

# Typing Python in LATEX

Senan Sekhon

April 10, 2023

This template provides environments for rendering aesthetic, professional-looking **Python** code in LATEX.

The code is rendered using the `minted` package, which is based on **Pygments**. The boxes surrounding them are created using the `tcolorbox` package. Note that this package must be loaded with the `minted` library, e.g. by adding `\tcbuselibrary{minted}` to the preamble, for these to work.

The `pythonbox` environment creates a box for Python code (just the box, not the code). The starred version `pythonbox*` creates a “soft” box with no border.

```
\begin{pythonbox}  
This is a \texttt{pythonbox}.  
\end{pythonbox}
```

This is a pythonbox.

```
\begin{pythonbox}[The title]  
This is a \texttt{pythonbox} with a title.  
\end{pythonbox}
```

The title

This is a pythonbox with a title.

```
\begin{pythonbox*}  
This is a soft \texttt{pythonbox}.  
\end{pythonbox*}
```

This is a soft pythonbox.

```
\begin{pythonbox*}[The title]  
This is a soft \texttt{pythonbox} with a title.  
\end{pythonbox*}
```

The title

This is a soft pythonbox with a title.

The `python` environment creates Python code inside a `pythonbox`. In this environment, you can directly enter Python code and `minted` will color the text accordingly. Similarly, the starred version `python*` has no border.

```
\begin{python}  
def next_two(x):  
    lst=[x+i for i in range(3)]  
    return lst  
\end{python}
```

```
def next_two(x):  
    lst=[x+i for i in range(3)]  
    return lst
```

```
\begin{python}[The function]  
def next_two(x):  
    lst=[x+i for i in range(3)]  
    return lst  
\end{python}
```

The function

```
def next_two(x):  
    lst=[x+i for i in range(3)]  
    return lst
```

```
\begin{python*}  
def next_two(x):  
    lst=[x+i for i in range(3)]  
    return lst  
\end{python*}
```

```
def next_two(x):  
    lst=[x+i for i in range(3)]  
    return lst
```

```
\begin{python*}[The function]  
def next_two(x):  
    lst=[x+i for i in range(3)]  
    return lst  
\end{python*}
```

The function

```
def next_two(x):  
    lst=[x+i for i in range(3)]  
    return lst
```

The `\begin{python}` and `\begin{python*}` boxes have three optional arguments:

- The first argument, delimited by brackets [ ], specifies the title.
- The second argument, delimited by parentheses ( ), specifies the options for the `minted` environment, i.e. the code itself. These are handled by the `minted` package.
- The third argument, delimited by braces { }, specifies the options for the box containing the code. These are handled by the `tcolorbox` package.

```
\begin{python}[How to calculate a factorial](style=one-dark){colback=gray!40!black,colframe=blue}
def factorial(n):
    product=1          # Start with $1$
    for k in range(2,n+1):  # For each $k=2,3,\ldots,n$,
        product*=k      # Multiply by $k$
    return product
\end{python}
```

### How to calculate a factorial

```
def factorial(n):
    product=1          # Start with 1
    for k in range(2,n+1):  # For each $k = 2, 3, \dots, n$,
        product*=k      # Multiply by $k$
    return product
```

In the above example, the option `style=one-dark` is an option for the `code`, while the options `colback=gray!40!black` and `colframe=blue` are options for the `box`. Note that the option `style` only specifies the style for the code itself, it does not change the background color — this has to be done separately by specifying the `tcolorbox` option `colback`<sup>1</sup>.

See <https://pygments.org/styles/> for a list of available styles. You can also use other `tcolorbox` environments<sup>2</sup>.

The pre-defined options for the code are `autogobble` (this removes any common indentation from all lines of code), `breaklines` (to allow lines that are too long to be broken) and `mathescape` (to allow `LATeX` math to be rendered in code comments). See the [documentation](#) for the `minted` package for a list of available options.

```
\begin{python}
def f(x):
    # This function takes a number $x$
    → and returns $x^2$
    return x**2
\end{python}
```

```
def f(x):
    # This function takes a number x and
    → returns $x^2$
    return x**2
```

You can also typeset Python code by itself (without the box) using the `pythoncode` environment. This is based on the `\newminted` command from the `minted` package, and contains the same pre-defined options as `python`.

```
\begin{pythoncode}
def least_square_above(x):
    return 0 if x<0 else (int(x**0.5)+1)**2
\end{pythoncode}
```

```
def least_square_above(x):
    return 0 if x<0 else (int(x**0.5)+1)**2
```

You can overwrite these options or provide additional options by using `pythoncode*`, but it is better to use the standard `minted` environment: `\begin{minted}[\langle options \rangle]{python}` ... `\end{minted}`.

```
\begin{pythoncode*}{style=manni}
def least_square_above(x):
    return 0 if x<0 else (int(x**0.5)+1)**2
\end{pythoncode*}
```

```
def least_square_above(x):
    return 0 if x<0 else (int(x**0.5)+1)**2
```

Finally, note that `LATeX` does not *run* the Python code, it only typesets it. If you want to run the code, you can paste it into [TutorialsPoint](#):

[https://www.tutorialspoint.com/online\\_python\\_compiler.php](https://www.tutorialspoint.com/online_python_compiler.php)

Or you can use any IDE, such as `PyCharm` or `Visual Studio Code`.

<sup>1</sup>This *cannot* be achieved by specifying the `minted` option `bgcolor` — doing this would only create a colored box around the code, rather than coloring the entire interior of the `tcolorbox`.

<sup>2</sup>For example, the `filingbox`, `railingbox` and `flagbox` environments from [this template](#).