pynotebook (0.1.3), with piton and pyluatex

1 Preamble

```
\documentclass{article}
\usepackage{pynotebook}
\usepackage[executable=python]{pyluatex}  % with a specific compilation !!
```

2 With gobble

Due to `gobble` options with piton, it's possible to add `gobble` parameters to the environments, given within last argument between `<...>`, and default is `empty` :

- `<gobble=xx>` ;
- `<env-gobble>` ;
- `<auto-gobble>` ;
- `<tabs-auto-gobble>`.

Explanations are given in the doc of piton :

- `https://ctan.org/pkg/piton`

3 Examples of text blocks

```
\begin{NotebookPitonMarkdown}{\linewidth}
{\Large\bfseries This is a test for a \textsf{Markdown} block.}

It's possible to use \LaTeX{} formulas, like %
\[
  \left\lbrace\begin{array}{l}
    F_0 = 0\\
    F_1 = 1 \\
    F_{n+2} = F_{n+1} + F_n
  \end{array}\right.
\]
\end{NotebookPitonMarkdown}

\begin{NotebookPitonRaw}{\linewidth}
This is a sample block, with RAW output.

Just to use all capacities of Jupyter notebook ;-)
\end{NotebookPitonRaw}
```

This is a test for a **Markdown** block.

It's possible to use LaTeX formulas, like

$$\left\lbrace\begin{array}{l} F_0 = 0 \\ F_1 = 1 \\ F_{n+2} = F_{n+1} + F_n \end{array}\right.$$

```
This is a sample block, with RAW output.

Just to use all capacities of Jupyter notebook ;-)
```

4 Examples of code blocks (with execution of code !)

4.1 With block In then block Out

```
\begin{NotebookPitonIn}{0.75\linewidth}
def fibonacci_aux(n,a,b):
  if n == 0 :
    return a
  elif n == 1 :
    return b
  else:
    return fibonacci_aux(n-1,b,a+b)

def fibonacci_of(n):
  return fibonacci_aux(n,0,1)

[fibonacci_of(n) for n in range(10)]
\end{NotebookPitonIn}
```

In [1]:

```
def fibonacci_aux(n,a,b):
    if n == 0 :
        return a
    elif n == 1 :
        return b
    else:
        return fibonacci_aux(n-1,b,a+b)

def fibonacci_of(n):
    return fibonacci_aux(n,0,1)

[fibonacci_of(n) for n in range(10)]
```

```
\begin{NotebookPitonOut}{0.75\linewidth}
def fibonacci_aux(n,a,b):
  if n == 0 :
    return a
  elif n == 1 :
    return b
  else:
    return fibonacci_aux(n-1,b,a+b)

def fibonacci_of(n):
  return fibonacci_aux(n,0,1)

print([fibonacci_of(n) for n in range(10)])
\end{NotebookPitonOut}
```

Out [1]:

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

```
\SetJupyterLng{fr}
\SetJupyterParSkip{\baselineskip}
\setcounter{JupyterIn}{11}
```

```
\begin{NotebookPitonIn}[center]{0.75\linewidth}
def fibonacci_aux(n,a,b):
  if n == 0 :
    return a
  elif n == 1 :
    return b
  else:
    return fibonacci_aux(n-1,b,a+b)

def fibonacci_of(n):
  return fibonacci_aux(n,0,1)

print([fibonacci_of(n) for n in range(10)])
\end{NotebookPitonIn}
```

Entrée[12]:

```
1  def fibonacci_aux(n,a,b):
2      if n == 0 :
3          return a
4      elif n == 1 :
5          return b
6      else:
7          return fibonacci_aux(n-1,b,a+b)
8
9  def fibonacci_of(n):
10     return fibonacci_aux(n,0,1)
11
12 print([fibonacci_of(n) for n in range(10)])
```

```
\begin{NotebookPitonOut}[center]{0.75\linewidth}
def fibonacci_aux(n,a,b):
  if n == 0 :
    return a
  elif n == 1 :
    return b
  else:
    return fibonacci_aux(n-1,b,a+b)

def fibonacci_of(n):
  return fibonacci_aux(n,0,1)

print([fibonacci_of(n) for n in range(10)])
\end{NotebookPitonOut}
```

Sortie[12]:

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

```
\begin{NotebookPitonConsole}[center]{0.75\linewidth}
def fibonacci_aux(n,a,b):
  if n == 0 :
    return a
  elif n == 1 :
    return b
  else:
    return fibonacci_aux(n-1,b,a+b)

def fibonacci_of(n):
  return fibonacci_aux(n,0,1)

print([fibonacci_of(n) for n in range(10)])
\end{NotebookPitonConsole}
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

4.2 With block In/Out

```
\begin{NotebookPitonInOut}{0.75\linewidth}
def fibonacci_aux(n,a,b):
  if n == 0 :
    return a
  elif n == 1 :
    return b
  else:
    return fibonacci_aux(n-1,b,a+b)

def fibonacci_of(n):
  return fibonacci_aux(n,0,1)

print([fibonacci_of(n) for n in range(10)])
\end{NotebookPitonInOut}
```

In [1]:
```
1  def fibonacci_aux(n,a,b):
2      if n == 0 :
3          return a
4      elif n == 1 :
5          return b
6      else:
7          return fibonacci_aux(n-1,b,a+b)
8
9  def fibonacci_of(n):
10     return fibonacci_aux(n,0,1)
11
12 print([fibonacci_of(n) for n in range(10)])
```

Out [1]:
```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

4.3 Alternate environment for In/Out

Thanks to F. Pantigny, an alternate environment for `In/Out` is available, with *all* line numbers and continuation symbol.

```
\begin{NotebookPitonAllNum}{0.66\linewidth}
print([i**2 for i in range(50)])
\end{NotebookPitonAllNum}
```

In [2]:
```
1 print([i**2 for i in range(50)])
```

Out [2]:
```
1 [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]
```

5 Global example

This is a test for a **Markdown** block.

It's possible to use LaTeX formulas, like

$$\begin{cases} F_0 = 0\,;\ F_1 = 1 \\ F_{n+2} = F_{n+1} + F_n \end{cases}$$

```
This is a sample block, with RAW output.
Just to use all capacities of Jupyter notebook ;-)
```

In [1]:
```
def fibonacci_aux(n,a,b):
    if n == 0 :
        return a
    elif n == 1 :
        return b
    else:
        return fibonacci_aux(n-1,b,a+b)

def fibonacci_of(n):
    return fibonacci_aux(n,0,1)

print([fibonacci_of(n) for n in range(10)])
```

Out [1]:
```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

```
Let's compute Fibonacci terms from 10th to 20th :-)
```

In [2]:
```
[fibonacci_of(n) for n in range(10,21)]
```

```
[55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765]
```

```
Let's work with an other function.
This time in french :-)
```

In [3]:
```
def calculPerimetre(cote1, cote2, cote3) :
    perimetre = cote1 + cote2 + cote3
    return perimetre

perimetre1 = calculPerimetre(6, 4, 3)
perimetre2 = calculPerimetre(10, 3, 11)
print(f"Le périm de mon 1er triangle est {perimetre1}, et celui de mon 2d est {perimetre2}.")
```

Out [3]:
```
Le périm de mon 1er triangle est 13, et celui de mon 2d est 24.
```

In [4]:
```
A = 15
B = 10
C = 11
print(f"Le périmètre de mon triangle est {calculPerimetre(A,B,C)}.")
```

Out [4]:
```
Le périmètre de mon triangle est 36.
```

In [5]:
```
calculPerimetre(4, 4, 4)
```

```
12
```

In [6]:
```
print([i**2 for i in range(50)])
```

Out [6]:
```
[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]
```