L'extension LaTeX piton *

F. Pantigny
fpantigny@wanadoo.fr

29 novembre 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 LPEG [1] 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 mathematical value of arctan(x)

    n is the number of terms in 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) = π/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 0.99 de piton, à la date du 2022/11/29.

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

2.1 Chargement 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.

2.2 Les commandes et environnements à la disposition de l'utilisateur

L'extension piton fournit plusieurs outils pour composer du code Python : les commandes `\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. Par exemple :

```
\piton{def carré(x): return x*x}
```
```
def carré(x): return x*x
```

La syntaxe et les particularités de la commande sont détaillées ci-après.

— L'environnement `{Piton}` doit être utilisé pour composer des codes de plusieurs lignes. Comme cet environnement prend son argument selon un mode verbatim, il ne peut pas être utilisé dans l'argument d'une commande LaTeX. Pour les besoins de personnalisation, il est possible de définir de nouveaux environnements similaires à `{Piton}` en utilisant la commande `\NewPitonEnvironment` : cf. partie 3.3 p. 6.

— La commande `\PitonInputFile` doit être utilisée pour insérer et composer un fichier extérieur. Cette commande prend en argument optionnel entre crochets deux clés `first-line` et `last-line` qui permettent de n'insérer que la partie du fichier comprise entre les lignes correspondantes.

2.3 La syntaxe de la commande \piton

La commande `\piton` possède en fait une syntaxe double. Elle est peut être utilisée comme une commande standard de LaTeX prenant son argument entre accolades (`\piton{...}`), ou bien selon la syntaxe de la commande `\verb` où l'argument est délimité entre deux caractères identiques (par ex. : `\piton|...|`). On détaille maintenant ces deux syntaxes.

— Syntaxe `\piton{...}`

Quant son argument est donné entre accolades, la commande `\piton` ne prend pas son argument en mode verbatim. Les points suivants doivent être remarqués :

— plusieurs espaces successifs sont remplacés par un unique espace ;

— il n'est pas possible d'utiliser le caractère `%` à l'intérieur ;

— les accolades doivent apparaître par paires correctement imbriquées ;

— les commandes LaTeX (celles commençant par une contre-oblique `\` mais également les caractères actifs) sont complètement développées (mais non exécutées).

Un mécanisme d'échappement est fourni : les commandes `\\`, `\%`, `\{` et `\}` insèrent les caractères correspondants `\`, `%`, `{` et `}`. Ces deux dernières commandes ne sont nécessaires que si on a besoin d'insérer des accolades non équilibrées.

Les autres caractères (y compris `#`, `^`, `_`, `&`, `$` et `@`) doivent être insérés sans contre-oblique.

Exemples :

```
\piton{ma_chaîne = '\\n'}
\piton{def pair(n): return n\%2==0}
\piton{c="#"    # une affectation }
\piton{my_dict = {'a': 3, 'b': 4}}
```
```
ma_chaîne = '\n'
def pair(n): return n%2==0
c="#" # une affectation
my_dict = {'a': 3, 'b': 4}
```

La commande `\piton` avec son argument entre accolades peut être utilisée dans les arguments des autres commandes LaTeX. [3]

— Syntaxe `\piton|...|`

Quand la commande `\piton` prend son argument entre deux caractères identiques, cet argument est pris *en mode verbatim*. De ce fait, avec cette syntaxe, la commande `\piton` ne peut *pas* être utilisée dans l'argument d'une autre fonction.

Exemples :

```
\piton|ma_chaîne = '\n'|
\piton!def pair(n): return n%2==0!
\piton+c="#"    # une affectation +
\piton?my_dict = {'a': 3, 'b': 4}?
```

```
ma_chaîne = '\n'
def pair(n): return n%2==0
c="#"      # une affectation
my_dict = {'a': 3, 'b': 4}
```

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. [4]

— La clé `gobble` prend comme valeur un entier positif n : les n premiers caractères de chaque ligne sont alors retirés (avant formatage du code) dans les environnements `{Piton}`. Ces n caractères ne sont pas nécessairement des espaces.

— 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 analyse la dernière ligne de l'environnement, c'est-à-dire celle qui contient le `\end{Piton}` et détermine si cette ligne ne comporte que des espaces suivis par `\end{Piton}`. Si c'est le cas, piton calcule le nombre n de ces espaces et applique `gobble` avec cette valeur de n. Le nom de cette clé vient de *environment gobble* : le nombre d'espaces à retirer ne dépend que de la position des délimiteurs `\begin{Piton}` et `\end{Piton}` de l'environnement.

— Avec la clé `line-numbers`, les lignes *non vides* (et toutes les lignes des *docstrings*, y compris celles qui sont 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.

— 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.

Il est possible de donner à la clé `left-margin` la valeur spéciale `auto`. Avec cette valeur, une marge est insérée automatiquement pour les numéros de ligne quand l'une des clés `line-numbers` ou `all-line-numbers` est utilisée. Voir un exemple à la partie 5.1 p. 9.

— 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`. Cela permet de poursuivre la numérotation d'un environnement à l'autre.

— 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. La commande `\piton` peut par exemple être utilisée dans une note de bas de page. Exemple : `s = 'Une chaîne'`.
4. On rappelle que tout environnement LaTeX est, en particulier, un groupe.

— En activant la clé **show-spaces**, les espaces dans les chaînes courtes (c'est-à-dire celles délimitées par ' ou ") sont matérialisés par le caractère ␣ (U+2423 : OPEN BOX). Bien sûr, le caractère U+2423 doit être présent dans la fonte mono-chasse utilisée. [5]
Exemple : `my_string = 'Très␣bonne␣réponse'`

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

    def arctan(x,n=10):
        """Compute the mathematical value of arctan(x)

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

La commande **\PitonOptions** propose d'autres clés qui seront décrites plus loin (voir en particulier la coupure des pages et des lignes p. 8).

3.2 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. [6]

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.

5. L'extension piton utilise simplement la fonte mono-chasse courante. Pour la changer, le mieux est d'utiliser **\setmonofont** de fontspec.
6. On rappelle que tout environnement LaTeX est, en particulier, un groupe.

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. [7]

TABLE 1 – Les styles proposés par piton

Style	Usage
`Number`	les nombres
`String.Short`	les chaînes de caractères courtes (entre `'` ou `"`)
`String.Long`	les chaînes de caractères longues (entre `'''` ou `"""`) sauf les chaînes de documentation
`String`	cette clé fixe à la fois `String.Short` et `String.Long`
`String.Doc`	les chaînes de documentation (seulement entre `"""` suivant PEP 257)
`String.Interpol`	les éléments syntaxiques des champs des f-strings (c'est-à-dire les caractères `{`et `}`)
`Operator`	les opérateurs suivants : `!= == << >> - ~ + / * % = < > & . | @`
`Operator.Word`	les opérateurs suivants : `in`, `is`, `and`, `or` et `not`
`Name.Builtin`	la plupart des fonctions prédéfinies par Python
`Name.Function`	le nom des fonctions définies par l'utilisateur *au moment de leur définition*, c'est-à-dire après le mot-clé `def`
`Name.Decorator`	les décorateurs (instructions débutant par `@`)
`Name.Namespace`	le nom des modules (= bibliothèques extérieures)
`Name.Class`	le nom des classes au moment de leur définition, c'est-à-dire après le mot-clé `class`
`Exception`	le nom des exceptions prédéfinies (eg : SyntaxError)
`Comment`	les commentaires commençant par `#`
`Comment.LaTeX`	les commentaires commençant par `#>` qui sont composés en LaTeX par piton (et appelés simplement « commentaires LaTeX » dans ce document)
`Keyword.Constant`	`True`, `False` et `None`
`Keyword`	les mots-clés suivants : `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.`

7. 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`.

3.3 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`.
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}{O{}}{\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 Les échappements vers LaTeX

L'extension piton propose plusieurs mécanismes d'échappement vers LaTeX :

— Il est possible d'avoir des commentaires entièrement composés en LaTeX.

— Il est possible d'avoir, dans les commentaires Python, les éléments entre `$` composés en mode mathématique de LaTeX.

— Il est possible d'insérer du code LaTeX à n'importe quel endroit d'un listing Python.

4.1.1 Les « commentaires LaTeX »

Dans ce document, on appelle « commentaire LaTeX » des commentaires qui débutent par `#>`. Tout ce qui suit ces deux caractères, et jusqu'à la fin de la ligne, sera composé comme du code LaTeX standard.
Il y a deux outils pour personnaliser ces commentaires.

— Il est possible de changer le marquage syntaxique utilisé (qui vaut initialement `#>`). Pour ce faire, il existe une clé `comment-latex`, disponible seulement au chargement de piton (c'est-à-dire au moment du `\usepackage`), qui permet de choisir les caractères qui (précédés par `#`) serviront de marqueur syntaxique.

Par exemple, avec le chargement suivant :

```
\usepackage[comment-latex = LaTeX]{piton}
```

les commentaires LaTeX commenceront par `#LaTeX`.

Si on donne la valeur nulle à la clé `comment-latex`, tous les commentaires Python (débutant par `#`) seront en fait des « commentaires LaTeX ».

— Il est possible de changer le formatage du commentaire LaTeX lui-même en changeant le style piton `Comment.LaTeX`.

Par exemple, avec `\SetPitonStyle{Comment.LaTeX = \normalfont\color{blue}}`, les commentaires LaTeX seront composés en bleu.

Si on souhaite qu'un croisillon (`#`) soit affiché en début de commantaire dans le PDF, on peut régler `Comment.LaTeX` de la manière suivante :

```
\SetPitonStyle{Comment.LaTeX = \color{gray}\#\normalfont\space }
```

Pour d'autres exemples de personnalisation des commentaires LaTeX, voir la partie 5.2 p. 10.

4.1.2 La clé « math-comments »

Il est possible de demander que, dans les commentaires Python normaux, c'est-à-dire débutant par `#` (et non par `#>`), les éléments placés entre symboles `$` soient composés en mode mathématique de LaTeX (le reste du commentaire restant composé en verbatim).
La clé `math-comments`, qui n'est disponible qu'au chargement de piton (c'est-à-dire au moment du `\usepackage`), active ce comportement.

Dans l'exemple suivant, on suppose que la clé `math-comments` a été utilisée au chargement de piton.

```
\begin{Piton}
def carré(x):
    return x*x # renvoie $x^2$
\end{Piton}
```

```
def carré(x):
    return x*x # renvoie x^2
```

4.1.3 Le mécanisme « espace-inside »

Il est aussi possible de surcharger les listings Python pour y insérer du code LaTeX à peu près n'importe où (mais entre deux lexèmes, bien entendu). Cette fonctionnalité n'est pas activée par défaut par piton. Pour l'utiliser, il faut spécifier les deux caractères marquant l'échappement (le premier le commençant et le deuxième le terminant) en utilisant la clé `escape-inside` au chargement de piton (c'est-à-dire au moment du `\usepackage`). Les deux caractères peuvent être identiques.

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.

```
\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 débuter par `#>`).

4.2 Coupure des pages et des lignes

4.2.1 Coupure des pages

Par défaut les listings produits par l'environnement `{Piton}` et par la commande `\PitonInputFile` sont insécables.
Néanmoins, la commande `\PitonOptions` propose la clé `splittable` pour autoriser et autoriser de telles coupures.

— Si la clé `splittable` est utilisée sans valeur, les listings sont sécables n'importe où.
— Si la clé `splittable` est utilisée avec une valeur numérique n (qui doit être un entier naturel non nul), alors les listings seront sécables mais aucune coupure ne pourra avoir lieu entre les n premières lignes, ni entre les n dernières. De ce fait, `splittable=1` est équivalent à `splittable`.

Remarque
Même avec une couleur de fond (fixée avec `background-color`), les sauts de page sont possibles, à partir du moment où la clé `splittable` est utilisée. [8]

4.2.2 Coupure des lignes

Par défaut les lignes dans les listings produits par `{Piton}` et commande `\PitonInputFile` ne sont pas sécables.

Nouveau 0.99 Il existe néanmoins des clés (disponibles dans `\PitonOptions`) pour autoriser ces coupures.

— La clé `break-lines` active la coupure des lignes. Les seuls points de coupures possibles sont les espaces (y compris dans les chaînes de caractères).
— Avec la clé `indent-broken-lines`, l'indentation de la ligne coupée est respectée à chaque retour à la ligne.
— La clé `end-of-broken-line` correspond au symbole placé à la fin d'une ligne coupée. Sa valeur initiale est : `\hspace*{0.5em}\textbackslash`.
— La clé `continuation-symbol` correspond au symbole placé à chaque retour de ligne dans la marge gauche. Sa valeur initiale est : `+\;`.
— La clé `continuation-symbol-on-indentation` correspond au symbole placé à chaque retour de ligne au niveau de l'indentation (uniquement dans le cas où la clé `indent-broken-lines` est active). Sa valeur initiale est : `$\hookrightarrow\;$`.

Le code suivant a été composé dans une `{minipage}` de largeur 12 cm avec le réglage suivant :

```
\PitonOptions{break-lines,indent-broken-lines,background-color=gray!15}
```

```
def dict_of_liste(liste):
    """Convertit une liste de subrs et de descriptions de \
+   ↪ glyphes en dictionnaire"""
    dict = {}
    for liste_lettre in liste:
        if (liste_lettre[0][0:3] == 'dup'): # si c'est un subr
            nom = liste_lettre[0][4:-3]
            print("On traite le subr de numéro " + nom)
        else:
            nom = liste_lettre[0][1:-3] # si c'est un glyphe
            print("On traite le glyphe du caractère " + nom)
        dict[nom] = [traite_ligne_Postscript(k) for k in \
+       ↪ liste_lettre[1:-1]]
    return dict
```

8. Avec la clé `splittable`, 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`.

4.3 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`.

Néanmoins, il est également 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}` : voir un exemple sur la première page de ce document.

4.4 Tabulations

Même s'il est recommandé d'indenter les listings Python avec des espaces (cf. PEP 8), piton accepte les caractères de tabulations (U+0009) en début de ligne. Chaque caractère U+0009 est remplacé par n espaces. La valeur initiale de n est 4 mais on peut la changer avec la clé `tab-size` de `\PitonOptions`.
Remarque : Contrairement à ce qui se passe avec l'extension listings, la clé `gobble` (et ses variantes `auto-gobble` et `env-gobble`) agit *avant* la transformation des tabulations en espaces.

5 Exemples

5.1 Numérotation des lignes

On rappelle que l'on peut demander la numérotation des lignes des listings avec la clé `line-numbers` ou la clé `all-line-numbers`.
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, on peut utiliser l'option `left-margin=auto` qui va insérer une marge adaptée aux numéros qui seront insérés (elle est plus large quand les numéros dépassent 10).

```
\PitonOptions{background-color=gray!10, left-margin = auto, 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)                                    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) )
```

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)       #> 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 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 4.3 p. 9. 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) [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) )
```

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.

9. Un premier appel récursif.
10. 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}
```

```
def arctan(x,n=10):
    if x < 0:
        return -arctan(-x) [11]
    elif x > 1:
        return pi/2 - arctan(1/x) [12]
    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 3.2, 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*[13] 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 = \normalfont \color{gray},
    Keyword = \bfseries ,
    Name.Namespace = ,
    Name.Class = ,
    Name.Type = ,
    InitialValues = \color{gray}
  }
```

11. Un premier appel récursif.
12. Un deuxième appel récursif.
13. Voir : `https://dejavu-fonts.github.io`

```
from math import pi

def arctan(x,n=10):
    """Compute the mathematical value of arctan(x)

    n is the number of terms in 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) = π/2 pour x > 0)
    else:
        s = 0
        for k in range(n):
            s += (-1)**k/(2*k+1)*x**(2*k+1)
        return s
```

6 Utilisation avec pyluatex

L'extension pyluatex est une extension qui permet l'exécution de code Python à partir de `lualatex` (pourvu que Python soit installé sur la machine et que la compilation soit effectuée avec `lualatex` et `--shell-escape`).
Voici, à titre d'exemple, un environnement `{PitonExecute}` qui formate un listing Python (avec piton) et qui affiche également dessous le résultat de l'exécution de ce code avec Python.

```
\ExplSyntaxOn
\NewDocumentEnvironment { PitonExecute } { ! O { } }
  {
    \PyLTVerbatimEnv
    \begin{pythonq}
  }
  {
    \end{pythonq}
    \directlua
      {
        tex.print("\\PitonOptions{#1}")
        tex.print("\\begin{Piton}")
        tex.print(pyluatex.get_last_code())
        tex.print("\\end{Piton}")
        tex.print("")
      }
    \begin{center}
      \directlua{tex.print(pyluatex.get_last_output())}
    \end{center}
  }
\ExplSyntaxOff
```

Cet environnement `{PitonExcecute}` prend en argument optionnel (entre crochets) les options proposées par la commande `\PitonOptions`.

Voici un exemple d'utilisation de cet environnement `{PitonExecute}` :

```
\begin{PitonExecute}[background-color=gray!15]
def square(x):
    return x*x
```

```
print(f'The square of 12 is {square(12)}.')
\end{PitonExecute}
```

```
def square(x):
    return x*x
print(f'The square of 12 is {square(12)}.')
```

The square of 12 is 144.

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.