1 Preamble

Literpl employs its own internal parser to identify LaTeX-like tags that define code and result sections. One option is to use the `python` environment to mark code and wrap results with `result`. Both environments need to be defined in LaTeX so it doesn't get confused. The following preamble sets up both environments to render 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

Here is how we can use the environments we just introduced.

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

Python

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

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

Result

```
Hello, World!
```

3 Producing LaTeX

Literpl also recognizes `result` and `noresult` comments, which serve as markers for result sections. By using these comments, we can generate LaTeX markup directly as output.

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

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

Python

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

Hi!

4 Inline output

Furthermore, Literpl recognizes `linline` tags with two arguments. The first argument is treated as a Python printable expression, while the second argument will be replaced with its evaluated value.

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