y^{Ex}: a TEX-alike typesetter in Python

Marnanel Thurman*

Abstract

yex is an implementation of the core TEX system in pure Python. This article gives an overview of its development, the challenges faced, and possible future directions for the project.

1 Introduction

y^{Ex} is a TEX emulation written in pure Python. It aims to be as faithful a recreation of the TEX core as is possible. It has a strong test suite and plentiful inline documentation.

While I began the project as a means of learning TEX better, it has grown beyond that in every direction and continues to be an ongoing project. Because of this origin, I'm using *The TEXbook* as a spec rather than working from the WEB sources.

Any reimplementation, especially one which reimplements a compiled system in an interpreted language, necessarily has a set of goals which differ from the original: this serves to add diversity and robustness to the TEX ecosystem. Here are mine.

Firstly, y^{Ex} aims to get things right before aiming for completeness: that is, it values *depth over breadth*. The most commonly-used functionality is implemented, and it can set basic documents successfully.

Secondly, Python has a rich *existing infrastructure*: a wonderful resource. y^{Ex} is making as much use of that as possible. PDF handling can be handled by Python's existing PDF libraries; Markdown will be handled likewise; the library Beautiful Soup takes care of XML and HTML handling; and so on. y^{Ex} itself is already available in the Python package index.[1]

Where possible, parts of y^{Ex} will be split out into their own general-purpose libraries, so that they may better interact with other people's projects. It's a fine thing to create something beautiful. It's a thousand times better to build a source of creativity for others.

Lastly, *HTML output* is a particular focus of y^{Ex}. Although TEX is well-suited for producing this format, it is surprisingly underused. More on that shortly.

2 Where are we so far?

- We can set documents!
- SVG output, for debugging
- Serialisation
- Basic HTML output

Next steps:

- Full coverage of TEX controls
- PDF output
- Caching

3 Serialisation

All of y^{Ex}'s internal data structures can be serialised to simple JSON types, as can all processed documents. There are command line switches to output these formats. This is useful in debugging, as well as in integrating third-party systems outside Python.

This also enables caching. At present, for example, `plain.tex` takes several seconds to process. This would make it impractical to use as a library. However, with serialisation, it can be stored and retrieved at speed.

4 Docstrings

Python classes and functions can be documented inline, using a literate programming feature known as "docstrings". These can be formatted using a markup system, such as Restructured Text or Markdown. The docstrings are converted to HTML by a tool called Sphinx, for user help systems. Examples can be seen on sites such as Read The Docs.[2]

y^{Ex} will include a new Sphinx plugin to accept TEX formatting, so that y^{Ex} can produce its own documentation.

5 Input filters

It will also be possible to add input filters for other formatting systems to y^{Ex} itself, so that y^{Ex} documents can include inline HTML or Markdown. Each format will have a stylesheet of macros, one per tag type. The macros will be called to handle each tag, to represent their meaning to y^{Ex}.

6 HTML output

The current focus of development is HTML output. One of the historical barriers to HTML output from TEX has been wordwrap. HTML is built to reflow text on the fly — shoddily, compared to TEX's characteristically careful breaking of paragraphs into lines.

However, modern HTML toolkits such as Bootstrap use a different approach. They divide available display devices into "breakpoint"[3] classes, based on the viewport width: small (for devices such as mobile phones), medium (for laptops), large, and extra

* Thanks to Kit Thurman for proofreading.

[1] pypi.org/project/yex/

[2] See, for example, yex.readthedocs.io.

[3] See getbootstrap.com/docs/5.2/layout/breakpoints. This term "breakpoint" is unrelated to TEX's use of the word.

doi.org/10.47397/tb/43-2/tb134thurman-yex

large. This permits pages to adapt according to the device in use, an ability known as "responsiveness". The system exists for the sake of more complex formatting than merely wordwrap, but it suits yEx's purposes well.

This allows us to add an `\everypar` rule to the HTML output stylesheet, causing each paragraph to be processed four times, each with a different `\hsize`:

```
\let\endgraf=\par \let\endline=\cr
\def\wip##1\par{\let\widthspara=\relax
\special{html.responsive.start}\endgraf
\hsize=432pt
##1\endgraf
##\special{html.responsive.again}\hsize=576pt
##1\endgraf
##\special{html.responsive.again}\hsize=744pt
##1\endgraf
##\special{html.responsive.again}\hsize=992pt
##1\endgraf
\special{html.responsive.done}%
\let\widthspara=\wip}%
\let\widthspara=\wip
\everypar={\widthspara}
```

The `\special` directives tell the output driver to treat these as four versions of the same paragraph. The driver will meld them together, as with Bootstrap breakpoints, so that one is used on mobiles, one on laptops, and so on. The CSS makes the choice of which version to use based on the width of the viewport.

The widths here are taken from Bootstrap's breakpoint specification, where they are given in pixels. yEx allows widths to be specified in pixels (`px`) in addition to TEX's standard units. For interoperability, we give them in points, using W3C's definition of 96 pixels to the inch:[4] this makes a pixel equal to exactly 49152sp.

7 Impedance mismatches

There have been many challenges to overcome so far, even beyond the work of reimplementation of a system as complex as TEX — not to mention the writing of a test suite to prove it all works!

One major factor has been the distance between the priorities of yEx and TEX, which reflects the forty-year distance between them. Unsurprisingly for a program designed in the late 1970s, TEX has a general assumption of scarcity. There are only so many registers of each kind. Python expects you to say what you need and assume that the resources will be found. This difference in approach makes implementation far more interesting. For example, producing call stack traces for errors in TEX macros proved aggravatingly difficult. A TEX macro can be curried by omitting its final argument, thus:

```
\def\a#1#2{Hello, #2}
\def\b{\a x}
{\b world}
```

Because of this, the state of TEX's stack can't be mirrored by the Python stack. Further, any part of the TEX code above might be pushed onto TEX's token stack, and the call stack would still need to remain consistent. The solution involves a special class of token, `Internal`, which runs a given Python callback on being processed. These tokens can't be generated by the tokeniser; they are used for macro prologues and epilogues to maintain the call stack.

Another example is encapsulation: an important principle in Python. TEX is not careful with namespaces. One of the headaches this causes is that the order of loading TEX libraries can easily affect the results. That means that libraries can't be cached individually: the cached value of a library will vary according to which libraries were loaded before it. Thus yEx must wait until it's seen all the initial `\input` commands before making any decisions about caching. When it knows the full list of libraries needed, it must reload them from the cache in order, as a group.

8 The future

yEx's initial goal is to be able to typeset *The TEXbook*. That's still a long way off, though of course the speed at which we get there depends on how many people share the work. Contributions are always welcome! Visit `gitlab.com/marnanel/yex/` for the source.

Beyond that, I have a goal to make sure yEx gives solid results with as many of the packages in the standard TEX distributions as possible. Processing LATEX will be a very important step, though a huge one.

There are also many other TEX-like projects whose ideas we can share, many of which are being discussed at this conference. I look forward to seeing how we can work together.

◇ Marnanel Thurman
`https://gitlab.com/marnanel/yex`

[4] `www.w3.org/Style/Examples/007/units.en.html`