Petit guide — `ProfLycee`, `piton` et `pyluatex`

Pseudocode, code Python et console interactive

C. Pierquet

Juin 2026

Table des matières

1 Présentation et chargement **2**
1.1 Chargement 2

2 Pseudocode et code Python statique **2**
2.1 L'environnement `CodePiton` 2
2.2 Fil rouge — styles visuels 3
2.3 Pseudocode 5
2.4 Chargement depuis un fichier 5

3 Console interactive avec pyluatex **6**
3.1 L'environnement `ConsolePiton` 6
3.2 Fil rouge — console interactive 6

4 Style Thonny — éditeur et console **8**
4.1 Les deux environnements 8
4.2 Fil rouge — éditeur + console Thonny 8
4.3 Variante — chargement depuis un fichier 9

5 Récapitulatif **10**

1 Présentation et chargement

Le package `ProfLycee` propose, via sa librairie `piton`, plusieurs environnements pour afficher du code Python (et du pseudocode) avec coloration syntaxique soignée. La librairie s'appuie sur **piton**, un package de coloration syntaxique LuaLaTeX-natif qui ne nécessite aucun appel shell pour le code *statique*.

Pour l'exécution réelle du code (console interactive), on ajoute le package **pyluatex**, qui fait appel à un interpréteur Python externe via `--shell-escape`.

1.1 Chargement

Code

```
% Code statique uniquement (pas d'exécution)
\documentclass[french,a4paper,11pt]{article}
\usepackage{ProfLycee}
\useproflyclib{piton}                    % nécessite LuaLaTeX

% Code exécuté (console, style Thonny)
\documentclass[french,a4paper,11pt]{article}
\usepackage{ProfLycee}
\useproflyclib{piton}
\usepackage[executable=python]{pyluatex}  % LuaLaTeX + --shell-escape
```

À retenir

Compilation obligatoire en LuaLaTeX.
L'option `--shell-escape` n'est requise que lorsqu'on utilise `pyluatex` (exécution réelle du code). Pour le code et le pseudocode statiques, LuaLaTeX seul suffit.

2 Pseudocode et code Python statique

2.1 L'environnement `CodePiton`

Code

```
\begin{CodePiton}[Clés]{}
# code Python ici
\end{CodePiton}
```

Principales clés disponibles :

Clé	Effet	Défaut
`Style`	style visuel : `Classique`, `Carbon`, `Onglet`	`Classique`
`Largeur`	largeur de la boîte	`\linewidth`
`Alignement`	alignement : `flush left/right`, `center`	`flush left`
`Couleur`	couleur d'accent du cadre/onglet	`darkgray`
`CouleurFond`	couleur de fond	selon style
`CouleurNombres`	couleur des numéros de ligne	`gray`
`Cadre=false`	supprime le cadre	`true`
`Lignes=false`	supprime les numéros de ligne	`true`
`Filigrane`	logo Python en filigrane	`false`
`Titre={...}`	titre personnalisé de la boîte	—

2.2 Fil rouge — styles visuels

Code

```
% Style Classique (défaut)
\begin{CodePiton}{}
def valeur_absolue(x) :
  "Renvoie la valeur absolue de x"
  if x > 0 :
    return x
  else:
    return -x
\end{CodePiton}
```

Code Python

```
def valeur_absolue(x) :
  "Renvoie la valeur absolue de x"
  if x > 0 :
    return x
  else:
    return -x
```

Code

```
% Style Classique, largeur réduite, fond coloré
\begin{CodePiton}[Largeur=10cm,CouleurNombres=olive,CouleurFond=cyan!5]{}
def valeur_absolue(x) :
  "Renvoie la valeur absolue de x"
  if x > 0 :
    return x
  else:
    return -x
\end{CodePiton}
```

Code Python

```
def valeur_absolue(x) :
  "Renvoie la valeur absolue de x"
  if x > 0 :
    return x
  else:
    return -x
```

Code

```
% Style Carbon, largeur 10cm, centré
\begin{CodePiton}[Style=Carbon,Largeur=10cm,Alignement=center]{}
def valeur_absolue(x) :
  "Renvoie la valeur absolue de x"
  if x > 0 :
    return x
  else:
    return -x
\end{CodePiton}
```

Code Python

```python
def valeur_absolue(x) :
  "Renvoie la valeur absolue de x"
  if x > 0 :
    return x
  else:
    return -x
```

Code

```
% Sans cadre, aligné à droite, avec filigrane et titre personnalisé
\begin{CodePiton}%
  [Largeur=0.5\linewidth,Cadre=false,Alignement=flush right,%
   Filigrane,Titre={Script}]{}
def valeur_absolue(x) :
  "Renvoie la valeur absolue de x"
  if x > 0 :
    return x
  else:
    return -x
\end{CodePiton}
```

Script

```python
def valeur_absolue(x) :
  "Renvoie la valeur absolue de x"
  if x > 0 :
   return x
  else:
   return -x
```

Code

```
% Style Onglet, couleur rouge, sans numéros de ligne, avec filigrane
\begin{CodePiton}%
  [Style=Onglet,Couleur=red,Largeur=11cm,%
   Filigrane,Alignement=flush left,Lignes=false]{}
def valeur_absolue(x) :
  "Renvoie la valeur absolue de x"
  if x > 0 :
    return x
  else:
    return -x
\end{CodePiton}
```

Code Python

```python
def valeur_absolue(x) :
  "Renvoie la valeur absolue de x"
   if x > 0 :
  return x
    else:
  return -x
```

Remarque

Le second argument obligatoire {} de \begin{CodePiton}{} est réservé à des options piton de bas niveau (styles de coloration avancés). Dans un usage courant, on le laisse vide.

2.3 Pseudocode

Pour afficher du pseudocode dans le même esprit visuel, on dispose de l'environnement `PseudoCodePiton` (et de sa variante fichier `\PseudoCodePitonFichier`). Les clés sont les mêmes que pour `CodePiton`.

Code

```
\begin{PseudoCodePiton}[Largeur=\linewidth,Filigrane]{}
Algorithme : calcul mental
Variables  : nb1, nb2, resultat_saisi

Début
  nb1 ← randint(1, 10)
  nb2 ← randint(1, 10)
  Afficher(nb1, " * ", nb2, " = ? ")
  ...
Fin
\end{PseudoCodePiton}
```

Pseudo-Code

```
1 Algorithme : calcul mental
2 Variables  : nb1, nb2, resultat_saisi
3
4 Début
5   nb1 ← randint(1, 10)
6   nb2 ← randint(1, 10)
7   Afficher(nb1, " * ", nb2, " = ? ")
8   ...
9 Fin
```

2.4 Chargement depuis un fichier

Pour du code stocké dans un fichier externe, deux commandes sont disponibles :

Code

```
% Code Python depuis un fichier
\CodePitonFichier[Clés]{monfichier.py}

% Pseudocode depuis un fichier texte
\PseudoCodePitonFichier[Clés]{monfichier.txt}

% On peut créer le fichier directement dans le document avec scontents :
\begin{scontents}[overwrite,write-out=monfichier.py]
def calcul(n) :
  u = 0
  for i in range(n) :
    u = 3*u + 1
  return u
\end{scontents}
```

Code Python

```
1 def calcul(n) :
2   u = 0
3   for i in range(n) :
4     u = 3*u + 1
5   return u
```

3 Console interactive avec pyluatex

Avec pyluatex, le code Python est **réellement exécuté** par l'interpréteur au moment de la compilation, et les sorties sont capturées et mises en forme automatiquement.

3.1 L'environnement `ConsolePiton`

Code

```
\begin{ConsolePiton}[Options piton]<Clés>{Options tcbox}
instruction1
instruction2
...
\end{ConsolePiton}
```

Clés disponibles (entre <...>) :

Clé	Effet	Défaut
`AltStyle`	style alternatif de la console	`false`
`Logo`	affiche le logo Python dans l'en-tête	`true`
`Largeur`	largeur de la console	`\linewidth`
`Alignement`	alignement de la console	`flush left`

3.2 Fil rouge — console interactive

On commence par définir la fonction dans un bloc python (exécuté mais non affiché), puis on l'utilise dans la console :

Code

```
% Bloc d'initialisation (exécuté, non affiché)
\begin{python}
from random import randint
def valeur_absolue(x) :
  if x > 0 :
    return x
  else:
    return -x
\end{python}
```

Code

```
% Console standard
\begin{ConsolePiton}{}
1+1
2**10
valeur_absolue(-3)
valeur_absolue(0)
print(f"La valeur absolue de 5 est {valeur_absolue(5)}")
\end{ConsolePiton}
```

Début de la console Python

```
>>> 1+1
2
>>> 2**10
1024
>>> valeur_absolue(-3)
3
>>> valeur_absolue(0)
0
>>> print(f"La valeur absolue de 5 est {valeur_absolue(5)}")
La valeur absolue de 5 est 5
```

Fin de la console Python

Code

```
% Console, largeur réduite, centrée, sans logo
\begin{ConsolePiton}<Largeur=11cm,Alignement=center,Logo=false>{}
liste = [randint(1,20) for i in range(10)]
print(liste)
print(max(liste), min(liste), sum(liste))
\end{ConsolePiton}
```

Début de la console Python

```
>>> liste = [randint(1,20) for i in range(10)]
>>> print(liste)
[7, 10, 8, 17, 15, 6, 7, 10, 5, 8]
>>> print(max(liste), min(liste), sum(liste))
17 5 93
```

Fin de la console Python

Code

```
% Console style alternatif, largeur réduite, centrée
\begin{ConsolePiton}<AltStyle,Largeur=10cm,Alignement=center>{}
[i**2 for i in range(50)]
\end{ConsolePiton}
```

Début de la console Python

```
>>> [i**2 for i in range(50)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144,
169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529,
576, 625, 676, 729, 784, 841, 900, 961, 1024, 1089,
1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764,
1849, 1936, 2025, 2116, 2209, 2304, 2401]
```

Fin de la console Python

À retenir

Les résultats affichés dans la console dépendent de l'exécution réelle par Python : les listes aléatoires (`randint`) donneront des valeurs différentes à chaque compilation.
Le bloc `\begin{python}...\end{python}` est persistant dans la session : les fonctions et variables définies restent disponibles pour les blocs suivants.

4 Style Thonny — éditeur et console

Pour un rendu proche de l'environnement de développement **Thonny** (très utilisé en lycée), `ProfLycee` propose deux environnements complémentaires.

4.1 Les deux environnements

Code

```
% Éditeur (affichage du code source, avec onglet de fichier)
\begin{PitonThonnyEditor}[Clés]{Options tcbox}<Options piton>
# code Python
\end{PitonThonnyEditor}

% Console (résultats d'exécution)
\begin{PitonThonnyConsole}[Clés]{Options tcbox}<Options piton>
instructions
\end{PitonThonnyConsole}
```

Clés disponibles (entre `[...]`) :

Clé	Effet	Défaut
`NomFichier`	nom affiché dans l'onglet de l'éditeur	`script.py`
`IntroConsole`	texte d'introduction de la console	version Python détectée
`Largeur`	largeur de l'environnement	`\linewidth`

4.2 Fil rouge — éditeur + console Thonny

On définit d'abord la fonction dans un bloc `python` (exécuté silencieusement), puis on affiche l'éditeur et la console séparément :

Code

```
% Initialisation silencieuse
\begin{python}
from math import gcd
def est_duffy(n) :
  nb_div, somme_div = 0, 0
  for i in range(1, n+1) :
    if n % i == 0 :
      nb_div += 1
      somme_div += i
  return gcd(somme_div, n) == 1
\end{python}
```

Code

```
% Éditeur Thonny
\begin{PitonThonnyEditor}[NomFichier=tpcapytale.py,Largeur=12cm]{}
#PROJET CAPYTALE
from math import gcd
def est_duffy(n) :
  nb_div = 0
  somme_div = 0
  for i in range(1, n+1) :
    if n % i == 0 :
      nb_div += 1
      somme_div += i
  return gcd(somme_div, n) == 1
\end{PitonThonnyEditor}
```

tpcapytale.py

```python
#PROJET CAPYTALE
from math import gcd
def est_duffy(n) :
  nb_div = 0
  somme_div = 0
  for i in range(1, n+1) :
    if n % i == 0 :
      nb_div += 1
      somme_div += i
  return gcd(somme_div, n) == 1
```

Code

```
% Console Thonny associée
\begin{PitonThonnyConsole}[IntroConsole={/usr/bin/python 3.12},Largeur=12cm]{}
est_duffy(6)
est_duffy(13)
est_duffy(265)
from random import randint
nb = randint(1,100000)
nb, est_duffy(nb)
\end{PitonThonnyConsole}
```

console

```
/usr/bin/python 3.12
 >>> est_duffy(6)
 False
 >>> est_duffy(13)
 True
 >>> est_duffy(265)
 True
 >>> from random import randint
 >>> nb = randint(1,100000)
 >>> nb, est_duffy(nb)
 (47710, False)
```

Remarque

L'éditeur et la console Thonny fonctionnent indépendamment : l'éditeur est purement *statique* (affichage du code source), tandis que la console fait appel à `pyluatex` pour l'exécution réelle. On peut donc utiliser l'éditeur seul, sans `pyluatex`, si on ne souhaite qu'afficher le code.

4.3 Variante — chargement depuis un fichier

Code

```
% Éditeur Thonny depuis un fichier externe
\PitonThonnyEditorFichier[NomFichier=script.py,Largeur=9cm]{monfichier.py}
```

5 Récapitulatif

Environnement / Commande	Exécution	pyluatex	Style
`CodePiton`	non	non	Classique / Carbon / Onglet
`CodePitonFichier`	non	non	idem, depuis fichier
`PseudoCodePiton`	non	non	Classique / Carbon / Onglet
`PseudoCodePitonFichier`	non	non	idem, depuis fichier
`ConsolePiton`	oui	**oui**	console REPL colorée
`PitonThonnyEditor`	non	non	style Thonny éditeur
`PitonThonnyEditorFichier`	non	non	idem, depuis fichier
`PitonThonnyConsole`	oui	**oui**	style Thonny console

— Documentation complète : https://ctan.org/pkg/proflycee
— `piton` : https://ctan.org/pkg/piton
— `pyluatex` : https://ctan.org/pkg/pyluatex