as normal LaTeX code.

Example:

```
\begin{jupyter}[]
x = 10
y = 20
print(x+y) @\hspace*{3cm}\color{orange}$\alpha$ Result@
\end{jupyter}
```

```
x = 10
y = 20
print(x+y)                  α Result
```

This feature allows the user to manually emphasize elements such as function or method names when they are not automatically detected by the syntax highlighting engine.

10 Syntax highlighting

Python syntax highlighting is provided by the `listings` engine. The package defines a custom style called `pyjupyter-syntax`.

The style includes:

- highlighting of Python keywords

- highlighting of operators
- colored comments
- colored strings
- support for triple-quoted strings
- automatic line breaking

11 UTF-8 characters

The package includes support for several accented UTF-8 characters when used inside listings environments.

This allows code comments written with common accented characters to compile correctly.

```
print("L'étudiant Ouyèté se débrouille en LaTeX !")
```

12 Version history

The following list summarizes the evolution of the `pyjupyter` package.

- **v1.0.0 (2026/03/06)**: Initial release.
  - Basic Jupyter-like styling using `tcolorbox` and `listings`.
  - Default syntax highlighting for Python.
  - Integration of `sourcecodepro` as the default font.
- **v1.1.0 (2026/03/10)**: Major update with enhanced flexibility.
  - Global font selection: Added the `font` package option to choose between `sourcecodepro`, `beramono`, `inconsolata`, `lmtt`, `courier`, and `txtt`.
  - Improved numbering: Redesigned the `numbered` style to ensure compatibility with any font selected in the preamble.
  - Code size customization: Introduced the `codesize` option to globally modify the font size of code blocks from the preamble.