

I. Présenter du code informatique

Certains packages proposent des fonctionnalités nécessitant de coder un peu en Lua.¹ Pour ces projets, la documentation doit pouvoir présenter des lignes de code Lua ; c'est pour cette raison que `tutodoc` permet de faire cela aisément, et bien plus.²

Important.

Les outils de cette section permettent aussi de présenter du code \LaTeX , mais il ne faut pas les utiliser pour de simples cas d'utilisation, car les macros et les environnements présentés juste après servent à étudier du code, et non juste à l'employer : se reporter à la section ?? page ?? pour faire appel aux bons outils pour la mise en forme de cas d'utilisation \LaTeX .

1. Codes « en ligne »

La macro `\tdoccodein` attend deux arguments : le 1^{er} indique le langage de programmation, et le 2^e donne le code à mettre en forme. Il est possible d'utiliser une option de fonctionnement identique à ce que propose `\tdoclathexin` : voir la section ?? page ?? . Voici des cas d'utilisation possibles.³

```
1: \tdoccodein{py}{print("OK" if i = 0 else "KO")} \\
2: \tdoccodein[style = bw]{py}{print("OK" if i = 0 else "KO")} \\
3: \tdoccodein[style = igor, showspace]%
   {py}{print("OK" if i = 0 else "KO")}
-----
1: print("OK" if i = 0 else "KO")
2: print("OK" if i = 0 else "KO")
3: print("OK" if i = 0 else "KO")
```

Note.

Sur la page <https://pygments.org/languages/> se trouve la liste complète des langages supportés avec leur nom court. Par exemple, il est possible de mettre en forme du `Brainfuck` comme cette séquence `+++++++[>++++>++++>++++>+<<<<-]>+>+.+++++.+++.` qui sert à afficher `Hello`.

2. Codes tapés directement

On peut taper directement du code dans un document \LaTeX via `\begin{tdoccode} ... \end{tdoccode}` qui attend un argument indiquant le langage de programmation, et d'éventuelles options entre crochets et/ou entre chevrons de fonctionnements identiques à ce que propose `\begin{tdoclathex}` ... `\end{tdoclathex}` : voir la section ?? page ?? .⁴

Exemple I.1 (Fonctionnement standard).

```
\begin{tdoccode}{pl}
print "Qui êtes-vous ? ";
my $name = <STDIN>;

chomp($name);

if ($name eq "") {
    print "Ah, pas très bavard aujourd'hui !\n";
} else {
    print "Bonjour $name.\n";
    print "Épatant ! En fait, pas du tout...\n";
}
```

1. Pour les mathématiques, on peut citer `luacas` et `tkz-elements`.

2. La mise en forme des codes étant faite via les packages `minted` et `tcolorbox`, les macros et les environnements présentés dans cette section permettent la mise en forme de codes dans tous les langages supportés par `Pygments`, un projet Python utilisé en coulisse par `minted`.

3. Une couleur de fond est volontairement utilisée pour subtilement faire ressortir les codes mis en forme. Par exemple, taper `\tdoccodein{py}{funny = "ah"*3}` produira `funny = "ah"*3`.

4. Noter que la coloration des codes \LaTeX est correcte du point de vue lexical, mais fautive sémantiquement parlant.

```
\end{tdoccode}
```

Ceci donne :

```
print "Qui êtes-vous ? ";
my $name = <STDIN>;

chomp($name);

if ($name eq "") {
    print "Ah, pas très bavard aujourd'hui !\n";
} else {
    print "Bonjour $name.\n";
    print "Épatant ! En fait, pas du tout...\n";
}
```

Exemple I.2 (Personnalisation ponctuelle du rendu).

```
\begin{tdoccode}[style = solarized-light, linenos]%
    <leftrule = 22pt, colback = orange!5, colframe = red!35>%
    {lua}
io.write("Qui êtes-vous ? ")
local name = io.read()

if name == "" then
    print("Ah, pas très bavard aujourd'hui !")
else
    print("Bonjour " .. name .. ".")
    print("Épatant ! En fait, pas du tout...")
end
\end{tdoccode}
```

Ceci donne :

```
1 io.write("Qui êtes-vous ? ")
2 local name = io.read()
3
4 if name == "" then
5     print("Ah, pas très bavard aujourd'hui !")
6
7 else
8     print("Bonjour " .. name .. ".")
9     print("Épatant ! En fait, pas du tout...")
10 end
```