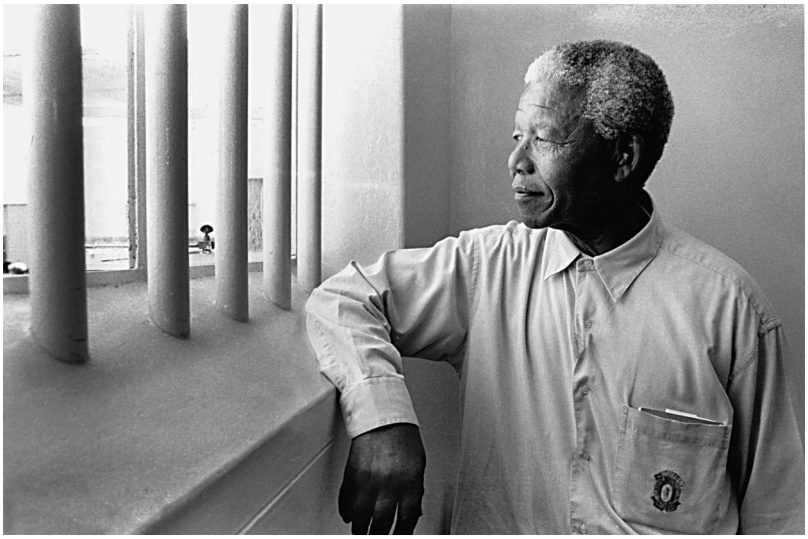
PYTHON et LaTeX
Comment générer des sujets aléatoirement à l'aide du package pythontex ?

Laurence LE FOLL - Jean-Marc DUQUESNOY - Fabrice EUDES
Groupe ArCSiN — IREM de Lille
Dernière version : 25 mars 2025

Ubuntu, notion humaniste, originaire du Sud de l'Afrique, qui pourrait être traduite par « je suis ce que je suis grâce à ce que nous sommes »



Regardez vers le
voyez, et demandez-vous ce qui fait que l'univers existe. Soyez curieux.

Stephen Hawking , physicien théoricien et cosmologiste.

Table des matières

1 Introduction

L'un des objectifs de ce document est de montrer comment, dans un source LaTeX[1], inclure un script, codé en langage PYTHON, qui, après une succession de trois compilations[2], produit un fichier au format pdf, le code PYTHON ayant été interprété.
L'autre objectif sera de proposer une méthode permettant, à partir d'un sujet donné écrit en LaTeX, de produire plusieurs sujets identiques, mais contenant des données générées aléatoirement. Des explications seront proposées à la section 10.
Nous n'avons pas la prétention, et c'est un euphémisme de l'écrire, de dresser une liste exhaustive des possibilités de ce package, ni celle des innombrables possibilités du langage PYTHON. De nombreux documents très bien construits sont à disposition sur la toile.
Le document existe, avec ses nombreux défauts, mais il est là…
À vous d'en faire bon usage, ou pas…
N'en déplaise aux esprits chagrins, le package pythontex est un moyen de procéder pour inclure du code PYTHON, mais évidemment pas le seul.
L'outil idéal est, à notre sens, celui qui répond à vos besoins et problématiques à moindre frais…

2 Quelles compilations ?

Après avoir enregistré le fichier source LaTeX sous le nom « `MonSource.tex` »[3], il suffira, dans un terminal[4], à la racine de votre source LaTeX, d'exécuter[5] successivement les commandes suivantes :

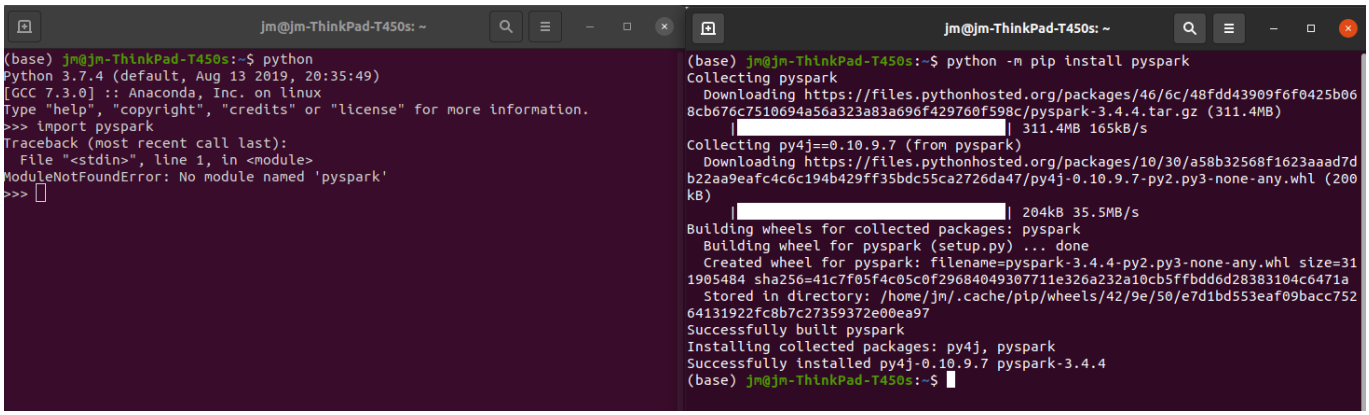
- `pdflatex MonSource.tex`
  puis
- `pythontex MonSource.tex`
  et enfin
- `pdflatex MonSource.tex`

Le fichier « `MonSource.pdf` » est alors créé et les scripts codés en langage PYTHON dans le source « `MonSource.tex` » ont été interprétés.
Nous conseillons aux utilisateurs des éditeurs **Texmaker** et **TeXstudio** de visiter la page `https://www.mathweb.fr/euclide/pythontex/` pour configurer cette suite de commandes dans leur éditeur.
Précision importante : lors des compilations pythontex, il est possible qu'apparaisse un message d'erreur disant que telle ou telle librairie PYTHON n'est pas installée.
Nous allons, dans ce qui suit, proposer une méthode permettant de savoir si une librairie est, ou non, installée.
Exemple 1 : la librairie *sympy*
En console, on appelle l'interpréteur PYTHON, puis importe la librairie *sympy*.

```
(base) jm@jm-ThinkPad-T450s:~$ python
Python 3.7.4 (default, Aug 13 2019, 20:35:49)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import sympy
>>>
```

1. Le package pythontex ayant été inséré dans le préambule du source LaTeX
2. Le mode d'emploi sera précisé dans la section qui suit.
3. Inclure le package pythontex dans le préambule de votre source LaTeX. Le préambule utilisé, nommé « `preambule_alea.tex` », sera fourni pour qu'un éventuel utilisateur puisse démarrer sans trop de soucis.
4. Terminal ou shell ou console, selon que l'on travaille sous une distribution Linux ou sous Window$.
5. Si le source LaTeX contient du code pstricks, il faudra suivre les instructions qui se trouvent dans la section 10.3.

On remarque que la librairie *sympy* est installée.
Exemple 2 : la librairie *pyspark*
La console de gauche indique que cette librairie n'est pas installée. On ouvre une deuxième console et la ligne de commande `python -m pip install` va alors installer la librairie.



3 Quelques balises pythontex utiles

3.1 On peut afficher une console « virtuelle »

On utilise la balise `pyconsole`.
Voici un exemple d'utilisation qui compile quelques commandes PYTHON, on insère dans un source LaTeX le script qui suit :

Code LaTeX

```
\begin{pyconsole}
from math import *
x=2
y=6
z=x+y
x,y,z=y,x,3*z
x
y
z
L=[3,1,8,9,2]
L.append(99)
L
z==8
a=2
b=3
c=5*sin(a)+2*cos(b)
print(c)
for i in range(5):
        print(2*i*exp(-i))

\end{pyconsole}
```

Ce qui donne à l'exécution :

Sortie LaTeX

```
>>> from math import *
>>> x=2
>>> y=6
>>> z=x+y
>>> x,y,z=y,x,3*z
>>> x
6
>>> y
2
>>> z
24
>>> L=[3,1,8,9,2]
>>> L.append(99)
>>> L
[3, 1, 8, 9, 2, 99]
>>> z==8
False
>>> a=2
>>> b=3
>>> c=5*sin(a)+2*cos(b)
>>> print(c)
2.566502140927518
>>> for i in range(5):
...     print(2*i*exp(-i))
...
0.0
0.7357588823428847
0.5413411329464508
0.29872241020718365
0.14652511110987343
```

3.2 Afficher un script Python non interprété à la compilation

Il suffit d'insérer le code suivant, par exemple, à l'aide de la balise `pyblock` :

Code LaTeX

```
\begin{pyblock}
from math import *
a=2
b=3
c=5*sin(a)+2*cos(b)
\end{pyblock}
```

qui affiche en sortie :

Sortie LaTeX

```
from math import *
a=2
b=3
c=5*sin(a)+2*cos(b)
```

Si on veut numéroter chaque ligne de code, il suffit d'insérer le code suivant :

Code LaTeX

```
\begin{pyblock}[][numbers=left]
from math import *
a=2
b=3
c=5*sin(a)+2*cos(b)
\end{pyblock}
```

qui affiche en sortie :

Sortie LaTeX

```
1 from math import *
2 a=2
3 b=3
4 c=5*sin(a)+2*cos(b)
```

3.3 Compiler du code Python sans afficher le script associé

Il suffit d'insérer le code suivant à l'aide de la balise `pycode` :

Code LaTeX

```
\begin{pycode}
n=0
u=3
while u<10:
        n=n+1
        u=1.1*u
        print(n,u)
\end{pycode}
```

Ce qui affichera après les trois compilations successives :

Sortie LaTeX

```
1 3.3000000000000003
2 3.630000000000001
3 3.993000000000001
4 4.392300000000001
5 4.831530000000002
6 5.314683000000002
7 5.846151300000003
8 6.430766430000004
9 7.073843073000004
10 7.781227380300005
11 8.559350118330006
12 9.415285130163006
13 10.356813643179308
```

Il suffira ensuite, si on veut faire apparaître dans le fichier « `MonSource.pdf` » généré les dernières valeurs calculées des variables u et n, d'insérer, par exemple, dans le source :

Code LaTeX

```
La dernière valeur de $u$ calculée est égale à \py{u}, et celle de $n$ à \py{n}.
```

Ce qui donne :

Sortie LaTeX

La dernière valeur de u calculée est égale à 10.356813643179308, et celle de n à 13.

3.4 Compiler du code Python qui génère et affiche un graphique

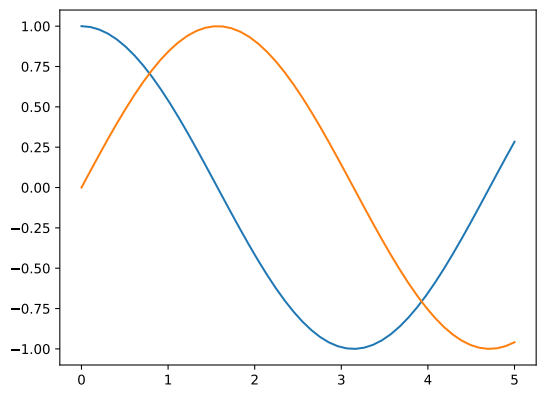
Par exemple, on va générer les graphes des fonctions sinus et cosinus sur l'intervalle $[0;5]$ et les insérer dans le pdf. On insère le code suivant dans le source LaTeX :

Code LaTeX

```
\begin{pycode}
from  math import *
from matplotlib import pyplot
X=linspace(0,5)
plt.plot(X,cos(X))
plt.plot(X,sin(X))
# On sauvegarde la figure suivant le chemin (relatif) mentionné
plt.savefig('Figures/graphique.pdf',bbox_inches='tight')
print(r'\includegraphics[scale=0.5]{Figures/graphique.pdf}')
\end{pycode}
```

Ce qui donne, après les trois compilations, une figure insérée dans le pdf :

Sortie LaTeX



La méthode alternative suivante permet de gérer le placement de la figure.

- Le code LaTeX suivant remplace le précédent :

Code LaTeX

```
\begin{pycode}
from  math import *
from matplotlib import pyplot
X=linspace(0,5)
```

```
plt.plot(X,cos(X))
plt.plot(X,sin(X))
# On sauvegarde la figure suivant le chemin (relatif) mentionné
plt.savefig('Figures/graphique.pdf',bbox_inches='tight')
\end{pycode}
```

On remarque que la ligne suivante a été supprimée.

Code LaTeX

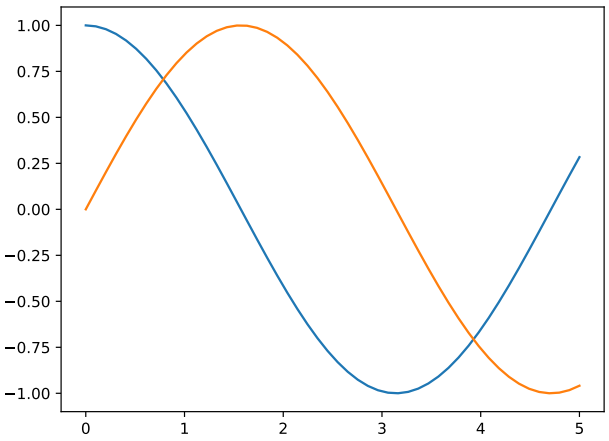
```
print(r'\includegraphics[scale=0.5]{Figures/graphique.pdf}')
```

- Après compilations, le graphique a été généré et enregistré dans l'arborescence à l'emplacement « `Figures/graphique.pdf` ».
  On peut alors insérer dans « `MonSource.tex` » la ligne qui suit à l'endroit désiré :

Code LaTeX

```
\begin{center}
  \IfFileExists{Figures/graphique.pdf}{\includegraphics[scale=0.57]
  ↪  {Figures/graphique.pdf}}{}
\end{center}
```

L'avantage est que la figure peut être placée n'importe où dans le fichier PDF généré, par exemple ici.



4 Insérer un tableau de valeurs

Dans ce paragraphe, nous allons proposer quelques scripts à insérer dans un source LaTeX afin de générer un(des) tableau(x).
Le lecteur (et utilisateur ?) remarquera qu'une *ligne de code LaTeX* est insérée dans `print(r"ligne de code LaTeX")`.

4.1 Tableau de valeurs associées à une fonction

Soit f la fonction définie sur $\mathbb{R}$ par $f(x) = xe^{-x}$.
Déterminons les valeurs de $f(x)$ pour x variant de 0 à 5, avec un pas de 1, valeurs que l'on va insérer dans un tableau centré sur la page.
Insérons le code suivant dans le source LaTeX :

Code LaTeX

```
\begin{pycode}
from math import *
def f(x):
        return(x*exp(-x))
print(r'\begin{center}')
print(r'\begin{tabular}{|c|c|}')
print(r'\hline')
print(r'$x$ & $f(x)$ \\ \hline')
for i in range(0,6):
        print(r'%d & %s \\ \hline' %(i,str(f(i))))
print(r'\end{tabular}')
print(r'\end{center}')
\end{pycode}
```

Ce qui donne après compilations :

Sortie LaTeX

x	$f(x)$
0	0.0
1	0.36787944117144233
2	0.2706705664732254
3	0.14936120510359183
4	0.07326255555493671
5	0.03368973499542734

4.2 Remplir une table de vérité (programme BTS SIO)

⚠ Il est recommandé, pour une première lecture, dans le but de tester les scripts, de passer ce paragraphe qui pourrait paraître indigeste. 🙂

Deux exemples de connecteurs logiques particuliers

La barre de *Sheffer* est le connecteur binaire qui, à toutes propositions P et Q, associe la proposition, notée P | Q, équivalente à $\neg(P \wedge Q)$. On l'appelle aussi « nand », contraction de « not and ».
Le connecteur de *Peirce* est le connecteur binaire qui, à toutes propositions P et Q, associe la proposition, notée P ↓ Q, équivalente à $\neg(P \vee Q)$. On l'appelle aussi « nor », contraction de « not or ».
Ils sont particuliers car chacun de ces connecteurs permet à lui-seul d'obtenir la négation, la conjonction et la disjonction.
P, Q et R étant des propositions quelconques, établir la table de vérité des propositions P ↓ (Q | R) et (P ↓ Q) | (P ↓ R).
Voici le code à inclure dans le source :

Code LaTeX

```
\begin{pycode}
import math

def nand(p,q):
        return(not(p and q))
def nor(p,q):
        return(not(p or q))
```

```
L=[True,False]
Tab3=[]
for i in L:
        for j in L:
                for k in L:
                        Tab3.append([str(i),str(j),str(k),str(nor(i,nand(j,k)))
                         ↪  ,str(nand(nor(i,j),nor(i,k)))])

print(r'\begin{tabular}{|c|c|c|c|c|}')
print(r'\hline ')
print(r'$P$ & $Q$ & $R$ & $P \downarrow (Q \mid R)$ & $(P \downarrow Q) \mid (P
 ↪  \downarrow R)$ \\ \hline ')
for i in range(len(Tab3)):
        print(r'%s & %s & %s & %s & %s \\ \hline' %(str(Tab3[i][0]),str(Tab3[i][1])
         ↪  ,str(Tab3[i][2]),str(Tab3[i][3]),str(Tab3[i][4])))
print(r'\end{tabular}')
\end{pycode}
```

Ce qui donne, après exécution du script :

Sortie LaTeX

P	Q	R	P ↓ (Q | R)	(P ↓ Q) | (P ↓ R)
True	True	True	False	True
True	True	False	False	True
True	False	True	False	True
True	False	False	False	True
False	True	True	True	True
False	True	False	False	True
False	False	True	False	True
False	False	False	False	False

5 Utiliser la librairie sympy

La librairie *sympy* permet le calcul symbolique.

⚠ Cette librairie *sympy* n'est pas nécessairement installée dans la configuration par défaut de l'interpréteur PYTHON.

5.1 Développement, factorisation

Le code suivant, inséré dans une source LaTeX, donne un exemple pour développer ou factoriser une expression.

Code LaTeX

```
\begin{pycode}
from  math import *
from sympy import *
x=Symbol('x')
y=Symbol('y')
exp=x*(5*x+2)*(-6*x+4)
```

```
exp3=(x+y)**6
exp2=x**3-4*x**2+5*x-2
\end{pycode}

Développement : $\py{latex(exp)}=\py{latex(expand(exp))}$

Développement : $\py{latex(exp3)}=\py{latex(expand(exp3))}$

Factorisation : $\py{latex(exp2)}=\py{latex(factor(exp2))}$
```

Ce qui donne :

Sortie LaTeX

Développement : $x(4-6x)(5x+2) = -30x^3 + 8x^2 + 8x$

Développement : $(x+y)^6 = x^6 + 6x^5y + 15x^4y^2 + 20x^3y^3 + 15x^2y^4 + 6xy^5 + y^6$

Factorisation : $x^3 - 4x^2 + 5x - 2 = (x-2)(x-1)^2$

5.2 Dérivée et primitive d'une fonction, intégrale d'une fonction sur un intervalle

Dérivée d'une fonction

On insère le code suivant dans le source LaTeX pour déclarer une fonction et calculer sa dérivée :

Code LaTeX

```
\begin{pycode}
from  math import *
from sympy import *
x=Symbol("x")
f=x**3+x**2-5*x+10
deriv=diff(f,x)
\end{pycode}
```

Puis le code suivant dans le source LaTeX pour afficher la fonction, sa dérivée et l'expression factorisée de cette dernière.

Code LaTeX

```
Soit $f$ la fonction définie sur \R par: $f(x)=\py{latex(f)}$

La fonction dérivée est la fonction $f':x \longmapsto \py{latex(deriv)} =
 ↪  \py{latex(factor(deriv))}$
```

On obtient :

Sortie LaTeX

Soit f la fonction définie sur $\mathbb{R}$ par : $f(x) = x^3 + x^2 - 5x + 10$
La fonction dérivée est la fonction $f' : x \longmapsto 3x^2 + 2x - 5 = (x-1)(3x+5)$

Primitive et intégrale

Le code suivant, inséré dans le source LaTeX, permet de calculer l'image d'un réel par une fonction, de déterminer une primitive, et(ou) calculer l'intégrale sur un intervalle donné, d'une fonction.

Code LaTeX

```
\begin{pycode}
from  math import *
from sympy import *
x=Symbol('x')
expression=x*(5*x+2)*(-6*x+4)
a=expression.subs(x,2)
b=integrate(expression,x)
c=integrate(expression,(x,2,3))
\end{pycode}
```

Puis le code suivant dans le source LaTeX pour afficher les résultats.

Code LaTeX

```
Si $f(x)=\py{latex(expression)}$

alors $f(2)=\py{latex(a)}$

et $\int f(x)\text{d}x=\py{latex(b)}$

et  $\int_2^3 f(x)\text{d}x=\py{latex(c)}$
```

Ce qui donne :

Sortie LaTeX

Si $f(x)=x(4-6x)(5x+2)$
alors $f(2)=-192$
et $\displaystyle\int f(x)\mathrm{d}x=-\frac{15x^4}{2}+\frac{8x^3}{3}+4x^2$
et $\displaystyle\int_2^3 f(x)\mathrm{d}x=-\frac{2501}{6}$

Résolution d'équation

Pour résoudre une équation du type $f(x)=0$ dans le cas où $f(x)=x(5x+2)(-6x+4)$, on insère le code suivant dans le source LaTeX :

Code LaTeX

```
\begin{pycode}
d=solve(expression,x)
\end{pycode}
```

Puis le code LaTeX suivant :

Code LaTeX

```
$f(x)=0$ si $x \in \py{latex(d)}$
```

Ce qui donne :

Sortie LaTeX

$f(x) = 0$ si $x \in \left[-\frac{2}{5},\ 0,\ \frac{2}{3}\right]$

⚠ Python renvoie l'ensemble des solutions entre crochets, cela peut prêter à confusion.

6 Aide aux développements

Le script Python qui suit génère aléatoirement les coefficients de différents polynômes de degré 1, pour proposer une liste permettant aux élèves de se familiariser avec la distributivité de la multiplication par rapport à l'addition :

Code Python

```
\begin{pyblock}
from math import *
from sympy import *
from random import *
# x variable symbolique
x=symbols('x')
# ou bien x=Symbol('x'), ou bien x=Var('x')
# La liste C va contenir les 20 polynômes de degré 1 dont les coefficients
# sont générés aléatoirement.
C=[]
for i in range(20):
        C.append((randint(1,10)*x+randint(-10,10)))
# la liste d contient les coefficients placés devant le produit de 2 éléments
# de la liste C. Certains coefficients seront des rationnels
d=[]
for i in range(6):
    k=randint(-20,20)
    if k!=0:
        d.append(k)
\end{pyblock}
```

On insère alors le code suivant dans le source LaTeX.

Code LaTeX

```
\begin{multicols}{2}
\raggedcolumns%
Les développements proposés :
\begin{enumerate}
\item $A(x)=(\py{latex(C[0])}) \times (\py{latex(C[1])})$
\item $A(x)=(\py{latex(C[2])}) \times (\py{latex(C[3])})$
\item $A(x)=(\py{latex(C[4])}) \times (\py{latex(C[5])})$
\item $A(x)=(\py{latex(C[6])}) \times (\py{latex(C[7])})$
```

```
\item $A(x)=(\py{latex(C[8])}) \times (\py{latex(C[9])})$
\item $A(x)=(\py{latex(C[10])}) \times (\py{latex(C[11])})$
\item $A(x)=\py{d[0]} \times(\py{latex(C[12])}) \times (\py{latex(C[13])})$
\item $A(x)=\py{d[1]} \times(\py{latex(C[14])}) \times (\py{latex(C[15])})$
  % Rational(5,8) renvoie le rationnel 5/8
\item $A(x)=\py{latex(Rational(d[2],d[3]))} \times(\py{latex(C[16])}) \times
 ↪  (\py{latex(C[17])})$
\item $A(x)=\py{latex(Rational(d[4],d[5]))} \times(\py{latex(C[18])}) \times
 ↪  (\py{latex(C[19])})$
\end{enumerate}

\columnbreak

Les réponses sont ici :

\begin{enumerate}
\item $A(x)=\py{latex(expand(C[0]*C[1]))}$
\item $A(x)=\py{latex(expand(C[2]*C[3]))}$
\item $A(x)=\py{latex(expand(C[4]*C[5]))}$
\item $A(x)=\py{latex(expand(C[6]*C[7]))}$
\item $A(x)=\py{latex(expand(C[8]*C[9]))}$
\item $A(x)=\py{latex(expand(C[10]*C[11]))}$
\item $A(x)=\py{latex(expand(d[0]*C[12]*C[13]))}$
\item $A(x)=\py{latex(expand(d[1]*C[14]*C[15]))}$
\item $A(x)=\py{latex(expand(Rational(d[2],d[3])*C[16]*C[17]))}$
\item $A(x)=\py{latex(expand(Rational(d[4],d[5])*C[18]*C[19]))}$
\end{enumerate}
\end{multicols}
```

Ce qui donne :

Sortie LaTeX

Les développements proposés :

1. $A(x) = (5x-2) \times (4x-2)$
2. $A(x) = (9x-4) \times (7x-5)$
3. $A(x) = (9x+6) \times (10x+10)$
4. $A(x) = (5x-1) \times (10x+5)$
5. $A(x) = (7x+2) \times (7x-9)$
6. $A(x) = (5x+5) \times (3x+1)$
7. $A(x) = 16 \times (2x-10) \times (4x+8)$
8. $A(x) = -12 \times (5x+4) \times (3x-5)$
9. $A(x) = -\dfrac{7}{19} \times (x-2) \times (8x-1)$
10. $A(x) = -\dfrac{13}{12} \times (7x-6) \times (7x+6)$

Les réponses sont ici :

1. $A(x) = 20x^2 - 18x + 4$
2. $A(x) = 63x^2 - 73x + 20$
3. $A(x) = 90x^2 + 150x + 60$
4. $A(x) = 50x^2 + 15x - 5$
5. $A(x) = 49x^2 - 49x - 18$
6. $A(x) = 15x^2 + 20x + 5$
7. $A(x) = 128x^2 - 384x - 1280$
8. $A(x) = -180x^2 + 156x + 240$
9. $A(x) = -\dfrac{56x^2}{19} + \dfrac{119x}{19} - \dfrac{14}{19}$
10. $A(x) = 39 - \dfrac{637x^2}{12}$

7 Une liste de dérivées et de primitives

Le but de cet exemple est de proposer un script pythontex qui déclare une liste de fonctions, puis affiche pour chacune de ces fonctions sa dérivée et une de ses primitives.

⚠ Il est recommandé, pour une première lecture, dans le but de tester les scripts, de passer ce paragraphe qui pourrait paraître indigeste. 🙂

On insère le code suivant dans le source LaTeX.

Code LaTeX

```
Liste de fonctions avec leur fonction dérivée et une primitive :

\begin{pycode}
from sympy import *

# Définition de la variable
x = symbols('x')

# Liste de fonctions
fonctions = ['x**2','x/(x+1)','x*sin(x)','x*exp(-x)','sin(x)', 'cos(x)',
 ↪  'tan(x)','sin(x)**2', 'cos(x)**2', 'tan(x)**2', 'x*log(x)', 'atan(x)',
 ↪  'sin(3*x)', 'cos(5*x)', 'tan(x)','asin(x)','acos(x)']

# Génération du tableau LaTeX
print(r'\begin{tabular}{|Cc|Cc|Cc|}')
print(r'\hline')
print(r' Expression de $f(x)$ & Dérivée de $f$ & Primitive de $f$ \\')
print(r'\hline')

# Remplissage du tableau
for f in fonctions:
    deriv = diff(f, x)
    primitive = integrate(f, x)
    print(r' $%s$ & $%s$ & $%s$ \\ ' % (latex(eval(f)), latex(deriv),
     ↪  latex(primitive)))
    print(r'\hline')

print(r'\end{tabular}')
\end{pycode}
```

Ce qui donne :

Liste de fonctions avec leur fonction dérivée et une primitive :

Sortie LaTeX

Expression de $f(x)$	Dérivée de f	Primitive de f
x^2	$2x$	$\frac{x^3}{3}$
$\frac{x}{x+1}$	$-\frac{x}{(x+1)^2}+\frac{1}{x+1}$	$x-\log(x+1)$
$x\sin(x)$	$x\cos(x)+\sin(x)$	$-x\cos(x)+\sin(x)$
xe^{-x}	$-xe^{-x}+e^{-x}$	$(-x-1)e^{-x}$
$\sin(x)$	$\cos(x)$	$-\cos(x)$
$\cos(x)$	$-\sin(x)$	$\sin(x)$
$\tan(x)$	$\tan^2(x)+1$	$-\log(\cos(x))$
$\sin^2(x)$	$2\sin(x)\cos(x)$	$\frac{x}{2}-\frac{\sin(x)\cos(x)}{2}$
$\cos^2(x)$	$-2\sin(x)\cos(x)$	$\frac{x}{2}+\frac{\sin(x)\cos(x)}{2}$
$\tan^2(x)$	$\left(2\tan^2(x)+2\right)\tan(x)$	$-x+\frac{\sin(x)}{\cos(x)}$
$x\log(x)$	$\log(x)+1$	$\frac{x^2\log(x)}{2}-\frac{x^2}{4}$
$\operatorname{atan}(x)$	$\frac{1}{x^2+1}$	$x\operatorname{atan}(x)-\frac{\log\left(x^2+1\right)}{2}$
$\sin(3x)$	$3\cos(3x)$	$-\frac{\cos(3x)}{3}$
$\cos(5x)$	$-5\sin(5x)$	$\frac{\sin(5x)}{5}$
$\tan(x)$	$\tan^2(x)+1$	$-\log(\cos(x))$
$\operatorname{asin}(x)$	$\frac{1}{\sqrt{1-x^2}}$	$x\operatorname{asin}(x)+\sqrt{1-x^2}$
$\operatorname{acos}(x)$	$-\frac{1}{\sqrt{1-x^2}}$	$x\operatorname{acos}(x)-\sqrt{1-x^2}$

8 Nombres premiers, décomposition en facteurs premiers, PGCD

Pour obtenir ce qui suit, il suffit de compiler le source LaTeX « `NombresPremiers.tex` »
Lien de téléchargement : https://nuage03.apps.education.fr/index.php/s/s2ZSZ8BiE8wY6TJ
Les compilations donnent

Sortie LaTeX

On donne la liste L_{130} des nombres premiers inférieurs à 130 :

$L_{130} = [2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,109,113,127]$

1. Déterminer la décomposition en facteurs premiers du pgcd de $2^1\times3^2\times5^3\times7^4$ et de $2^2\times3^3\times5^3\times7^2$.
2. Décomposer le nombre 226 en produit de facteurs premiers.
3. Décomposer le nombre 122 en produit de facteurs premiers.
4. Justifier que le nombre 2029 est premier.

Éléments de correction

1. Déterminer la décomposition en facteurs premiers du pgcd de $2^1 \times 3^2 \times 5^3 \times 7^4$ et de $2^2 \times 3^3 \times 5^3 \times 7^2$.
   Le pgcd de $2^1 \times 3^2 \times 5^3 \times 7^4$ et de $2^2 \times 3^3 \times 5^3 \times 7^2$ est égal à $\boxed{2^1 \times 3^2 \times 5^3 \times 7^2}$.
2. Décomposer le nombre 226 en produit de facteurs premiers.
   Voici la décomposition en facteurs premiers de 226, obtenue à l'aide des divisions successives :

226	2
113	113
1	1

   Ce qui donne :

$$\boxed{226 = 2^1 \times 113^1}$$

3. Décomposer le nombre 122 en produit de facteurs premiers.
   Voici la décomposition en facteurs premiers de 122, obtenue à l'aide des divisions successives :

122	2
61	61
1	1

   Ce qui donne :

$$\boxed{122 = 2^1 \times 61^1}$$

4. Justifier que le nombre 2029 est premier.
   On calcule $\sqrt{2029} \approx 45{,}044422518220834$.
   On remarque qu'aucun nombre premier (voir la liste L_{130}) plus petit que 45 n'est un diviseur de 2029.
   Donc 2029 est premier.

9 Triangle de Pascal

On insère le code suivant dans le source LaTeX.

Code LaTeX

```
Voici le triangle de Pascal qui affiche les $\binom{n}{k}$ pour $n$ variant de 0 à
 ↪  13.

\vspace{2ex}

\begin{pycode}
n = 13

def trianglePascal(n):
    T = [[0] * (n+1) for p in range(n+1)]
    for n in range(n+1):
        if n == 0:
            T[n][0] = 1
        else:
```

```
            for k in range(n+1):
                if k == 0:
                    T[n][0] = 1
                else:
                    T[n][k] = T[n-1][k-1] + T[n-1][k]
    return T

T = trianglePascal(n)

print('\\begin{tabular}{|>{\\columncolor{black!10}}c|*{',n,'}{c!{\\dashed}}c|}
 ↪  \\rowcolor{black!10}\\hline\\diagbox[height=8mm]{$n$}{$k$}')
for k in range(n+1):
    print('&',k)
for j in range(n+1):
    print(r'\\ \hline ',j)
    for k in range(n+1):
        if k == 0:
            print('&1')
        else:
            if T[j][k] != 0:
                print('&',T[j][k])
            else:
                print('&')
print(r'\\ \hline \end{tabular}')
\end{pycode}
```

Ce qui donne :

Sortie LaTeX

Voici le triangle de Pascal qui affiche les $\binom{n}{k}$ pour n variant de 0 à 13.

n \ k	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1													
1	1	1												
2	1	2	1											
3	1	3	3	1										
4	1	4	6	4	1									
5	1	5	10	10	5	1								
6	1	6	15	20	15	6	1							
7	1	7	21	35	35	21	7	1						
8	1	8	28	56	70	56	28	8	1					
9	1	9	36	84	126	126	84	36	9	1				
10	1	10	45	120	210	252	210	120	45	10	1			
11	1	11	55	165	330	462	462	330	165	55	11	1		
12	1	12	66	220	495	792	924	792	495	220	66	12	1	
13	1	13	78	286	715	1287	1716	1716	1287	715	286	78	13	1

10 Générer plusieurs sujets

10.1 Le meilleur arrive!

Le package pythontex permet, à partir d'un source LaTeX dans lequel on a inséré un script PYTHON appelant la librairie *random*, de créer plusieurs versions d'un même sujet identique dont les données sont générées aléatoirement. Le script qui suit, enregistré dans le fichier « `Plusieurs.sh` », à exécuter en console sous le shell `bash`[6], permet d'illustrer cette idée en créant plusieurs versions différentes à partir d'une même source LaTeX présent dans le même répertoire.

Script Bash

```
#!/bin/bash

# On utilise deux répertoires nommés SUJETS et CORRIGES
# comme lieu de stockage temporaire.
# Si les répertoires existent, on les renomme le temps
# d'exécution du script à l'aide d'une marque horaire.
timestamp=$(date +%s)
if [ -e "SUJETS" ]; then
    mv SUJETS SUJETS-$timestamp
fi
if [ -e "CORRIGES" ]; then
    mv CORRIGES CORRIGES-$timestamp
fi
mkdir SUJETS
mkdir CORRIGES

# Paramètres du script
read -p "Nom du source LaTeX ? " fichier
read -p "Nombre de pages de l'énoncé ? " enonce
read -p "Nombre de sujets ? " nb

# Boucle pour le nombre de sujets demandés.
for i in `seq 1 $nb`; do
    # On copie le source LaTeX puis on effectue les trois compilations
    # nécessaires pour obtenir le PDF contenant sujet et corrigé.
    cp $fichier.tex exemplaire$i.tex
    pdflatex exemplaire$i.tex
    pythontex exemplaire$i.tex
    pdflatex exemplaire$i.tex
    # On utilise pdftk pour séparer l'énoncé du corrigé associé et les
    # placer dans les dossiers ad-hoc.
    pdftk exemplaire$i.pdf cat 1-$enonce output SUJETS/sujet$i.pdf
    pdftk exemplaire$i.pdf cat $((1+$enonce))-end output CORRIGES/corrige$i.pdf
    # On supprime les fichiers générés par les compilations.
    rm -rf pythontex-files-exemplaire$i
    rm exemplaire$i.*
    # Fichiers d'extensions 'aux', 'log', 'pdf', 'pytxcode', 'tex'
    # et éventuellement 'out'.
done
```

6. Dans le cas d'un système d'exploitation de type GNU/Linux.

```
# pdftk permet de concaténer les énoncés ainsi que les corrigés associés pour
# n'avoir que deux pdf à imprimer si besoin.
pdftk SUJETS/*.pdf cat output Enonces-$fichier.pdf
pdftk CORRIGES/*.pdf cat output Corriges-$fichier.pdf

# On efface les fichiers et répertoires temporaires
rm -rf SUJETS CORRIGES

# Si nécessaire, on rétablit les anciens noms des répertoires
if [ -e "SUJETS-$timestamp" ]; then
    mv SUJETS-$timestamp SUJETS
fi
if [ -e "CORRIGES-$timestamp" ]; then
    mv CORRIGES-$timestamp CORRIGES
fi
```

Pour les lecteurs travaillant sous Window$, on peut procéder de façon similaire avec un script Powershell.

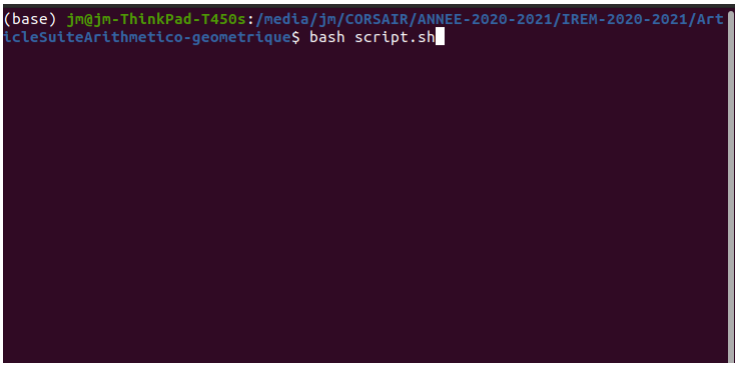
Script PowerShell

```
# Suppression de l'affichage des commandes exécutées.
Set-PSDebug -off

# Efface la console
Clear-Host

# On utilise deux répertoires comme lieux de stockage temporaire pour les énoncés et
 ↪  les corrigés.
# On ajoute un horodatage pour s'assurer de ne pas utiliser de répertoires
 ↪  existants.
$marque=Get-Date -Format FileDateTimeUniversal
$sujets="SUJETS-"+$marque
$corriges="CORRIGES-"+$marque
mkdir $sujets
mkdir $corriges

# Paramètres du script
$fichier=Read-Host "Nom du source LaTeX ? (sans extension) "
[int]$n=Read-Host "Nombre de pages de l'énoncé ? "
[int]$m=$n+1
[int]$nb=Read-Host "Nombre de sujets ? "

# Boucle pour le nombre de sujets demandés.
for ($i=1;$i -le $nb;$i++)
{
    Write-Host "Génération du sujet"$i

    # préparation
    $source="exemplaire"+$i+".tex"
    $resultat="exemplaire"+$i+".pdf"
    copy ($fichier+".tex") $source
```

```
    # chaîne de compilations
    pdflatex -shell-escape -interaction=nonstopmode $source
    pythontex $source
    pdflatex -shell-escape -interaction=nonstopmode $source

    # découpage
    pdftk $resultat cat 1-$n output ($sujets+"\sujet"+$i+".pdf")
    pdftk $resultat cat $m-end output ($corriges+"\corrige"+$i+".pdf")

    # nettoyage des fichiers de compilation
    Remove-Item -Recurse pythontex-files-exemplaire$i
    Remove-Item exemplaire$i*
}

# Assemblages
# - un premier PDF pour tous les énoncés
# - un deuxième PDF pour tous les énoncés
pdftk ($sujets+"\*") cat output ("Enonces-"+$fichier+".pdf")
pdftk ($corriges+"\*") cat output ("Corriges-"+$fichier+".pdf")

# Nettoyage final
Remove-Item -Recurse $sujets
Remove-Item -Recurse $corriges

Write-Host "Terminé !"
```

- Script bash
  Lien de téléchargement : https://nuage03.apps.education.fr/index.php/s/eX3KXnHM2Fzjd5j
- Script PowerShell
  Lien de téléchargement : https://nuage03.apps.education.fr/index.php/s/YpfXC6y7pqazDyT

En console :



10.2 Exemple 1 : Méthode de Héron pour approcher la racine carrée d'un nombre

Pour obtenir les sujets et corrigés associés, on peut utiliser le souce LaTeX « `RacineCarree.tex` »
Lien de téléchargement : https://nuage03.apps.education.fr/index.php/s/m2iriFS3bpdQiNq
Les compilations donnent

- pour les sujets
  https://nuage03.apps.education.fr/index.php/s/yRd3ixApqKb69iH
- pour les corrigés
  https://nuage03.apps.education.fr/index.php/s/X2SXrAPXzrFYTCr

Que contient le fichier source « `RacineCarree.tex` » ?

Il permet d'étudier une suite récurrente $(u_n)_{n\in\mathbb{N}}$ définie par :

$$\begin{cases} u_0 \in \mathbb{R}^* \\ \text{Pour tout entier naturel } n,\ u_{n+1} = \dfrac{1}{2}\left(u_n + \dfrac{a}{u_n}\right) \end{cases}$$

où u_0 est fixé et a est généré aléatoirement à l'aide de la librairie *random*, plus particulièrement de la fonction `randint`.
L'intérêt est de pouvoir créer autant de versions d'énoncés et de corrigés associés qu'il y aura de nombres a générés.
Ces corrigés étudient aussi les vitesses de convergences pour trois méthodes algorithmiques de recherche d'un zéro d'une fonction.

10.3 Exemple 2 : Un sujet du DNB de 2024

Pour obtenir les sujets et corrigés associés, on peut utiliser le souce LaTeX « `DNB_Martinique2024.tex` »
Lien de téléchargement : https://nuage03.apps.education.fr/index.php/s/k6HNcqnieWHxLS4
Les compilations donnent

- pour les sujets
  https://nuage03.apps.education.fr/index.php/s/ZLLWzKNJCeaX24M
- pour les corrigés
  https://nuage03.apps.education.fr/index.php/s/RNFwRyGAPRkeE6K

Que contient le fichier source « `DNB_Martinique2024.tex` » ?

Il contient quatre des cinq exercices du sujet originel aléatoirisés. Trois des exercices ont pour thèmes l'arithmétique, la géométrie plane et les programmes de calcul. Le quatrième est un QCM.

Chaîne de compilations spécifique

Le source « `DNB_Martinique2024.tex` » contient des scripts écrits dans le langage pstricks.
Il faudra donc remplacer les trois compilations successives `pdflatex`, `pythontex` puis `pdflatex` par les compilations
`latex`, `pythontex`, puis `latex` et enfin `dvips`, `ps2pdf`.
Pour générer plusieurs sujets, on pourra exécuter sous bash le script « `Plusieurs_DNB.sh` » qui prend en compte cette nouvelle chaîne de compilations De nouveau, on peut en produire une version PowerShell.

- Script bash
  Lien de téléchargement : https://nuage03.apps.education.fr/index.php/s/trFcmyqjTPYMrgK
- Script PowerShell
  Lien de téléchargement : https://nuage03.apps.education.fr/index.php/s/iM5e4qmQFL8dG78

10.4 Exemple 3 : Une évaluation en BTS SIO sur la numération et la logique

Pour obtenir les sujets et corrigés associés, on peut utiliser le souce LaTeX « `Logique-Numeration.tex` »
Lien de téléchargement : https://nuage03.apps.education.fr/index.php/s/e4N3LPtYc4mLYwi
Les compilations donnent

- pour les sujets
  https://nuage03.apps.education.fr/index.php/s/KD7MjpQZgN3oNWR
- pour les corrigés
  https://nuage03.apps.education.fr/index.php/s/pmsZwH8b6qJGTJF

Que contient le fichier source « `Logique-Numeration.tex` » ?

Il propose une évaluation sur des conversions aléatoirisées Binaire → Décimal et Décimal → Binaire, ainsi que l'étude du connecteur de *Sheffer*.

10.5 exemple 4 : Un exercice type BAC sur les probabilités conditionnelles et la loi binomiale

Pour obtenir les sujets et corrigés associés, on peut utiliser le souce LaTeX « `Probabilites.tex` »
Lien de téléchargement : https://nuage03.apps.education.fr/index.php/s/LeMHN8oraddmGXH
Les compilations donnent

- pour les sujets
  https://nuage03.apps.education.fr/index.php/s/ySJZ5NARTfJ5Qd7
- pour les corrigés
  https://nuage03.apps.education.fr/index.php/s/FCaCsgK9CtdBDgq

Que contient le fichier source « `Probabilites.tex` » ?

Dans cet exercice, les proportions initiales sont aléatoirisées. Par conséquent, la probabilité de succès pour la variable aléatoire suivant une loi binomiale l'est également.