The package piton*

F. Pantigny
fpantigny@wanadoo.fr

October 23, 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[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 `--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 `{Piton}`.

```
from math import pi

def arctan(x,n=10):
    '''Compute the value of arctan(x)
       n is the number of terms of 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)[2]
    else:
        s = 0
        for k in range(n):
            s += (-1)**k/(2*k+1)*x**(2*k+1)
        return s
```

The package piton is entirely contained in the file `piton.sty`. This file may be put in the current directory or in a `texmf` tree. However, the best is to install piton with a TeX distribution such as MiKTeX, TeX Live or MacTeX.

2 Use of the package

The package piton should be loaded with the classical command `\usepackage`: `\usepackage{piton}`.

Nevertheless, we have two remarks:

*This document corresponds to the version 0.8 of piton, at the date of 2022/10/23.

[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/

[2] This LaTeX escape has been done by beginning the comment by `##`.

- the package piton uses the package xcolor (but piton does *not* load xcolor: if xcolor is not loaded before the `\begin{document}`, a fatal error will be raised).
- the package piton must be used with LuaLaTeX exclusively: if another LaTeX engine (`latex`, `pdflatex`, `xelatex`,...) is used, a fatal error will be raised.

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

- The command `\piton` should be used to typeset small pieces of code inside a paragraph. *Caution*: That fonction takes in its argument *verbatim*. Therefore, it cannot be used in the argument of another command (however, it can be used within an environment).
- The environment `{Piton}` should be used to typeset multi-lines code. For sake of customization, it's possible to define new environments similar to the environment `{Piton}` with the command `\NewPitonEnvironment`: cf. 3.4 p. 4.
- The command `\PitonInputFile` is used to insert and typeset a whole external file.

It's possible to compose comments in LaTeX by beginning them with `##` (it's a "LaTeX escape"). The characters `##` themselves won't be printed and the spaces after `##` are removed. These comments will be called "LaTeX comments" in this document.

3 Customization

3.1 The command \PitonOptions

The command `\PitonOptions` takes in as argument a comma-separated list of *key=value* pairs. The scope of the settings done by that command is the current TeX group.[3]

- The key `gobble` takes in as value a positive integer n: the first n characters are discarded (before the process of highlightning of the code) for each line of the environment `{Piton}`.
- Then the key `auto-gobble` is in force, the extension piton computes the minimal value n of the number of consecutives space beginning each (non empty) line of the environment `{Piton}` and applies `gobble` with that value of n.
- When the key `env-gobble` is in force, piton applies `gobble` with a value of n equal to the number of spaces before `\end{Piton}` on the last line (if that line contains only spaces). The name of that key comes from *environment gobble*: the effect of gobble is set by the position of the commands `\begin{Piton}` and `\end{Piton}` which delimit the current environment.
- With the key `line-numbers`, the *non empty* lines are numbered in the environments `{Piton}` and in the listings resulting from the use of `\PitonInputFile`.
- With the key `all-line-numbers`, *all* the lines are numbered, including the empty ones.
- By default, the counter of lines is set to zero at the beginning of each environment `{Piton}` or use of `\PitonInputFile`. With the key `resume`, that reinitialisation is not done.
- The key `splittable` allows pages breaks within the environments `{Piton}` and the listings produced by `\PitonInputFile`.
- The key `background-color` sets the background color of the environments `{Piton}` and the listings produced by `\PitonInputFile` (that background has a width of `\linewidth`). Even with a background color, the pages breaks are allowed, as soon as the key `splittable` is in force.[4]

[3] We remind that an LaTeX environment is, in particular, a TeX group.

[4] The environments `{Piton}` are breakable, even within a breakable environment of tcolorbox. Remind that an environment of tcolorbox included in another environment of tcolorbox is *not* breakable, even when both environments use the key `breakable` of tcolorbox.

- **New 0.8** The key `left-margin` corresponds to a margin on the left. That key may be useful in conjonction with the key `line-numbers` or the key `line-all-numbers` if one does not want the numbers in an overlapping position on the left.

```
\PitonOptions{line-numbers,auto-gobble,background-color = gray!15}
\begin{Piton}
    from math import pi

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

3.2 The key escape-inside

The key `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 thrown directly to LaTeX (and composed by 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}
```

```
def fact(n):
    if n==0:
        return 1
    else:
        return n*fact(n-1)
```

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 `##`; such comments are merely called "LaTeX comments" in this document).

3.3 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.[5]

The command `\SetPitonStyle` takes in 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}{...}`.

With that setting, we will have : `def cube(x) : return x * x * x`

The different styles are described in the table 1. The initial settings done by piton in `piton.sty` are inspired by the style manni de Pygments.[6]

3.4 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}` with the classical commands `\newenvironment` or `\NewDocumentEnvironment`.
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}{}{}{}
```

If one wishes an environment `{Python}` with takes in as optional argument (between square brackets) the keys of the command `\PitonOptions`, it's possible to program as follows:

```
\NewPitonEnvironment{Python}{O{}}{\PitonOptions{#1}}{}
```

If one wishes to format Python code in a box of tcolorbox, it's possible to define an environment `{Python}` with the following code:

[5] We remind that an LaTeX environment is, in particuilar, a TeX group.

[6] See: https://pygments.org/styles/. Remark that, by default, Pygments provides for its style manni a colored background whose color is the HTML color `#F0F3F3`.

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

4 Advanced features

4.1 Footnotes in the environments of piton

If you want to put footnotes in an environment `{Piton}` or (or, more unlikely, in a listing produced by `\PitonInputFile`), you can use a pair `\footnotemark`–`\footnotetext`.

It's also possible to extract the footnotes with the help of the package footnote or the package footnotehyper.

If piton is loaded with the option `footnote` (with `\usepackage[footnote]{piton}` or with `\PassOptionsToPackage`), the package footnote is loaded (if it is not yet loaded) and it is used to extract the footnotes.

If piton is loaded with the option `footnotehyper`, the package footnotehyper is loaded (if it is not yet loaded) ant it is used to extract footnotes.

Caution: The packages footnote and footnotehyper are incompatible. The package footnotehyper is the successor of the package footnote and should be used preferently. The package footnote has some drawbacks, in particular: it must be loaded after the package xcolor and it is not perfectly compatible with hyperref.

In this document, the package piton has been loaded with the option `footnotehyper`: see the first page of this document for an example of a footnote in an environment `{Piton}`.

5 Examples

5.1 Line numbering

By default, the numbers of the lines are composed by piton in an overlapping position on the left (by using internally the command `\llap` of LaTeX).
In order to avoid that overlapping, it's necessary to reserve a place for the number of the lines with the key `left-margin`.

```
\PitonOptions{background-color=gray!10, left-margin = 5mm, line-numbers}
\begin{Piton}
def arctan(x,n=10):
    if x < 0:
        return -arctan(-x)        ## (appel récursif)
    elif x > 1:
        return pi/2 - arctan(1/x) ## (autre appel récursif)
    else:
        return sum( (-1)**k/(2*k+1)*x**(2*k+1) for k in range(n) )
\end{Piton}
```

```
1 def arctan(x,n=10):
2     if x < 0:
3         return -arctan(-x)        (appel récursif)
4     elif x > 1:
5         return pi/2 - arctan(1/x) (autre appel récursif)
6     else:
7         return sum( (-1)**k/(2*k+1)*x**(2*k+1) for k in range(n) )
```

5.2 Formatting of the LaTeX comments

It's possible to modify the style `Comment.LaTeX` (with `\SetPitonStyle`) in order to display the LaTeX comments (which begin with `##`) aligned on the right margin.

```
\PitonOptions{background-color=gray!10}
\SetPitonStyle{Comment.LaTeX = \hfill \normalfont\color{gray}}
\begin{Piton}
def arctan(x,n=10):
    if x < 0:
        return -arctan(-x)        ## appel récursif
    elif x > 1:
        return pi/2 - arctan(1/x) ## autre appel récursif
    else:
        return sum( (-1)**k/(2*k+1)*x**(2*k+1) for k in range(n) )
\end{Piton}
```

```
def arctan(x,n=10):
    if x < 0:
        return -arctan(-x)                                    appel récursif
    elif x > 1:
        return pi/2 - arctan(1/x)                             autre appel récursif
    else:
        return sum( (-1)**k/(2*k+1)*x**(2*k+1) for k in range(n) )
```

It's also possible to display these LaTeX comments in a kind of second column by limiting the width of the Python code by an environment `{minipage}` of LaTeX.

```
\PitonOptions{background-color=gray!10}
\NewDocumentCommand{\MyLaTeXCommand}{m}{\hfill \normalfont\itshape\rlap{\quad #1}}
\SetPitonStyle{Comment.LaTeX = \MyLaTeXCommand}
\begin{minipage}{12cm}
\begin{Piton}
def arctan(x,n=10):
    if x < 0:
        return -arctan(-x)       ## appel récursif
    elif x > 1:
        return pi/2 - arctan(1/x) ## autre appel récursif
    else:
        s = 0
        for k in range(n):
            s += (-1)**k/(2*k+1)*x**(2*k+1)
        return s
\end{Piton}
\end{minipage}
```

```
def arctan(x,n=10):
    if x < 0:
        return -arctan(-x)                                    appel récursif
    elif x > 1:
        return pi/2 - arctan(1/x)                             autre appel récursif
    else:
        s = 0
        for k in range(n):
            s += (-1)**k/(2*k+1)*x**(2*k+1)
        return s
```

5.3 Notes in the listings

In order to be able to extract the notes (which are typeset with the command `\footnote`, the extension piton must be loaded with the key `footnote` or the key `footenotehyper` as explained in the section 4.1 p. 5. In this document, the extension piton has been loaded with the key `footnotehyper`.

Of course, in an environment `{Piton}`, a command `\footnote` may appear only within a LaTeX comment (which begins with `##`). It's possible to have comments which contain only that command `\footnote`. That's the case in the following example.

```
\PitonOptions{background-color=gray!10}
\begin{Piton}
def arctan(x,n=10):
    if x < 0:
        return -arctan(-x)##\footnote{First recursive call.}]
    elif x > 1:
        return pi/2 - arctan(1/x)##\footnote{Second recursive call.}
    else:
        return sum( (-1)**k/(2*k+1)*x**(2*k+1) for k in range(n) )
\end{Piton}
```

```
def arctan(x,n=10):
    if x < 0:
        return -arctan(-x)[7]
    elif x > 1:
        return pi/2 - arctan(1/x)[8]
    else:
        return sum( (-1)**k/(2*k+1)*x**(2*k+1) for k in range(n) )
```

If an environment `{Piton}` is used in an environment `{minipage}` of LaTeX, the notes are composed, of course, at the foot of the environment `{minipage}`. Recall that such `{minipage}` can't be broken by a page break.

```
\PitonOptions{background-color=gray!10}
\emphase\begin{minipage}{\linewidth}
\begin{Piton}
def arctan(x,n=10):
    if x < 0:
        return -arctan(-x)##\footnote{First recursive call.}
    elif x > 1:
        return pi/2 - arctan(1/x)##\footnote{Second recursive call.}
    else:
        return sum( (-1)**k/(2*k+1)*x**(2*k+1) for k in range(n) )
\end{Piton}
\end{minipage}
```

```
def arctan(x,n=10):
    if x < 0:
        return -arctan(-x)[a]
    elif x > 1:
        return pi/2 - arctan(1/x)[b]
    else:
        return sum( (-1)**k/(2*k+1)*x**(2*k+1) for k in range(n) )
```

[a] First recursive call.
[b] Second recursive call.

If one embed an environment `{Piton}` in an environment `{minipage}` (typically in order to limit the width of a colored background), it's necessary to embed the whole environment `{minipage}` in an environment `{savenotes}` (of footnote or footnotehyper) in order to have the footnotes composed at the bottom of the page.

[7] First recursive call.
[8] Second recursive call.

```
\PitonOptions{background-color=gray!10}
\begin{savenotes}
\begin{minipage}{13cm}
\begin{Piton}
def arctan(x,n=10):
    if x < 0:
        return -arctan(-x)##\footnote{First recursive call.}
    elif x > 1:
        return pi/2 - arctan(1/x)##\footnote{Second recursive call.}
    else:
        return sum( (-1)**k/(2*k+1)*x**(2*k+1) for k in range(n) )
\end{Piton}
\end{minipage}
\end{savenotes}
```

```
def arctan(x,n=10):
    if x < 0:
        return -arctan(-x)[9]
    elif x > 1:
        return pi/2 - arctan(1/x)[10]
    else:
        return sum( (-1)**k/(2*k+1)*x**(2*k+1) for k in range(n) )
```

5.4 An example of tuning of the styles

The graphical styles have been presented in the section 3.3, p. 4.

We present now an example of tuning of these styles adapted to the documents in black and white. We use the font *DejaVu Sans Mono*[11] specified by the command `\setmonofont` of fontspec.

```
\setmonofont[Scale=0.85]{DejaVu Sans Mono}

\SetPitonStyle
  {
    Number = ,
    String = \itshape ,
    String.Doc = \color{gray} \slshape ,
    Operator = ,
    Operator.Word = \bfseries ,
    Name.Builtin = ,
    Name.Function = \bfseries \colorbox{gray!20} ,
    Comment = \color{gray} ,
    Comment.LaTeX = \color{gray},
    Keyword = \bfseries ,
    Name.Namespace = ,
    Name.Class = ,
    Name.Type = ,
    InitialValues = \color{gray}
  }
```

```
from math import pi

def arctan(x,n=10):
    '''Compute the value of arctan(x)
       n is the number of terms if the sum'''
```

[9] First recursive call.
[10] Second recursive call.
[11] See: https://dejavu-fonts.github.io

```
if x < 0:
    return -arctan(-x) # appel récursif
elif x > 1:
    return pi/2 - arctan(1/x)
    (we have used that arctan(x) + arctan(1/x) = π/2 for x > 0)
else:
    s = 0
    for k in range(n):
        s += (-1)**k/(2*k+1)*x**(2*k+1)
    return s
```

The previous example has been composed while the key `splittable` (of `\PitonOptions`) is in force. That's why a page break may have occurred.

Table 1: Usage of the different styles

Style	Usage
`Number`	the numbers
`String.Short`	the short strings (between `'` or `"`)
`String.Long`	the long strings (between `'''` or `"""`) except the documentation strings
`String`	that keys sets both `String.Short` and `String.Long`
`String.Doc`	the documentation strings
`String.Interpol`	the syntactic elements of the fields of the f-strings (that is to say the characters `{`, `}` and `:`)
`Operator`	the following operators : `!= == << >> - ~ + / * % = < > & . | @`
`Operator.Word`	the following operators : `in`, `is`, `and`, `or` and `not`
`Name.Builtin`	the predefined functions of Python
`Name.Function`	the name of the functions defined by the user, at the point of their definition (that is to say after the keyword `def`)
`Name.Decorator`	the decorators (instructions beginning by `@` in the classes)
`Name.Namespace`	the name of the modules (= external libraries)
`Name.Class`	the name of the classes at the point of their definition
`Exception`	the names of the exceptions (eg: `SyntaxError`)
`Comment`	the comments beginning with `#`
`Comment.LaTeX`	the comments beginning by `##` which are composed in LaTeX by piton (and called merely "LaTeX comments" in this document)
`Keyword.Constant`	`True`, `False` and `None`
`Keyword`	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.`

6 Implementation

```
\NeedsTeXFormat{LaTeX2e}
\RequirePackage{l3keys2e}
\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 }
```

The boolean `\c_@@_footnotehyper_bool` will indicate if the option `footnotehyper` is used.

```
\bool_new:N \c_@@_footnotehyper_bool
```

The boolean `\c_@@_footnote_bool` will indicate if the option `footnote` is used, but quicky, it will also be set to `true` if the option `footnotehyper` is used.

```
\bool_new:N \c_@@_footnote_bool
```

We define a set of keys for the options at load-time.

```
\keys_define:nn { piton / package }
  {
    footnote .bool_set:N = \c_@@_footnote_bool ,
    footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
    escape-inside .tl_set:N = \c_@@_escape_inside_tl ,
    escape-inside .initial:n = ,
    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~keys~available~here~
    are~'escape-inside',~'footnote'~and~'footnotehyper'.~Other~keys~are~
    available~in~\token_to_str:N \PitonOptions.\\
    That~key~will~be~ignored.
  }
```

We process the options provided by the user at load-time.

```
\ProcessKeysOptions { piton / package }


\begingroup
\cs_new_protected:Npn \@@_set_escape_char:nn #1 #2
  {
    \lua_now:n { begin_escape = "#1" }
    \lua_now:n { 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


\hook_gput_code:nnn { begindocument } { . }
  {
    \@ifpackageloaded { xcolor }
      { }
```

```
      { \msg_fatal:nn { piton } { xcolor~not~loaded } }
  }
\msg_new:nnn { piton } { xcolor~not~loaded }
  {
    xcolor~not~loaded \\
    The~package~'xcolor'~is~required~by~'piton'.\\
    This~error~is~fatal.
  }
\msg_new:nnn { piton } { footnote~with~footnotehyper~package }
  {
    Footnote~forbidden.\\
    You~can't~use~the~option~'footnote'~because~the~package~
    footnotehyper~has~already~been~loaded.~
    If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
    within~the~environments~of~piton~will~be~extracted~with~the~tools~
    of~the~package~footnotehyper.\\
    If~you~go~on,~the~package~footnote~won't~be~loaded.
  }
\msg_new:nnn { piton } { footnotehyper~with~footnote~package }
  {
    You~can't~use~the~option~'footnotehyper'~because~the~package~
    footnote~has~already~been~loaded.~
    If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
    within~the~environments~of~piton~will~be~extracted~with~the~tools~
    of~the~package~footnote.\\
    If~you~go~on,~the~package~footnotehyper~won't~be~loaded.
  }

\bool_if:NT \c_@@_footnote_bool
  {
```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```
    \@ifclassloaded { beamer }
      { \bool_set_false:N \c_@@_footnote_bool }
      {
        \@ifpackageloaded { footnotehyper }
          { \@@_error:n { footnote~with~footnotehyper~package } }
          { \usepackage { footnote } }
      }
  }
\bool_if:NT \c_@@_footnotehyper_bool
  {
```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```
    \@ifclassloaded { beamer }
      { \bool_set_false:N \c_@@_footnote_bool }
      {
        \@ifpackageloaded { footnote }
          { \@@_error:n { footnotehyper~with~footnote~package } }
          { \usepackage { footnotehyper } }
        \bool_set_true:N \c_@@_footnote_bool
      }
  }
```

The flag `\c_@@_footnote_bool` is raised and so, we will only have to test `\c_@@_footnote_bool` in order to know if we have to insert an environment `{savenotes}`.

6.1 Parameters

The following token list will contains the (potential) informations to write on the `aux` (to be used in the next compilation).

```
\tl_new:N \g_@@_aux_tl
```

The following flag corresponds to the key `splittable` of `\PitonOptions`.

```
\bool_new:N \l_@@_splittable_bool
```

The following string corresponds to the key `background-color` of `\PitonOptions`.

```
\str_new:N \l_@@_background_color_str
```

We will compute the maximal width of the lines of an environment `{Piton}` in `\l_@@_width_dim`.

```
\dim_new:N \l_@@_width_dim
```

The value of that dimension as written on the `aux` file will be stored in `\l_@@_width_on_aux_dim`.

```
\dim_new:N \l_@@_width_on_aux_dim
```

We will count the environments `{Piton}` (and, in fact, also the commands `\PitonInputFile`, despite the name `\g_@@_env_int`).

```
\int_new:N \g_@@_env_int
```

The following boolean corresponds to the key `slim` of `\PitonOptions`.

```
\bool_new:N \l_@@_slim_bool
```

The following dimension corresponds to the key `left-margin` of `\PitonOptions`.

```
\dim_new:N \l_@@_left_margin_dim
```

6.2 The gobbling mechanism

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) } }
```

6.3 Treatment of a line of code

In the contents provided by Lua, each line of the Python code will be surrounded by `\@@_begin_line:` and `\@@_end_line:`.

```
\cs_set_protected:Npn \@@_begin_line: #1 \@@_end_line:
  {
    \bool_lazy_and:nnT \l_@@_splittable_bool \c_@@_footnote_bool
      { \begin { savenotes } }
```

Be careful: there is curryfication in the following lines.

```
    \bool_if:NTF \l_@@_slim_bool
      { \hbox_set:Nn \l_tmpa_box }
      {
        \str_if_empty:NTF \l_@@_background_color_str
          { \hbox_set_to_wd:Nnn \l_tmpa_box \linewidth }
          {
            \hbox_set_to_wd:Nnn \l_tmpa_box
              { \dim_eval:n { \linewidth - 0.5 em } }
          }
      }
      {
        \skip_horizontal:N \l_@@_left_margin_dim
        \bool_if:NT \l_@@_line_numbers_bool
          {
            \bool_if:NF \l_@@_all_line_numbers_bool
              { \tl_if_empty:nF { #1 } }
              \@@_print_number:
          }
        \strut
        \str_if_empty:NF \l_@@_background_color_str \space
        #1 \hfil
      }
```

We compute in `\l_@@_width_dim` the maximal width of the lines of the environments.

```
    \dim_compare:nNnT { \box_wd:N \l_tmpa_box } > \l_@@_width_dim
      { \dim_set:Nn \l_@@_width_dim { \box_wd:N \l_tmpa_box } }
    \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + 1.25 pt }
    \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + 1.25 pt }
    \tl_if_empty:NTF \l_@@_background_color_str
      { \box_use_drop:N \l_tmpa_box }
      {
        \vbox_top:n
          {
            \hbox:n
              {
                \exp_args:NV \color \l_@@_background_color_str
                \vrule height \box_ht:N \l_tmpa_box
                       depth \box_dp:N \l_tmpa_box
                       width \l_@@_width_on_aux_dim
              }
            \skip_vertical:n { - \box_ht_plus_dp:N \l_tmpa_box }
            \box_set_wd:Nn \l_tmpa_box \l_@@_width_on_aux_dim
            \box_use_drop:N \l_tmpa_box
          }
      }
    \bool_lazy_and:nnT \l_@@_splittable_bool \c_@@_footnote_bool
      { \end { savenotes } }
    \vspace { - 2.5 pt }
  }
```

6.4 PitonOptions

The following parameters correspond to the keys `line-numbers` and `all-line-numbers`.

```
\bool_new:N \l_@@_line_numbers_bool
\bool_new:N \l_@@_all_line_numbers_bool
```

The following flag corresponds to the key `resume`.

```
\bool_new:N \l_@@_resume_bool
```

Be careful! The name of the following set of keys must be considered as public! Hence, it should *not* be changed.

```
\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  ,
    resume           .bool_set:N        = \l_@@_resume_bool ,
    resume           .value_forbidden:n = true ,
    splittable       .bool_set:N        = \l_@@_splittable_bool ,
    splittable       .default:n         = true ,
    background-color .str_set:N         = \l_@@_background_color_str ,
    background-color .value_required:n  = true ,
    slim             .bool_set:N        = \l_@@_slim_bool ,
    slim             .default:n         = true ,
```

```
    left-margin      .dim_set:N         = \l_@@_left_margin_dim ,
    left-margin      .value_required: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,~
    left-margin,~line-numbers,~resume,~slim~and~splittable.\\
    If~you~go~on,~that~key~will~be~ignored.
  }
```

The argument of `\PitonOptions` is provided by curryfication.

```
\NewDocumentCommand \PitonOptions { } { \keys_set:nn { PitonOptions } }
```

6.5 The numbers of the lines

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 \g_@@_line_int

\cs_new_protected:Npn \@@_print_number:
  {
    \int_gincr:N \g_@@_line_int
    \hbox_overlap_left:n
      {
        { \color { gray } \footnotesize \int_to_arabic:n \g_@@_line_int }
        \skip_horizontal:n { 0.4 em }
      }
  }
```

6.6 The command to write on the aux file

```
\cs_new_protected:Npn \@@_write_aux:
  {
    \tl_if_empty:NF \g_@@_aux_tl
      {
        \iow_now:Nn \@mainaux { \ExplSyntaxOn }
        \iow_now:Nx \@mainaux
          {
            \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
              { \exp_not:V \g_@@_aux_tl }
          }
        \iow_now:Nn \@mainaux { \ExplSyntaxOff }
      }
    \tl_gclear:N \g_@@_aux_tl
  }

\cs_new_protected:Npn \@@_width_to_aux:
  {
    \bool_if:NT \l_@@_slim_bool
      {
        \str_if_empty:NF \l_@@_background_color_str
          {
            \tl_gput_right:Nx \g_@@_aux_tl
              {
                \dim_set:Nn \l_@@_width_on_aux_dim
```

```
                { \dim_use:N \l_@@_width_dim + 0.5 em }
            }
        }
    }
}
```

6.7 The main commands and environments for the final user

```
\NewDocumentCommand { \piton } { v }
  {
    \group_begin:
      \ttfamily
      \cs_set_protected:Npn \@@_begin_line: { }
      \cs_set_protected:Npn \@@_end_line: { }
      \lua_now:n { Parse(token.scan_argument()) } { #1 }
    \group_end:
  }
```

The command `\@@_piton:n` does *not* take in its argument verbatim.

```
\cs_new_protected:Npn \@@_piton:n #1
  {
    \group_begin:
      \cs_set_protected:Npn \@@_begin_line: { }
      \cs_set_protected:Npn \@@_end_line: { }
      \lua_now:e { Parse(token.scan_argument()) } { #1 }
    \group_end:
  }


\NewDocumentCommand { \PitonInputFile } { m }
  {
    \int_gincr:N \g_@@_env_int
    \tl_gclear:N \g_@@_aux_tl
    \tl_if_exist:cT { c_@@ _ \int_use:N \g_@@_env_int _ tl }
      { \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl } }
    \bool_if:NF \l_@@_splittable_bool
      { \bool_if:NT \c_@@_footnote_bool { \begin { savenotes } } }
    \dim_compare:nNnT \l_@@_width_on_aux_dim = \c_zero_dim
      { \dim_set_eq:NN \l_@@_width_on_aux_dim \linewidth }
    \bool_if:NF \l_@@_resume_bool { \int_gzero:N \g_@@_line_int }
    \group_begin:
      \dim_set_eq:NN \parindent \c_zero_dim
      \ttfamily
      \lua_now:e { ParseFile(token.scan_argument()) } { #1 }
    \@@_width_to_aux:
    \group_end:
    \@@_write_aux:
    \bool_if:NF \l_@@_splittable_bool
      { \bool_if:NT \c_@@_footnote_bool { \end { savenotes } } }
  }


\NewDocumentCommand { \NewPitonEnvironment } { m m m m }
  {
    \dim_zero:N \parindent
```

We construct a TeX macro which will catch as argument all the tokens until `\end{`*name_env*`}` with, in that `\end{`*name_env*`}`, the catcodes of `\`, `{` 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 { _@@_collect_ #1 :w }
          ####1
          \c_backslash_str end \c_left_brace_str #1 \c_right_brace_str
```

```
      }
          {
            \group_end:
            \mode_if_vertical:TF
              { \mode_leave_vertical: }
              \newline
```

Be careful: there is curryfication in the following code.

```
            \bool_if:NF \l_@@_splittable_bool
              {
                \bool_if:NT \c_@@_footnote_bool { \begin { savenotes } }
                \vbox_top:n
              }
              {
                \ttfamily
                \dim_zero:N \lineskip
                \int_case:nnF \l_@@_gobble_int
                  {
                    0
```

Be careful: the last argument is provided by curryfication.

```
                    { \lua_now:n { Parse(token.scan_argument()) } }
                    { -1 }
                    { \lua_now:n { AutoGobbleParse(token.scan_argument()) } }
                    { -2 }
                    { \lua_now:n { EnvGobbleParse(token.scan_argument()) } }
                  }
                  {
                     \exp_args:NV \@@_define_gobble_syntax:n \l_@@_gobble_int
                     \lua_now:n { GobbleParse(token.scan_argument()) }
                  }
                { ##1 }
                \vspace { 2.5 pt }
                \@@_width_to_aux:
              }
            \bool_if:NF \l_@@_splittable_bool
              { \bool_if:NT \c_@@_footnote_bool { \end { savenotes } } }
```

The following `\end{#1}` is only for the groups and the stack of environments of LaTeX.

```
            \end { #1 }
            \@@_write_aux:
          }
```

We can now define the new environment.

We are still in the definition of the command `\NewPitonEnvironment`...

```
    \NewDocumentEnvironment { #1 } { #2 }
      {
        #3
        \int_gincr:N \g_@@_env_int
        \tl_gclear:N \g_@@_aux_tl
        \tl_if_exist:cT { c_@@ _ \int_use:N \g_@@_env_int _ tl }
          { \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl } }
        \dim_compare:nNnT \l_@@_width_on_aux_dim = \c_zero_dim
          { \dim_set_eq:NN \l_@@_width_on_aux_dim \linewidth }
        \bool_if:NF \l_@@_resume_bool { \int_gzero:N \g_@@_line_int }
        \group_begin:
        \box_clear:N \l_tmpa_box
        \tl_map_function:nN
          { \ \\ \{ \} \$ \& \# \^ \_ \% \~ }
          \char_set_catcode_other:N
        \use:c { _@@_collect_ #1 :w }
      }
      { #4 }
```

The following code is for technical reasons. We want to change the catcode of `^^M` before catching the arguments of the new environment we are defining. Indeed, if not, we will have problems if there

is a final optional argument in our environment (if that final argument is not used by the user in an instance of the environment, a spurious space is inserted, probably because the `^^M` is converted to space).

```
    \AddToHook { env / #1 / begin } { \char_set_catcode_other:N \^^M }
  }
```

This is the end of the definition of the command `\NewPitonEnvironment`.

```
\NewPitonEnvironment { Piton } { } { } { }
```

6.8 The styles

The following command is fundamental: it will be used by the Lua code.

```
\NewDocumentCommand { \PitonStyle } { m } { \use:c { pitonStyle #1 } }
```

The following command takes in its argument by curryfication.

```
\NewDocumentCommand { \SetPitonStyle } { } { \keys_set:nn { piton / Styles } }

\cs_new_protected:Npn \@@_math_scantokens:n #1
  { \normalfont \scantextokens { $#1$ } }

\keys_define:nn { piton / Styles }
  {
    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 ,
    Name.Decorator   .tl_set:c = pitonStyle Name.Decorator ,
    Name.Decorator   .value_required:n = true ,
    Name.Function    .tl_set:c = pitonStyle Name.Function ,
    Name.Function    .value_required:n = true ,
    Keyword          .tl_set:c = pitonStyle Keyword ,
    Keyword          .value_required:n = true ,
    Keyword.Constant .tl_set:c = pitonStyle Keyword.Constant ,
    Keyword.constant .value_required:n = true ,
    String.Doc       .tl_set:c = pitonStyle String.Doc ,
    String.Doc       .value_required:n = true ,
    Interpol.Inside  .tl_set:c = pitonStyle Interpol.Inside ,
    Interpol.Inside  .value_required:n = true ,
    String.Long      .tl_set:c = pitonStyle String.Long ,
    String.Long      .value_required:n = true ,
    String.Short     .tl_set:c = pitonStyle String.Short ,
    String.Short     .value_required:n = true ,
    String           .meta:n = { String.Long = #1 , String.Short = #1 } ,
    Comment.Math     .tl_set:c = pitonStyle Comment.Math ,
    Comment.Math     .default:n = \@@_math_scantokens:n ,
    Comment.Math     .initial:n = ,
    Comment          .tl_set:c = pitonStyle Comment ,
    Comment          .value_required:n = true ,
    InitialValues    .tl_set:c = pitonStyle InitialValues ,
    InitialValues    .value_required:n = true ,
    Number           .tl_set:c = pitonStyle Number ,
    Number           .value_required:n = true ,
    Name.Namespace   .tl_set:c = pitonStyle Name.Namespace ,
    Name.Namespace   .value_required:n = true ,
    Name.Class       .tl_set:c = pitonStyle Name.Class ,
    Name.Class       .value_required:n = true ,
    Name.Builtin     .tl_set:c = pitonStyle Name.Builtin ,
    Name.Builtin     .value_required:n = true ,
    Name.Type        .tl_set:c = pitonStyle Name.Type ,
```

```
    Name.Type         .value_required:n = true ,
    Operator          .tl_set:c = pitonStyle Operator ,
    Operator          .value_required:n = true ,
    Operator.Word     .tl_set:c = pitonStyle Operator.Word ,
    Operator.Word     .value_required:n = true ,
    Post.Function     .tl_set:c = pitonStyle Post.Function ,
    Post.Function     .value_required:n = true ,
    Exception         .tl_set:c = pitonStyle Exception ,
    Exception         .value_required:n = true ,
    Comment.LaTeX     .tl_set:c = pitonStyle Comment.LaTeX ,
    Comment.LaTeX     .value_required:n = true ,
    unknown           .code:n =
      \msg_error:nn { piton } { Unknown~key~for~SetPitonStyle }
  }

\msg_new:nnn { piton } { 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,~
    Number,~
    Operator,~
    Operator.Word,~
    String,~
    String.Doc,~
    String.Long,~
    String.Short,~and~
    String.Interpol.
  }
```

6.9 The initial style

The initial style is inspired by the style "manni" of Pygments.

```
\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:n ,
    Dict.Value       = \@@_piton:n ,
    Post.Function    = \@@_piton:n ,
    Interpol.Inside  = \color{black}\@@_piton:n ,
  }
```

6.10 Security

```
\AddToHook { env / piton / begin }
   { \msg_fatal:nn { piton } { No~environment~piton } }

\msg_new:nnn { piton } { No~environment~piton }
  {
    There~is~no~environment~piton!\\
    There~is~an~environment~{Piton}~and~a~command~
    \token_to_str:N \piton\ but~there~is~no~environment~
    {piton}.~This~error~is~fatal.
  }
```

6.11 The Lua code

```
\ExplSyntaxOff
\RequirePackage{luacode}



\begin{luacode*}
local P, S, V , C , Ct, Cc, Cf = lpeg.P, lpeg.S, lpeg.V, lpeg.C, lpeg.Ct, lpeg.Cc, lpeg.Cf


--[[ By convention, a capture which provides as value a table (and not a string), provides, in fact,
a string (the first element of the table) which is a formatting LaTeX instruction (it will be
thrown back to TeX with normal catcodes (ant not ``other'' catcode for everybody).]]

local function L(string)
  return Cc ( { string } )
end

local function K(pattern, style)
  return
     L ( "{\\PitonStyle{" .. style .. "}{" )
     * C(pattern)
     * L ( "}}" )
end


--[[ The text in "escape" (between begin_escape and end_escape) is captured
and put in a table (with only one component). Indeed, we have decided that a capture
which is encapsulated in a table must be transmitted to TeX with the normal TeX catcodes.]]

local Escape = P(begin_escape)
               * Ct ( C ( ( 1 - P(end_escape) ) ^ 1 ) )
               * P(end_escape)


lpeg.locale(lpeg) -- mandatory

local alpha , digit , space , punct = lpeg.alpha , lpeg.digit , lpeg.space , lpeg.punct

-- Remember that à, â, ç, etc. are strings of length 2 (2 bytes)
local letter = alpha + P "_"
```

```
 + P "â" + P "à" + P "ç" + P "é" + P "è" + P "ê" + P "ë" + P "ï" + P "î" + P "ô" + P "û" + P "ü" +
     P "Â" + P "À" + P "Ç" + P "É" + P "È" + P "Ê" + P "Ë" + P "Ï" + P "Î" + P "Ô" + P "Û" + P "Ü"

local alphanum = letter + digit

local identifier = letter * alphanum ^ 0

local Identifier = C ( identifier )

local Space = C ( ( space - P "\r" ) ^ 1 )

local SkipSpace = C ( ( space - P "\r" ) ^ 0 )

local Punct = C ( punct )


local EOL = ( P "\r" )
            *
            (
              ( space^0 * -1 )
              +
              Cc (
                   {
                     luatexbase.catcodetables.expl ,
                     '\\__piton_end_line: \\bool_if:NT \\l__piton_splittable_bool \\newline \\__piton_begin
                   }
                 )
            )


local Number =
   K (
       ( digit^1 * P "." * digit^0 + digit^0 * P "." * digit^1 + digit^1 )
       * ( S "eE" * S "+-" ^ -1 * digit^1 ) ^ -1
       + digit^1
       , 'Number' )


local Word = C ( ( ( 1 - space ) - S "'\"\r[()]" - digit ) ^ 1 )

if begin_escape ~= ''
then  Word = C ( ( ( 1 - space - P(begin_escape) - P(end_escape) ) - S "'\"\r[()]" - digit ) ^ 1 )
end

local Delim = C ( S "[()]" )


local Keyword =
   K ( P "assert" + P "break" + P "case" + P "continue" + P "del"
       + P "elif" + P "else" + P "except" + P "exec" + P "finally" + P "for" + P "from"
       + P "global" + P "if" + P "import" + P "lambda" + P "non local"
       + P "pass" + P "return" + P "try" + P "while"
       + P "with" + P "yield" + P "yield from" ,
   'Keyword' )
   + K ( P "True" + P "False" + P "None" , 'Keyword.Constant' )


local Builtin =
   K ( P "__import__" + P "abs" + P "all" + P "any" + P "bin" + P "bool" + P "bytearray"
     + P "bytes" + P "chr" + P "classmethod" + P "compile" + P "complex" + P "delattr"
     + P "dict" + P "dir" + P "divmod" + P "enumerate" + P "eval" + P "filter"
     + P "float" + P "format" + P "frozenset" + P "getattr" + P "globals" + P "hasattr"
     + P "hash" + P "hex" + P "id" + P "input" + P "int" + P "isinstance" + P "issubclass"
```

```
    + P "iter" + P "len" + P "list" + P "locals" + P "map" + P "max" + P "memoryview" + P "min"
    + P "next" + P "object" + P "oct" + P "open" + P "ord" + P "pow" + P "print" + P "property"
    + P "range" + P "repr" + P "reversed" + P "round" + P "set" + P "setattr" + P "slice"
    + P "sorted" + P "staticmethod" + P "str" + P "sum" + P "super" + P "tuple" + P "type"
    + P "vars" + P "zip" ,
  'Name.Builtin' )


local Exception =
  K ( "ArithmeticError" + P "AssertionError" + P "AttributeError"
   + P "BaseException" + P "BufferError" + P "BytesWarning" + P "DeprecationWarning"
   + P "EOFError" + P "EnvironmentError" + P "Exception" + P "FloatingPointError"
   + P "FutureWarning" + P "GeneratorExit" + P "IOError" + P "ImportError"
   + P "ImportWarning" + P "IndentationError" + P "IndexError" + P "KeyError"
   + P "KeyboardInterrupt" + P "LookupError" + P "MemoryError" + P "NameError"
   + P "NotImplementedError" + P "OSError" + P "OverflowError"
   + P "PendingDeprecationWarning" + P "ReferenceError" + P "ResourceWarning"
   + P "RuntimeError" + P "RuntimeWarning" + P "StopIteration"
   + P "SyntaxError" + P "SyntaxWarning" + P "SystemError" + P "SystemExit"
   + P "TabError" + P "TypeError" + P "UnboundLocalError" + P "UnicodeDecodeError"
   + P "UnicodeEncodeError" + P "UnicodeError" + P "UnicodeTranslateError"
   + P "UnicodeWarning" + P "UserWarning" + P "ValueError" + P "VMSError"
   + P "Warning" + P "WindowsError" + P "ZeroDivisionError"
   + P "BlockingIOError" + P "ChildProcessError" + P "ConnectionError"
   + P "BrokenPipeError" + P "ConnectionAbortedError" + P "ConnectionRefusedError"
   + P "ConnectionResetError" + P "FileExistsError" + P "FileNotFoundError"
   + P "InterruptedError" + P "IsADirectoryError" + P "NotADirectoryError"
   + P "PermissionError" + P "ProcessLookupError" + P "TimeoutError"
   + P "StopAsyncIteration" + P "ModuleNotFoundError" + P "RecursionError" ,
  'Exception' )

local RaiseException = K ( P "raise" , 'Keyword' ) * SkipSpace * Exception * C ( P "(" )

local ExceptionInConsole = Exception *  C ( ( 1 - P "\r" ) ^ 0 ) * EOL


local Namespace =
  K ( P "from" , 'Keyword' ) * Space * K ( alphanum^1 , 'Name.Namespace' )
  *  ( Space * K ( P "import" , 'Keyword' ) ) ^ -1


local ImportAs = K ( P "import" , 'Keyword' )
                 * Space
                 * K ( identifier , 'Name.Namespace' )
                 * ( SkipSpace * C ( P "," ) * SkipSpace * K ( identifier , 'Name.Namespace' ) ) ^ 0
                 * (
                     Space * K ( P "as" , 'Keyword' ) * Space * K ( identifier , 'Name.Namespace' )
                   ) ^ 0

local Class = K ( P "class" , 'Keyword' )
              * ( Space * K ( identifier , 'Name.Class' ) ) ^ -1

local Decorator = K ( P "@" * letter^1 , 'Name.Decorator' )

local SingleShortInterpol =
    K ( P "{" , 'String.Interpol')
  * K ( ( 1 - S "}':" ) ^ 0 , 'Interpol.Inside' )
  * C ( P ":" * (1 - S "}:'") ^ 0 ) ^ -1
  * K ( P "}" , 'String.Interpol' )

local DoubleShortInterpol =
    K ( P "{" , 'String.Interpol' )
  * K ( ( 1 - S "}\":" ) ^ 0 , 'Interpol.Inside' )
```

```
  * ( K ( P ":" , 'String.Interpol' ) * C ( (1 - S "}:\"") ^ 0 ) ) ^ -1
  * K ( P "}" , 'String.Interpol' )

local SingleLongInterpol =
    K ( P "{" , 'String.Interpol' )
  * K ( ( 1 - S "}:\r" - P "'''" ) ^ 0 , 'Interpol.Inside' )
  * C ( P ":" * (1 - S "}:\r" - P "'''" ) ^ 0 ) ^ -1
  * K ( P "}" , 'String.Interpol' )

local DoubleLongInterpol =
    K ( P "{" , 'String.Interpol' )
  * K ( ( 1 - S "}:\r" - P "\"\"\"" ) ^ 0 , 'Interpol.Inside' )
  * C ( P ":" * (1 - S "}:\r" - P "\"\"\"" ) ^ 0 ) ^ -1
  * K ( P "}" , 'String.Interpol' )

local SingleShortPureString = C ( ( P "\\'" + P "{{" + P "}}" + 1 - S "{}'" ) ^ 1 )

local DoubleShortPureString = C ( ( P "\\\"" + P "{{" + P "}}" + 1 - S "{}\"" ) ^ 1 )

local SingleLongPureString = C ( ( 1 - P "'''" - S "{}'\r" ) ^ 1 )

local DoubleLongPureString = C ( ( 1 - P "\"\"\"" - S "{}\"\r" ) ^ 1 )

local SingleShortString =
    L ( "{\\PitonStyle{String.Short}{" )
     * (
           C ( P "f'" + P "F'" )
           * ( SingleShortInterpol + SingleShortPureString ) ^ 0
           * C ( P "'" )
         +
           C ( ( P "'" + P "r'" + P "R'" ) * ( P "\\'" + 1 - S "'\r" ) ^ 0 * P "'" )
       )
     * L ( "}}" )

local DoubleShortString =
    L ( "{\\PitonStyle{String.Short}{" )
     * (
           C ( P "f\"" + P "F\"" )
           * ( DoubleShortInterpol + DoubleShortPureString ) ^ 0
           * C ( P "\"" )
         +
           C ( ( P "\"" + P "r\"" + P "R\"" ) * ( P "\\\"" + 1 - S "\"\r" ) ^ 0 * P "\"" )
       )
     * L ( "}}" )


local ShortString = SingleShortString + DoubleShortString


local SingleLongString =
      L "{\\PitonStyle{String.Long}{"
       * (
             C ( S "fF" * P "'''" )
             * ( SingleLongInterpol + SingleLongPureString ) ^ 0
             * L "}}"
             * (
                 EOL
                 +
                 L "{\\PitonStyle{String.Long}{"
                 * ( SingleLongInterpol + SingleLongPureString ) ^ 0
                 * L "}}"
                 * EOL
               ) ^ 0
```

```
            * L "{\\PitonStyle{String.Long}{"
            * ( SingleLongInterpol + SingleLongPureString ) ^ 0
          +
            C ( ( S "rR" ) ^ -1  * P "'''" * ( 1 - P "'''" - P "\r" ) ^ 0 )
            * L "}}"
            * (
                L "{\\PitonStyle{String.Long}{"
                * C ( ( 1 - P "'''" - P "\r" ) ^ 0 )
                * L "}}"
                * EOL
              ) ^ 0
            * L "{\\PitonStyle{String.Long}{"
            * C ( ( 1 - P "'''" - P "\r" ) ^ 0 )
         )
       * C ( P "'''" )
       * L "}}"


local DoubleLongString =
      L "{\\PitonStyle{String.Long}{"
       * (
            C ( S "fF" * P "\"\"\"" )
            * ( DoubleLongInterpol + DoubleLongPureString ) ^ 0
            * L "}}"
            * (
                EOL
                +
                L "{\\PitonStyle{String.Long}{"
                * ( DoubleLongInterpol + DoubleLongPureString ) ^ 0
                * L "}}"
                * EOL
              ) ^ 0
            * L "{\\PitonStyle{String.Long}{"
            * ( DoubleLongInterpol + DoubleLongPureString ) ^ 0
          +
            C ( ( S "rR" ) ^ -1  * P "\"\"\"" * ( 1 - P "\"\"\"" - P "\r" ) ^ 0 )
            * L "}}"
            * (
                L "{\\PitonStyle{String.Long}{"
                * C ( ( 1 - P "\"\"\"" - P "\r" ) ^ 0 )
                * L "}}"
                * EOL
              ) ^ 0
            * L "{\\PitonStyle{String.Long}{"
            * C ( ( 1 - P "\"\"\"" - P "\r" ) ^ 0 )
         )
       * C ( P "\"\"\"" )
       * L "}}"



local LongString = SingleLongString + DoubleLongString


local Expression =
    P { "E" ,
        E = ( 1 - S "{}()[]\r," ) ^ 0
            *  (
                 (   P "{" * V "F" * P "}"
                   + P "(" * V "F" * P ")"
                   + P "[" * V "F" * P "]" ) * ( 1 - S "{}()[]\r," ) ^ 0
               ) ^ 0 ,
        F = ( 1 - S "{}()[]\r\"'" ) ^ 0
```

```
            * ( (
                    P "'" * (P "\\'" + 1 - S"'\r" )^0 * P "'"
                  + P "\"" * (P "\\\"" + 1 - S"\"\r" )^0 * P "\""
                  + P "{"  * V "F" * P "}"
                  + P "(" * V "F" * P ")"
                  + P "[" * V "F" * P "]"
                ) * ( 1 - S "{}()[]\r\"'" ) ^ 0 ) ^ 0 ,
      }

local Param = SkipSpace * K ( identifier , '' ) * SkipSpace
                    *  ( K ( P "=" * Expression , 'InitialValues' )
                         +  K ( P ":" , '' ) * SkipSpace * K ( letter^1 , 'Name.Type' ))
                + SkipSpace * K ( alphanum ^ 1 , '' ) * SkipSpace

local Params =  Param * (  K ( P "," , '' ) * Param ) ^ 0

local StringDoc = K ( P "\"\"\"" , 'String.Doc' )
                    * ( K ( (1 - P "\"\"\"" - P "\r" ) ^ 0 , 'String.Doc' ) * EOL ) ^ 0
                    * K ( ( 1 - P "\"\"\"" - P "\r" ) ^ 0 * P "\"\"\"" , 'String.Doc' )
                + K ( P "'''" , 'String.Doc' )
                    * ( K ( (1 - P "'''" - P "\r")^0 , 'String.Doc' ) * EOL ) ^ 0
                    * K ( ( 1 - P "'''" - P "\r")^0 * P "'''" , 'String.Doc' )

local CommentMath = P "$" * K ( ( 1 - S "$\r" ) ^ 1 , 'Comment.Math' ) * P "$"


local Comment = L ( "{\\PitonStyle{Comment}{" )
               * C ( P "#" ) * ( CommentMath + C ( ( 1 - S "$\r" ) ^ 1 ) ) ^ 0
               * L ( "}}" )
               * ( EOL + -1 )

local CommentLaTeX =
   P "##"
   * L "{\\PitonStyle{Comment.LaTeX}{\\ignorespaces"
   * Ct ( C ( ( 1 - P "\r" ) ^ 0 ) )
   * L "}}"
   * ( EOL + -1 )

local DefFunction =
   K ( P "def" , 'Keyword' )
   * ( Space
       * K ( identifier , 'Name.Function' )
       * ( SkipSpace * K ( P "(" , '' ) * Params * K ( P ")" , '' ) ) ^ -1
       * ( SkipSpace
            * K ( ( 1 - S ":\r" )^0 , 'Post.Function' )
            * K ( P ":" , 'Keyword' )
            * SkipSpace
            * ( EOL + CommentLaTeX + Comment )
            * SkipSpace
            * StringDoc ) ^ -1
      ) ^ -1

local ItemDict = ShortString * SkipSpace * C ( P ":" ) * K ( Expression , 'Dict.Value' )

local ItemOfSet = SkipSpace * ( ItemDict + ShortString ) * SkipSpace

local Set = C ( P "{" )
            * ItemOfSet * ( C ( P "," ) * ItemOfSet )  ^ 0
            * C ( P "}" )

local Operator = K ( P "!=" + P "==" + P "<<" + P ">>" + S "-~+/*%=<>&.@|" , 'Operator')

local OperatorWord = K ( P "in" + P "is" + P "and" + P "or" + P "not" , 'Operator.Word')
```

```
808
809 local SyntaxPython =
810       ( ( space - P "\r" ) ^0 * P "\r" ) ^ -1 *
811       (  ( space^1 * -1 )
812          + EOL
813          + Space
814          + Escape
815          + CommentLaTeX
816          + LongString
817          + Comment
818          + ExceptionInConsole
819          + Set
820          + Delim
821          + Class * ( Space + Punct + EOL )
822          + Namespace * ( Space + Punct + EOL )
823          + ImportAs
824          + RaiseException
825          + Keyword * ( Space + Punct + EOL )
826          + DefFunction
827          + ShortString
828          + Decorator * ( Space + Punct + EOL )
829          + Operator
830          + OperatorWord * ( Space + Punct + EOL )
831          + Builtin * ( Space + Punct + EOL )
832          + Identifier
833          + Number
834          + Word
835       ) ^0 * -1
836
837
838 local MinimalSyntax =
839   P { "S" ;
840       S = K ( (1 - P "\r" ) ^ 0 , '') + EOL * S
841     }
842
843
844 function Parse(code)
845   local t = Ct(SyntaxPython) : match(code)
846   tex.sprint( luatexbase.catcodetables.expl , '\\__piton_begin_line:' )
847   if t then else t = Ct(MinimalSyntax) : match(code) end
848   for i = 1 , #t do
849     if type(t[i]) == 'string'
850     then
851       tex.sprint(luatexbase.catcodetables.CatcodeTableOther, t[i])
852     else
853       tex.tprint(t[i])
854     end
855   end
856   tex.sprint( luatexbase.catcodetables.expl , '\\__piton_end_line:' )
857 end
858
859 function ParseFile(name)
860   s = ''
861   for line in io.lines(name) do s = s .. '\r' .. line end
862   Parse(s)
863 end
864
865
866 function define_gobble_syntax(n)
867   GobbleSyntax = ( 1 - P "\r" ) ^ (-n)  * C ( ( 1 - P "\r" ) ^0 )
868                * ( C ( P "\r" )
869                    * ( 1 - P "\r" ) ^ (-n)
870                    * C ( ( 1 - P "\r" ) ^0 )
```

```
                ) ^ 0
end

function GobbleParse(code)
  local t = Ct(GobbleSyntax):match(code)
  local new_code = ""
  for i = 1 , #t do
    new_code = new_code .. t[i]
  end
  Parse(new_code)
end

function add(acc,new_value)
  return acc + new_value
end




--[[ The following LPEG returns as capture the minimal number of spaces at
the beginning of the lines of code]]
AutoGobbleSyntax =
    ( space ^ 0 * P "\r" ) ^ -1
    * Cf (
           (
             ( P " " ) ^ 0 * P "\r"
             +
             Cf ( Cc(0) * ( P " " * Cc(1) ) ^ 0 , add )
             * ( 1 - P " " ) * ( 1 - P "\r" ) ^ 0 * P "\r"
           ) ^ 0
           *
           ( Cf ( Cc(0) * ( P " " * Cc(1) ) ^ 0 , add )
           * ( 1 - P " " ) * ( 1 - P "\r" ) ^ 0 ) ^ -1 ,
           math.min
         )

function AutoGobbleParse(code)
  local n = AutoGobbleSyntax:match(code)
  if n==0
  then Parse(code)
  else define_gobble_syntax(n)
       GobbleParse(code)
  end
end


--[[ The following LPEG returns as capture the number of spaces at the last line,
that is to say begin the \end{Piton} ]]

EnvGobbleSyntax =
       ( ( 1 - P "\r" ) ^ 0 * P "\r" ) ^ 0
         * Cf ( Cc(0) * ( P " " * Cc(1) ) ^ 0 , add ) * -1

function EnvGobbleParse(code)
  local n = EnvGobbleSyntax:match(code)
  if n==0
  then Parse(code)
  else define_gobble_syntax(n)
       GobbleParse(code)
  end
end
\end{luacode*}
```

7 History

Changes between versions 0.6 and 0.7

New keys `resume`, `splittable` and `background-color` in `\PitonOptions`.
The file `piton.lua` has been embedded in the file `piton.sty`. That means that the extension piton is now entirely contained in the file `piton.sty`.

Changes between versions 0.7 and 0.8

New keys `footnote` and `footnotehyper` at load-time.
New key `left-margin`.