PythonTeX Quickstart

github.com/gpoore/pythontex

Compiling

Compiling a document that uses PythonTeX involves three steps: run LaTeX, run `pythontex.py`, and finally run LaTeX again. You may wish to create a symlink or launching wrapper for `pythontex.py`, if one was not created during installation. PythonTeX is compatible with the pdfLaTeX, XeLaTeX, and LuaLaTeX engines. There are minor engine-specific differences.

Commands

`\py` returns a string representation of its argument. For example, `\py{2 + 4**2}` produces 18, and `\py{'ABC'.lower()}` produces abc. `\py`'s argument can be delimited by curly braces, or by a matched pair of other characters (just like `\verb`).

`\pyc` executes code. By default, anything that is printed is automatically included in the document (see `autoprint`/`autostdout` in the main documentation). For example, `\pyc{var = 2}` creates a variable, and then its value may be accessed later via `\py{var}`: 2.

`\pyb` executes and typesets code. For example, `\pyb{var = 2}` typesets `var = 2` in addition to creating the variable. If anything is printed, it is not automatically included, but can be accessed via `\printpythontex` and `\stdoutpythontex`.

`\pyv` only typesets code. For example, `\pyv{var = 2}` produces `var = 2`.

Environments

There are `pycode`, `pyblock`, and `pyverbatim` environments, which are the environment equivalents of `\pyc`, `\pyb`, and `\pyv`. For example,

```
\begin{pycode}
print(r'\begin{center}')
print(r'\textit{A message from Python!}')
print(r'\end{center}')
\end{pycode}
```

produces

A message from Python!

There is also a `pyconsole` environment that emulates a Python interactive console. For example,

```
\begin{pyconsole}
var = 1 + 1
var
\end{pyconsole}
```

yields

```
>>> var = 1 + 1
>>> var
2
```

The `\begin` and `\end` of an environment should be on lines by themselves. Code in environments may be indented; see the `gobble` option in the documentation for more details.

Macro programming

PythonTeX commands can be used inside other commands in macro programming. They will usually work fine, but curly braces should be used as delimiters and special LaTeX characters such as `%` and `#` should be avoided in the Python code. PythonTeX environments cannot be used inside LaTeX commands, due to the way LaTeX deals with verbatim content and catcodes.

Additional features

PythonTeX provides many additional features. The working and output directories can be specified. The user can determine when code is executed with the package option `rerun`, based on factors such as modification and exit status. By default, all commands and environments run in a single session, providing continuity. Commands and environments accept an optional argument that specifies the session in which the code is executed; sessions run in parallel. PythonTeX provides a utilities class that is always imported into each session. The utilities class provides methods for tracking dependencies and automatically cleaning up created files.

PythonTeX also provides the `depythontex` utility, which creates a copy of a document in which all PythonTeX commands and environments have been replaced by their output. The resulting document is more suitable for journal submission, sharing, and conversion to other document formats.

Extended characters

PythonTeX supports extended characters under all LaTeX engines. For example, consider the following example from Python:

```
my_string = '¥ § ß Ğ Ð Ñ Ö þ ø'
```

This requires some engine-specific packages.

- Under pdfLaTeX, your documents need

  ```
  \usepackage[T1]{fontenc}
  \usepackage[utf8]{inputenc}
  ```

  or a similar configuration.

- Under LuaLaTeX, your documents need

  ```
  \usepackage{fontspec}
  ```

  or a similar configuration.

- Under XeLaTeX, your documents need

  ```
  \usepackage{fontspec}
  \defaultfontfeatures{Ligatures=TeX}
  ```

  or a similar configuration.

If you are using Python 2, you will also need to specify that you are using Unicode. You may want

```
from __future__ import unicode_literals
```

at the beginning of your Python code. Or you can just load the PythonTeX package with the option `pyfuture=all`, which will import `unicode_literals` automatically.