

L'extension LaTeX `piton`^{*}

F. Pantigny
fpantigny@wanadoo.fr

23 octobre 2022

Résumé

L'extension `piton` propose des outils pour composer du code Python avec une coloration syntaxique en utilisant la bibliothèque Lua LPEG. L'extension `piton` nécessite l'emploi de LuaLaTeX.

1 Présentation

L'extension `piton` utilise la librairie Lua nommée LPEG¹ pour « parser » le code Python et le composer avec un coloriage syntaxique. Comme elle utilise du code Lua, elle fonctionne uniquement avec `lualatex` (et ne va pas fonctionner avec les autres moteurs de compilation LaTeX, que ce soit `latex`, `pdflatex` ou `xelatex`). Elle n'utilise aucun programme extérieur et la compilation ne requiert donc pas `--shell-escape`. La compilation est très rapide puisque tout le travail du parseur est fait par la librairie LPEG, écrite en C.

Voici un exemple de code Python composé avec l'environnement `{Piton}` proposé par `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) # appel récursif
    elif x > 1:
        return pi/2 - arctan(1/x)
        (on a utilisé le fait que arctan(x) + arctan(1/x) =  $\pi/2$  pour  $x > 0$ )2
    else:
        s = 0
        for k in range(n):
            s += (-1)**k/(2*k+1)*x**(2*k+1)
        return s
```

L'extension LaTeX `piton` est entièrement contenue dans le fichier `piton.sty`. Ce fichier peut être placé dans le répertoire courant ou dans une arborescence `texmf`. Le mieux reste néanmoins d'installer `piton` avec une distribution TeX comme MiKTeX, TeX Live ou MacTeX.

^{*}Ce document correspond à la version 0.8 de `piton`, à la date du 2022/10/23.

1. LPEG est une librairie de capture de motifs (*pattern-matching* en anglais) pour Lua, écrite en C, fondée sur les PEG (*parsing expression grammars*) : <http://www.inf.puc-rio.br/~roberto/lpeg/>

2. Cet échappement vers LaTeX a été obtenu en débutant par `##`.

2 Utilisation de l’extension

L’extension `piton` se charge simplement avec la commande `\usepackage : \usepackage{piton}`. On a néanmoins deux remarques à formuler :

- l’extension `piton` utilise l’extension `xcolor` (néanmoins `piton` charge pas `xcolor` : si `xcolor` n’est pas chargée avant le `\begin{document}`, une erreur fatale sera levée) ;
- l’extension `piton` n’est utilisable qu’avec LuaLaTeX : si un autre moteur de compilation (comme `latex`, `pdflatex` ou `xelatex`) est utilisé, une erreur fatale sera levée.

L’extension `piton` fournit trois outils pour composer du code Python : la commande `\piton`, l’environnement `{Piton}` et la commande `\PitonInputFile`.

- La commande `\piton` doit être utilisée pour composer de petits éléments de code à l’intérieur d’un paragraphe. *Attention* : Cette fonction prend son argument en mode *verbatim* (comme la commande `\verb`) et, de ce fait, cette fonction ne peut pas être utilisée à l’intérieur d’un argument d’une autre fonction (on peut néanmoins l’utiliser à l’intérieur d’un environnement).
- L’environnement `{Piton}` doit être utilisé pour composer des codes de plusieurs lignes. Pour les besoins de personnalisation, il est possible de définir de nouveaux environnements similaires à `{Piton}` en utilisant la commande `\NewPitonEnvironment` : cf. partie 3.4 p. 4.
- La commande `\PitonInputFile` doit être utilisée pour insérer et composer un fichier extérieur.

Il est possible de composer des commentaires en LaTeX en commençant par `##` (c’est un échappement vers LaTeX). Les caractères `##` eux-mêmes ne seront pas imprimés et les espaces qui les suivent sont supprimés. Ces commentaires seront simplement appelés « commentaires LaTeX » dans ce document.

3 Personnalisation

3.1 La commande `\PitonOptions`

La commande `\PitonOptions` prend en argument une liste de couples *clé=valeur*. La portée des réglages effectués par cette commande est le groupe TeX courant.³

- La clé `gobble` peut comme valeur un entier positif n : les n premiers caractères de chaque ligne sont alors retirés (avant formatage du code) dans les environnements `{Piton}`.
- Quand la clé `auto-gobble` est activée, l’extension `piton` détermine la valeur minimale n du nombre d’espaces successifs débutant chaque ligne (non vide) de l’environnement `{Piton}` et applique `gobble` avec cette valeur de n .
- Quand la clé `env-gobble` est activée, `piton` applique la clé `gobble` avec une valeur de n égale au nombre d’espaces sur la ligne précédant le `\end{Piton}` (si cette ligne ne comporte que des espaces).
- Avec la clé `line-numbers`, les lignes *non vides* sont numérotées dans les environnements `{Piton}` et dans les listings produits par la commande `\PitonInputFile`.
- Avec la clé `all-line-numbers`, *toutes* les lignes sont numérotées, y compris les lignes vides.
- Avec la clé `resume`, le compteur de lignes n’est pas remis à zéro comme il l’est normalement au début d’un environnement `{Piton}` ou bien au début d’un listing produit par `\PitonInputFile`.
- La clé `splittable` autorise un saut de page dans les environnements `{Piton}` et les listings produits par `\PitonInputFile`.
- La clé `background-color` fixe la couleur de fond des environnements `{Piton}` et des listings produits par `\PitonInputFile` (ce fond a une largeur égale à la valeur courante de `\linewidth`).

3. On rappelle que tout environnement LaTeX est, en particulier, un groupe.

Même avec une couleur de fond, les sauts de page sont possibles, à partir du moment où la clé `splittable` est utilisée.⁴

- **Nouveau 0.8** La clé `left-margin` fixe une marge sur la gauche. Cette clé peut être utile, en particulier, en conjonction avec l’une des clés `line-numbers` et `all-line-numbers` si on ne souhaite pas que les numéros de ligne soient dans une position en débordement sur la gauche.

```
\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 if the sum'''
    if x < 0:
      return -arctan(-x) # appel récursif
    elif x > 1:
      return pi/2 - arctan(1/x)
      ## (on a utilisé le fait que $\arctan(x)+\arctan(1/x)=\pi/2$ 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}
```

```
1 from math import pi
2 def arctan(x,n=10):
3     '''Compute the value of arctan(x)
4       n is the number of terms if the sum'''
5     if x < 0:
6         return -arctan(-x) # appel récursif
7     elif x > 1:
8         return pi/2 - arctan(1/x)
9         (on a utilisé le fait que arctan(x) + arctan(1/x) = π/2 pour x > 0)
10    else:
11        s = 0
12        for k in range(n):
13            s += (-1)**k/(2*k+1)*x**(2*k+1)
14        return s
```

3.2 L’option escape-inside

L’option `escape-inside` doit être utilisée au chargement de `piton` (c’est-à-dire dans l’instruction `\usepackage`). Pour des raisons techniques, elle ne peut pas être fixée dans `\PitonOptions`. Elle prend comme valeur deux caractères qui seront utilisés pour délimiter des parties qui seront composées en LaTeX.

Dans l’exemple suivant, on suppose que l’extension `piton` a été chargée de la manière suivante :

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

Dans le code suivant, qui est une programmation récursive de la factorielle, on décide de surligner en jaune l’instruction qui contient l’appel récursif.

4. Un environnement `{Piton}` est sécable même dans un environnement de `tcolorbox`, à partir du moment où la clé `breakable` de `tcolorbox` est utilisée. On précise cela parce que, en revanche, un environnement de `tcolorbox` inclus dans un autre environnement de `tcolorbox` n’est pas sécable, même quand les deux utilisent la clé `breakable`.

```

\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)

```

Attention : L'échappement vers LaTeX permis par les caractères de `escape-inside` n'est pas actif dans les chaînes de caractères ni dans les commentaires (pour avoir un commentaire entièrement en échappement vers LaTeX, c'est-à-dire ce qui est appelé dans ce document « commentaire LaTeX », il suffit de le faire débiter par `##`).

3.3 Les styles

L'extension `piton` fournit la commande `\SetPitonStyle` pour personnaliser les différents styles utilisés pour formater les éléments syntaxiques des listings Python. Ces personnalisations ont une portée qui correspond au groupe TeX courant.⁵

La commande `\SetPitonStyle` prend en argument une liste de couples *clé=valeur*. Les clés sont les noms des styles et les valeurs sont les instructions LaTeX de formatage correspondantes.

Ces instructions LaTeX doivent être des instructions de formatage du type de `\bfseries`, `\slshape`, `\color{...}`, etc. (les commandes de ce type sont parfois qualifiées de *semi-globales*). Il est aussi possible de mettre, *à la fin de la liste d'instructions*, une commande LaTeX prenant exactement un argument.

Voici un exemple qui change le style utilisé pour le nom d'une fonction Python, au moment de sa définition (c'est-à-dire après le mot-clé `def`).

```

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

```

Dans cet exemple, `\colorbox{yellow!50}` doit être considéré comme le nom d'une fonction LaTeX qui prend exactement un argument, puisque, habituellement, elle est utilisée avec la syntaxe `\colorbox{yellow!50}{text}`.

Avec ce réglage, on obtient : `def cube(x) : return x * x * x`

Les différents styles sont décrits dans la table 1. Les réglages initiaux effectués par `piton` dans `piton.sty` sont inspirés par le style `manni` de Pygments.⁶

3.4 Définition de nouveaux environnements

Comme l'environnement `{Piton}` a besoin d'absorber son contenu d'une manière spéciale (à peu près comme du texte verbatim), il n'est pas possible de définir de nouveaux environnements directement au-dessus de l'environnement `{Piton}` avec les commandes classiques `\newenvironment` et `\NewDocumentEnvironment`.

5. On rappelle que tout environnement LaTeX est, en particulier, un groupe.

6. Voir <https://pygments.org/styles/>. À remarquer que, par défaut, Pygments propose pour le style `manni` un fond coloré dont la couleur est la couleur HTML `#F0F3F3`.

TABLE 1 – Les styles proposés par `piton`

Style	Usage
<code>Number</code>	les nombres
<code>String.Short</code>	les chaînes de caractères courtes (entre ' ou ")
<code>String.Long</code>	les chaînes de caractères longues (entre ''' ou """) sauf les chaînes de documentation
<code>String</code>	cette clé fixe à la fois <code>String.Short</code> et <code>String.Long</code>
<code>String.Doc</code>	les chaînes de documentation
<code>String.Interpol</code>	les éléments syntaxiques des champs des f-strings (c'est-à-dire les caractères {, } et :)
<code>Operator</code>	les opérateurs suivants : <code>!=</code> <code>==</code> <code><<</code> <code>>></code> <code>-</code> <code>~</code> <code>+</code> <code>/</code> <code>*</code> <code>%</code> <code>=</code> <code><</code> <code>></code> <code>&</code> <code>.</code> <code> </code> <code>@</code>
<code>Operator.Word</code>	les opérateurs suivants : <code>in</code> , <code>is</code> , <code>and</code> , <code>or</code> et <code>not</code>
<code>Name.Builtin</code>	la plupart des fonctions prédéfinies par Python
<code>Name.Function</code>	le nom des fonctions définies par l'utilisateur <i>au moment de leur définition</i> , c'est-à-dire après le mot-clé <code>def</code>
<code>Name.Decorator</code>	les décorateurs (instructions débutant par <code>@</code> dans les classes)
<code>Name.Namespace</code>	le nom des modules (= bibliothèques extérieures)
<code>Name.Class</code>	le nom des classes au moment de leur définition
<code>Exception</code>	le nom des exceptions prédéfinies (eg : <code>SyntaxError</code>)
<code>Comment</code>	les commentaires commençant par <code>#</code>
<code>Comment.LaTeX</code>	les commentaires commençant par <code>##</code> qui sont composés en LaTeX par <code>piton</code> (et appelés simplement « commentaires LaTeX » dans ce document)
<code>Keyword.Constant</code>	<code>True</code> , <code>False</code> et <code>None</code>
<code>Keyword</code>	les mots-clés suivants : <code>assert</code> , <code>break</code> , <code>case</code> , <code>continue</code> , <code>del</code> , <code>elif</code> , <code>else</code> , <code>except</code> , <code>exec</code> , <code>finally</code> , <code>for</code> , <code>from</code> , <code>global</code> , <code>if</code> , <code>import</code> , <code>lambda</code> , <code>non local</code> , <code>pass</code> , <code>raise</code> , <code>return</code> , <code>try</code> , <code>while</code> , <code>with</code> , <code>yield</code> , <code>yield from</code> .

C'est pourquoi `piton` propose une commande `\NewPitonEnvironment`. Cette commande a la même syntaxe que la commande classique `\NewDocumentEnvironment`.

Par exemple, avec l'instruction suivante, un nouvel environnement `{Python}` sera défini avec le même comportement que l'environnement `{Piton}` :

```
\NewPitonEnvironment{Python}{-}{-}
```

Si on souhaite un environnement `{Python}` qui prenne en argument optionnel entre crochets les clés de `\PitonOptions`, on peut écrire :

```
\NewPitonEnvironment{Python}{0}{\PitonOptions{#1}}
```

Si on souhaite un environnement `{Python}` qui compose le code inclus dans une boîte de `tcolorbox`, on peut écrire :

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

4 Fonctionnalités avancées

4.1 Notes de pied de page dans les environnements de `piton`

Si vous voulez mettre des notes de pied de page dans un environnement de `piton` (ou bien dans un listing produit par `\PitonInputFile`, bien que cela paraisse moins pertinent dans ce cas-là) vous pouvez utiliser une paire `\footnotemark–\footnotetext`.

Il est aussi possible d'extraire les notes de pieds de page avec l'extension `footnote` ou bien l'extension `footnotehyper`.

Si `piton` est chargée avec l'option `footnote` (avec `\usepackage{footnote}{piton}`) l'extension `footnote` est chargée (si elle ne l'est pas déjà) et elle est utilisée pour extraire les notes de pied de page.

Si `piton` est chargée avec l'option `footnotehyper`, l'extension `footnotehyper` est chargée (si elle ne l'est pas déjà) et elle est utilisée pour extraire les notes de pied de page.

Attention : Les extensions `footnote` et `footnotehyper` sont incompatibles. L'extension `footnotehyper` est le successeur de l'extension `footnote` et devrait être utilisée préférentiellement. L'extension `footnote` a quelques défauts ; en particulier, elle doit être chargée après l'extension `xcolor` et elle n'est pas parfaitement compatible avec `hyperref`.

Dans ce document, l'extension `piton` a été chargée avec l'option `footnotehyper` et c'est pourquoi des notes peuvent être mises dans les environnements `{Piton}` : cf. première page de ce document.

5 Exemples

5.1 Numérotation des lignes

Par défaut, les numéros de ligne sont composés par `piton` en débordement à gauche (en utilisant en interne la commande `\llap` de LaTeX).

Si on ne veut pas de débordement, il faut réserver une place à gauche pour les numéros de lignes avec la clé `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 Formatage des commentaires LaTeX

On peut modifier le style `Comment.LaTeX` (avec `\SetPitonStyle`) pour faire afficher les commentaires LaTeX (qui débutent par `##`) en butée à droite.

```
\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)
    elif x > 1:
        return pi/2 - arctan(1/x)
    else:
        return sum( (-1)**k/(2*k+1)*x**(2*k+1) for k in range(n) )
```

appel récursif

autre appel récursif

On peut aussi faire afficher les commentaires dans une deuxième colonne à droite si on limite la largeur du code proprement dit avec un environnement `{minipage}` de 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)
    elif x > 1:
        return pi/2 - arctan(1/x)
    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)
    elif x > 1:
        return pi/2 - arctan(1/x)
    else:
        s = 0
        for k in range(n):
            s += (-1)**k/(2*k+1)*x**(2*k+1)
        return s
```

appel récursif

autre appel récursif

5.3 Notes dans les listings

Pour pouvoir extraire les notes (introduites par `\footnote`), l'extension `piton` doit être chargée, soit avec la clé `footnote`, soit avec la clé `footnotehyper`, comme expliqué à la section 4.1 p. 5. Dans le présent document, l'extension `piton` a été chargée par la clé `footnotehyper`.

Bien entendu, une commande `\footnote` ne peut apparaître que dans un commentaire LaTeX (qui débute par `##`). Un tel commentaire peut se limiter à cette unique commande `\footnote`, comme dans l'exemple suivant.

```
\PitonOptions{background-color=gray!10}
\begin{Piton}
def arctan(x,n=10):
    if x < 0:
        return -arctan(-x)##\footnote{Un premier appel récursif.}]
    elif x > 1:
        return pi/2 - arctan(1/x)##\footnote{Un deuxième 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)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) )
```

Si on utilise l'environnement `{Piton}` dans un environnement `{minipage}` de LaTeX, les notes sont, bien entendu, composées au bas de l'environnement `{minipage}`. Rappelons qu'une telle `{minipage}` ne peut être coupée par un saut de page.

```
\PitonOptions{background-color=gray!10}
\emphase\begin{minipage}{\linewidth}
\begin{Piton}
def arctan(x,n=10):
    if x < 0:
        return -arctan(-x)##\footnote{Un premier appel récursif.}
    elif x > 1:
        return pi/2 - arctan(1/x)##\footnote{Un deuxième appel récursif.}
    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.* Un premier appel récursif.
b. Un deuxième appel récursif.

Si on encapsule l'environnement `{Piton}` dans un environnement `{minipage}` pour, typiquement, limiter la largeur d'un fond coloré, il faut encadrer l'ensemble dans un environnement `{savenotes}` (de `footnote` ou `footnotehyper`) si on veut avoir les notes reportées en pied de page.

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

-
7. Un premier appel récursif.
8. Un deuxième appel récursif.


```
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 Un exemple de réglage des styles

Les styles graphiques ont été présentés à la partie 3.3, p. 4.

On présente ici un réglage de ces styles adapté pour les documents en noir et blanc. On l'utilise avec la fonte *DejaVu Sans Mono*¹¹ spécifiée avec la commande `\setmonofont` de `fontspec`.

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

\SetPitonStyle
{
    Number = ,
    String = \itshape ,
    String.Doc = \color{gray} \itshape ,
    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'''
    if x < 0:
        return -arctan(-x) # appel récursif
    elif x > 1:
        return pi/2 - arctan(1/x)
        (on a utilisé le fait que arctan(x) + arctan(1/x) =  $\pi/2$  pour  $x > 0$ )
    else:
        s = 0
        for k in range(n):
            s += (-1)**k/(2*k+1)*x**(2*k+1)
        return s
```

9. Un premier appel récursif.

10. Un deuxième appel récursif.

11. Voir : <https://dejavu-fonts.github.io>

Autre documentation

Le document `piton.pdf` (fourni avec l'extension `piton`) contient une traduction anglaise de la documentation ici présente, ainsi que le code source commenté et un historique des versions.