The package **piton**

F. Pantigny
fpantigny@wanadoo.fr
September 19, 2022

**Abstract**

The package **piton** provides tools to typeset Python listings with syntactic highlighting by using the Lua library LPEG. It requires LuaLaTeX.

1 Presentation

The package **piton** uses the Lua library LPEG\textsuperscript{1} for parsing Python listings and typeset them with syntactic highlighting. Since it uses Lua code, it works with lualatex only (and won’t work with the other engines: latex, pdflatex and xelatex). It does not use external program and the compilation does not require \texttt{--shell-escape}. The compilation is very fast since all the parsing is done by the library LPEG, written in C.

Here is an exemple of code typeset by **piton**, with the environment \texttt{\{Piton\}}.

```python
from math import pi

def arctan(x,n=10):
    """Compute the value of arctan(x)
    n is the number of terms if the sum"
    if x < 0:
        return -arctan(-x) # recursive call
    elif x > 1:
        return pi/2 - arctan(1/x)
    (we have used that arctan(x) + arctan(1/x) = \pi/2 for x > 0)\textsuperscript{2}
    else
        s = 0
        for k in range(n):
            s += (-1)**k/(2*k+1)*x**(2*k+1)
        return s
```

2 Installation

The package **piton** is contained in two files: **piton.sty** and **piton.lua** (the LaTeX file **piton.sty** loaded by \texttt{\usepackage} will load the Lua file **piton.lua**). Both files must be in a repertory where LaTeX will be able to find them. The best way is to install them in a texmf tree.

\textsuperscript{*}This document corresponds to the version 0.6a of **piton**, at the date of 2022/09/19.

\textsuperscript{1}LPEG is a pattern-matching library for Lua, written in C, based on parsing expression grammars: http://www.inf.puc-rio.br/~roberto/lpeg/

\textsuperscript{2}This LaTeX escape has been done by beginning the comment by \texttt{##}
3 Use of the package

In order to use the package \texttt{piton}, one has only to load the package in its document with the standard command \texttt{\usepackage} and remember that the compilation must be done with \texttt{lualatex} (and no other LaTeX engine).

The package \texttt{piton} provides three tools to typeset Python code: the command \texttt{\piton{}}, the environment \texttt{\Piton{}} and the command \texttt{\PitonInputFile{}}.

- The command \texttt{\piton{}} should be used to typeset small pieces of code inside a paragraph. \textit{Caution}: That function takes in its argument \texttt{verbatim}. Therefore, it cannot be used in the argument of another command (however, it can be used within an environment).
- The environment \texttt{\Piton{}} should be used to typeset multi-lines code.
- The command \texttt{\PitonInputFile{}} is used to insert and typeset a whole external file.

It’s possible to compose comments in LaTeX by beginning with \texttt{##} (it’s a “LaTeX escape”). The characters \texttt{##} themselves won’t be printed and the spaces after \texttt{##} are removed.

4 Customization

4.1 The command \texttt{\PitonOptions{}}

The command \texttt{\PitonOptions{}} provides five keys: \texttt{gobble}, \texttt{auto-gobble}, \texttt{env-gobble}, \texttt{line-numbers} and \texttt{all-line-numbers}.

- The key \texttt{gobble} takes in as value a positive integer \texttt{n}: the first \texttt{n} characters are discarded (before the process of highlighting of the code) for each line of the environment \texttt{\Piton{}}.
- Then the key \texttt{auto-gobble} is in force, the extension \texttt{piton} computes the minimal value \texttt{n} of the number of consecutives space beginning each (non empty) line of the environment \texttt{\Piton{}} and applies \texttt{gobble} with that value of \texttt{n}.
- When the key \texttt{env-gobble} is in force, \texttt{piton} applies \texttt{gobble} with a value of \texttt{n} equal to the number of spaces before \texttt{\end{Piton}} on the last line (if that line contains only spaces).
- With the key \texttt{line-numbers}, the \texttt{non empty} lines are numbered in the environments \texttt{\Piton{}} and in the listings resulting from the use of \texttt{\PitonInputFile{}}.
- With the key \texttt{all-line-numbers}, \texttt{all} the lines are numbered, including the empty ones.

\PitonOptions{line-numbers,auto-gobble}
\begin{Piton}
\from math import pi
\def arctan(x,n=10):
  """Compute the value of arctan(x)"
  n is the number of terms if the sum"
  if x < 0:
    return -arctan(-x) # recursive call
  elif x > 1:
    return pi/2 - arctan(1/x)
    ## (on a utilisé le fait que $\arctan(x)+\arctan(1/x)=\frac{\pi}{2}$ pour $x>0$)
  else:
    s = 0
    for k in range(n):
      s += (-1)**k/(2*k+1)*x**(2*k+1)
  return s
\end{Piton}
from math import pi

def arctan(x,n=10):
    """Compute the value of arctan(x)
    n is the number of terms if the sum""
    if x < 0:
        return -arctan(-x) # recursive call
    elif x > 1:
        return pi/2 - arctan(1/x)
        (we have used that arctan(x) + arctan(1/x) = π/2 for x > 0)3
    else:
        s = 0
        for k in range(n):
            s += (-1)**k/(2*k+1)*x**(2*k+1)
        return s

4.2 The option escape-inside

The option escape-inside must be used when loading the package piton (that is to say in the instruction \usepackage). For technical reasons, it can’t be used in the command \PitonOptions. That option takes in as value two characters which will be used to delimit pieces of code which will composed in LaTeX.

In the following example, we assume that the extension piton has been loaded by the following instruction.

\usepackage[escape-inside=$$]{piton}

In the following code, which is a recursive programmation of the mathematical factorial, we decide to highlight in yellow the instruction which contains the recursive call.

\begin{Piton}
def fact(n):
    if n==0:
        return 1
    else:
        $\colorbox{yellow!50}{$return n*fact(n-1)$}$
\end{Piton}

\begin{Piton}
def fact(n):
    if n==0:
        return 1
    else:
        return n*fact(n-1)
\end{Piton}

Caution : The escape to LaTeX allowed by the characters of escape-inside is not active in the strings nor in the Python comments (however, it’s possible to have a whole Python comment composed in LaTeX by beginning it with ##).

3This LaTeX escape has been done by beginning the comment by ##
4.3 The styles

The package piton provides the command \SetPitonStyle to customize the different styles used to format the syntactic elements of the Python listings. The customizations done by that command are limited to the current TeX group.

The command \SetPitonStyle takes as argument a comma-separated list of key=value pairs. The keys are names of styles and the value are LaTeX formatting instructions.

These LaTeX instructions must be formatting instructions such as \color{...}, \bfseries, \slshape, etc. (the commands of this kind are sometimes called semi-global commands). It’s also possible to put, at the end of the list of instructions, a LaTeX command taking exactly one argument.

Here an example which changes the style used to highlight, in the definition of a Python function, the name of the function which is defined.

\SetPitonStyle
  { Name.Function = \bfseries \setlength{\fboxsep}{1pt}\colorbox{yellow!50} }

In that example, \colorbox{yellow!50} must be considered as the name of a LaTeX command which takes in exactly one argument, since, usually, it is used with the syntax \colorbox{yellow!50}{text}.

With that setting, we will have:

\begin{Verbatim}
  def cube(x) : return x * x * x
\end{Verbatim}

The different styles are described in the table 1.

4.4 Creation of new environments

Since the environment \{Piton\} has to catch its body in a special way (more or less as verbatim text), it’s not possible to construct new environments directly over the environment \{Piton\}

That’s why piton provides a command \NewPitonEnvironment. That command takes in three mandatory arguments.

That command has the same syntax as the classical environment \NewDocumentEnvironment.

With the following instruction, a new environment \{Python\} will be constructed with the same behaviour as \{Piton\}:

\NewPitonEnvironment{Python}{\begin{tcolorbox}}{\end{tcolorbox}}

If on wished to format Python code in a box in a box of tcolorbox, it’s possible to define an environment \{Python\} with the following code:

\NewPitonEnvironment{Python}{\begin{tcolorbox}}{\end{tcolorbox}}
<table>
<thead>
<tr>
<th>Style</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>the numbers</td>
</tr>
<tr>
<td>String.Short</td>
<td>the short strings (between ' or &quot;)</td>
</tr>
<tr>
<td>String.Long</td>
<td>the long strings (between ''' or &quot;&quot;&quot;&quot;) except the documentation strings</td>
</tr>
<tr>
<td>String</td>
<td>that keys sets both String.Short and String.Long</td>
</tr>
<tr>
<td>String.Doc</td>
<td>the documentation strings</td>
</tr>
<tr>
<td>String.Interpol</td>
<td>the syntactic elements of the fields of the f-strings (that is to say the characters {, } and :)</td>
</tr>
<tr>
<td>Operator</td>
<td>the following operators : != == &lt;&lt; &gt;&gt; - - + / * % = &lt; &gt; &amp; .</td>
</tr>
<tr>
<td>Operator.Word</td>
<td>the following operators : in, is, and, or et not</td>
</tr>
<tr>
<td>Name.Builtin</td>
<td>the predefined functions of Python</td>
</tr>
<tr>
<td>Name.Function</td>
<td>the name of the functions defined by the user, at the point of their definition (that is to say after the keyword def)</td>
</tr>
<tr>
<td>Name.Decorator</td>
<td>the decorators (instructions beginning by @ in the classes)</td>
</tr>
<tr>
<td>Name.Namespace</td>
<td>the name of the modules (= external libraries)</td>
</tr>
<tr>
<td>Name.Class</td>
<td>the name of the classes at the point of their definition</td>
</tr>
<tr>
<td>Exception</td>
<td>the names of the exceptions (eg: SyntaxError)</td>
</tr>
<tr>
<td>Comment</td>
<td>the comments beginning with #</td>
</tr>
<tr>
<td>LaTeX</td>
<td>the comments beginning by ## which are composed in LaTeX by piton (## is an espace sequence to LaTeX)</td>
</tr>
<tr>
<td>Keyword.Constant</td>
<td>True, False and None</td>
</tr>
<tr>
<td>Keyword</td>
<td>the following keywords : assert, break, case, continue, del, elif, else, except, exec, finally, for, from, global, if, import, lambda, non local, pass, raise, return, try, while, with, yield, yield from.</td>
</tr>
</tbody>
</table>
5 Implementation

\NeedsTeXFormat{LaTeX2e}
\ProvidesExplPackage{piton}{\myfiledate}{\myfileversion}{Highlight Python codes with LPEG on LuaLaTeX}

\msg_new:nnn { piton } { LuaLaTeX~mandatory }
{ The~package~'piton'~must~be~used~with~LuaLaTeX.\ It~won't~be~loaded. }
\sys_if_engine_luatex:F { \msg_critical:nn { piton } { LuaLaTeX~mandatory } }
\RequirePackage { luatexbase }

We define a set of keys piton/package for these options.
\keys_define:nn { piton / package }
{ escape-inside .tl_set:N = \c_@@_escape_inside_tl ,
unknown .code:n = \msg_error:nn { piton } { unknown-key-for-package }
}
\msg_new:nnn { piton } { unknown-key-for-package }
{ Unknown-key\ You-have-used-the-key-''\l_keys_key_str''-but-the-only-key-available-here-
is-the-key-‘escape-inside’.\ That-key-will-be-ignored. }
\tl_clear_new:N \c_@@_escape_inside_tl

We process the options when the package is loaded (with \usepackage).
\ProcessKeysOptions { piton / package }
\begin{group}
\cs_new_protected:Npn \@@_set_escape_char:nn #1 #2
{ \directlua { begin_escape = "#1" }
\directlua { end_escape = "#2" }
}
\cs_generate_variant:Nn \@@_set_escape_char:nn { x x }
\@@_set_escape_char:xx
{ \tl_head:V \c_@@_escape_inside_tl }
{ \tl_tail:V \c_@@_escape_inside_tl }
\endgroup

\AtBeginDocument
{ \@ifpackageloaded { xcolor }
{ \msg_fatal:nn { piton } { xcolor-not-loaded } }
}
\msg_new:nnn { piton } { xcolor-not-loaded }
{ The-package-'xcolor'-is-required-by-'piton'.\ This-error-is-fatal. }

\msg_new:nnn { piton } { piton.lua-not-found } { The file 'piton.lua' can't be found. The package 'piton' won't be loaded. }

\file_if_exist:nF { piton.lua } { \msg_critical:nn { piton } { piton.lua-not-found} }
\lua_now:e { require("piton.lua") }

The following function has not a name with the conventions of L3 because it will be used in the Lua code.
\cs_new:Npn \pitonEOL
{ \par \leavevmode \@@_print_number: }
\cs_new_protected:Npn \@@_print_number:
{ \bool_if:NT \l_@@_line_numbers_bool { \@@_actually_print_number: } }
\cs_new_protected:Npn \@@_actually_print_number:
{ \int_incr:N \l_@@_lineno_int \hbox_overlap_left:n
  { \color{gray} \footnotesize \int_to_arabic:n \l_@@_lineno_int \quad } }
\cs_new_protected:Npn \@@_define_gobble_syntax:n #1
{ \lua_now:n { define_gobble_syntax(#1) } }
\NewDocumentCommand { \piton } { v }
{ \group_begin: \ttfamily \lua_now:e { Parse(token.scan_argument()) } { #1 } \group_end: }

\int_new:N \l_@@_line_numbers_bool
\int_new:N \l_@@_all_line_numbers_bool
\int_new:N \l_@@_line_number_i:
\int_new:N \l_@@_lineno_int
\int_new:N \l_@@_gobble_int

The following counter will be used to count the lines in the code when the user requires the numbers of the lines to be printed.
\int_new:N \l_@@_lineno_int

The following integer is the number of characters to gobble on the left side of the Python listings. Of course, the initial value is 0.
\int_new:N \l_@@_gobble_int

\cs_new_protected:Npn \@@_define_gobble_syntax:n #1
{ \lua_now:n { define_gobble_syntax(#1) } }
\NewDocumentCommand { \PitonInputFile } { m } {
  \group_begin:
  \ttfamily
  \bool_if:NT \l_@@_line_numbers_bool {
    \@_actually_print_number:
    \vspace{\vspace{-\baselineskip}}
  }
  \lua_now:e { ParseFile(token.scan_argument()) } { \#1 }
  \group_end:
}

5.1 PitonOptions

\bool_new:N \l_@@_line_numbers_bool
\bool_new:N \l_@@_all_line_numbers_bool
\keys_define:nn { PitonOptions }
{
  gobble .int_set:N = \l_@@_gobble_int ,
  gobble .value_required:n = true ,
  auto-gobble .code:n = \int_set:Nn \l_@@_gobble_int { -1 } ,
  auto-gobble .value_forbidden:n = true ,
  env-gobble .code:n = \int_set:Nn \l_@@_gobble_int { -2 } ,
  env-gobble .value_forbidden:n = true ,
  line-numbers .bool_set:N = \l_@@_line_numbers_bool ,
  line-numbers .default:n = true ,
  all-line-numbers .code:n = \bool_set_true:N \l_@@_line_numbers_bool
  \bool_set_true:N \l_@@_all_line_numbers_bool ,
  all-line-numbers .value_forbidden:n = true ,
  unknown .code:n =
  \msg_error:nn { piton } { Unknown-key-for-PitonOptions }
}

\msg_new:nnn { piton } { Unknown-key-for-PitonOptions }
{
  Unknown-key \%
  The-key-'\l_keys_key_str'-is-unknown-for-\token_to_str:N \PitonOptions.-The-
  available-keys-are:-all-line-numbers,-auto-gobble,-env-gobble,-gobble-and-
  line-numbers.\%
  If-you-go-on,-that-key-will-be-ignored.
}

The argument of \PitonOptions is provided by currification.
\NewDocumentCommand \PitonOptions { }
{ \keys_set:nn { PitonOptions } }
\NewDocumentCommand { \NewPitonEnvironment } { m m m m }
{
  We construct a TeX macro which will catch as its argument all the tokens until newline + \end(Piton) with, in that newline + \end(Piton), the catcode of newline, \, \, \{ and \} equal to 12 ("other"). The latter explains why the definition of that function is a bit complicated.
\use:x
{ \cs_set_protected:Npn
  \use:c { __piton_collect_ #1 :w }
  ###1
  \c_backslash_str \c_left_brace_str \c_right_brace_str
}
{ \group_end: }
\par \addvspace { 0.5 \text{ em} }
{
\dim_set_eq:NN \parindent \c_zero_dim
\ttfamily
\bool_if:NT \l_@@_line_numbers_bool
{
\@@_actually_print_number:
\vspace{-\baselineskip}
}
\int_case:nnF \l_@@_gobble_int
{
0
Be careful: the last argument is provided by currification.
\begin{verbatim}
\{ \lua_now:e { Parse(token.scan_argument()) } \}
\{ -1 \}
\{ \lua_now:e { AutoGobbleParse(token.scan_argument()) } \}
\{ -2 \}
\{ \lua_now:e { EnvGobbleParse(token.scan_argument()) } \}
\}
\exp_args:NV \@@_define_gobble_syntax:n \l_@@_gobble_int
\lua_now:e { GobbleParse(token.scan_argument()) }
\}
\{ \#1 \}
\}
\par \addvspace { 0.5 \text{ em} }
The following \end{#1} is only for the groups and the stack of environments of LaTeX.
\end { #1 }
}

We can now define the new environment.
\begin{verbatim}
\NewDocumentEnvironment { #1 } { #2 }
{
\group_begin:
\tl_map_function:nN
\{ \ \{ \} \$ \& \# \^ \_ \% \~ \}
\char_set_catcode_other:N
\use:c { __piton_collect_ #1 :w }
\}
\AddToHook { env / #1 / begin } \char_set_catcode_other:N \^^M
\}
\end{verbatim}

\NewPitonEnvironment { Piton } { } { } { }
\NewDocumentCommand { \PitonStyle } { m } { \csname pitonStyle#1\endcsname }
\NewDocumentCommand { \SetPitonStyle } { } { \keys_set:nn { piton } }
\cs_new_protected:Npn \@@_math_scantokens:n #1
\{ \normalfont \scantextokens { $#1$ } }
\keys_define:nn { piton }
{
String.Interpol .tl_set:c = pitonStyle String.Interpol ,
String.Interpol .value_required:n = true ,
FormattingType .tl_set:c = pitonStyle FormattingType ,
FormattingType .value_required:n = true ,
Dict.Value .tl_set:c = pitonStyle Dict.Value ,
Dict.Value .value_required:n = true ,

\end{verbatim}

9
\msg_new:nnn { piton } { Unknown~key~for~SetPitonStyle } { Unknown-key-for-SetPitonStyle }

The-style-`\l_keys_key_str'-is-unknown.\`
This-key-will-be-ignored.\`

The-available-styles-are-(in-alphabetic-order):-
Comment,-
Comment.LaTeX,-
Dict.Value,-
Exception,-
InitialValues,-
Keyword,-
Keyword.Constant,-
Name.Builtin,-
Name.Class,-
Name.Decorator,-
Name.Function,-
Name.Namespace,-
\SetPitonStyle 
{
Comment = \color[HTML]{0099FF} \itshape ,
Exception = \color[HTML]{CC0000} ,
Keyword = \color[HTML]{006699} \bfseries ,
Keyword.Constant = \color[HTML]{006699} \bfseries ,
Name.Builtin = \color[HTML]{336666} ,
Name.Decorator = \color[HTML]{9999FF},
Name.Class = \color[HTML]{00AA88} \bfseries ,
Name.Function = \color[HTML]{CC00FF} ,
Name.Namespace = \color[HTML]{00CCFF} ,
Number = \color[HTML]{FF6600} ,
Operator = \color[HTML]{555555} ,
Operator.Word = \bfseries ,
String = \color[HTML]{CC3300} ,
String.Doc = \color[HTML]{CC3300} \itshape ,
String.Interpol = \color[HTML]{AA0000} ,
Comment.LaTeX = \normalfont \color[rgb]{.468,.532,.6} ,
Name.Type = \color[HTML]{336666} ,
InitialValues = \piton ,
Dict.Value = \piton ,
Post.Function = \piton ,
Interpol.Inside = \color{black}\piton ,
}