1 Preamble

Litrepl uses its own internal parser to determine LaTeX-looking tags defining code and result sections. As one option, it expects `python` environment to mark code and `result` wrapping results. Both tags need to be introduced to LaTeX so it doesn't get confused. The following preamble sets both environments to be rendered as framed boxes of fixed-width text with proper highlighting:

```
\newenvironment{python}
  {\VerbatimEnvironment
   \begin{minted}[breaklines,fontsize=\footnotesize]{python}}
  {\end{minted}}
\BeforeBeginEnvironment{python}{
  \begin{mdframed}[nobreak=true,frametitle=\tiny{Python}]}
\AfterEndEnvironment{python}{\end{mdframed}}

\newenvironment{result}{\verbatim}{\endverbatim}
\BeforeBeginEnvironment{result}{
  \begin{mdframed}[frametitle=\tiny{Result}]\footnotesize}
\AfterEndEnvironment{result}{\end{mdframed}}
```

2 Basic evaluation

```
\begin{python}
W='Hello, World!'
print(W)
\end{python}
```

Python

```
W='Hello, World!'
print(W)
```

Putting the cursor on it and typing the `:LEval` runs the code in the background Python interpreter.

`result` begin/end tags mark the result section. LitREPL replaces its content with the above code section's execution result.

```
\begin{result}
Hello, World!
\end{result}
```

Result

```
Hello, World!
```

3 Producing LaTeX

LitREPL also recognizes `result`/`noresult` comments as result section markers. With their help we can directly produce LaTeX markup as output:

```
\begin{python}
print("\\textbf{Hi!}")
\end{python}

%result
\textbf{Hi!}
%noresult
```

```
Python

print("\\textbf{Hi!}")
```

Hi!

4 Inline output

Additionally, VimREPL recognises `linline` 2-argument tags. The first arguement is treaten as a Python printable expression. The second arguemnt is to be replaced with its value.

The value of `W` is happen to be: Hello, World!