

penlightplus

Additions to the Penlight Lua Libraries

Kale Ewasiuk (kalekje@gmail.com)

2023-07-22

This package first loads the `[import]penlight` package.

The `pl` option may be passed to this package to create an alias for `penlight`.

`globals` option may be used to make several of the functions global (as discussed below).

texlua usage

If you want to use `penlightplus.lua` with the `texlua` interpreter (no document is made, but useful for testing your Lua code), you can access it by setting `__SKIP_TEX__ = true` before loading. For example:

```
package.path = package.path .. ';'..'path/to/texmf/tex/latex/penlightplus/?..lua'
package.path = package.path .. ';'..'path/to/texmf/tex/latex/penlight/?..lua'
penlight = require'penlight'

__SKIP_TEX__ = true --only required if you want to use
                  --penlightplus without a LaTeX run
__PL_GLOBALS__ = true -- optional, include global definitions
require'penlightplus'
```

The following global variables are defined:

`__SKIP_TEX__` If using the `penlightplus` package with `texlua` (good for troubleshooting), set this global before loading `penlight`

The gloals flags below are taken care of in the package options:

`__PL_GLOBALS__` If using package with `texlua` and you don't want to set some globals (described in next sections), set this global before to `true` loading `penlight`

`__PL_NO_HYPERREF__` a flag used to change the behaviour of a function, depending on

if you don't use the hyperref package

penlight additions

Some functionality is added to penlight/luatex.

`pl.hasval(x)` Python-like boolean testing
`COMP'xyz'()` Python-like comprehensions:
<https://lunarmodules.github.io/Penlight/libraries/pl.comprehension.html>
`math.mod(n,d)`, `math.mod2(n)` math modulus
`string.totable(s)` string a table of characters
`string.delspace(s)` clear spaces from string
`pl.char(n)` return letter corresponding to 1=a, 2=b, etc.
`pl.Char(n)` return letter corresponding to 1=A, 2=B, etc.

`pl.utils.filterfiles(dir,filt,rec)` Get files from dir and apply glob-like filters. Set rec to true to include sub directories

A pl.tex. module is added

`add_bkt_cnt(n)`, `close_bkt_cnt(n)`, `reset_bkt_cnt` functions to keep track of adding curly brackets as strings. `add` will return `n` (default 1) `{`'s and increment a counter. `close` will return `n` `}`'s (default will close all brackets) and decrement.
`_NumBkts` internal integer for tracking the number of brackets
`opencmd(cs)` prints `\cs {` and adds to the bracket counters.

`_xNoValue`, `_xTrue`, `_xFalse`: xparse equivalents for commands

`prt(x)`, `prtn(x)` print without or with a newline at end. Tries to help with special characters or numbers printing.
`prt1(l)`, `prtt(t)` print a literal string, or table
`wrt(x)`, `wrtn(x)` write to log
`help_wrt(s1, s2)` pretty-print something to console. S2 is a flag to help you find.
`prt_array2d(tt)` pretty print a 2d array

`pkgwarn(pkg, msg1, msg2)` throw a package warning
`pkgerror(pkg, msg1, msg2, stop)` throw a package error. If stop is true, immediately ceases compile.

`defcmd(cs, val)` like `\gdef`

`newcmd(cs, val)` like `\newcommand`
`renewcmd(cs, val)` like `\renewcommand`
`prvcmd(cs, val)` like `\providecommand`
`deccmd(cs, dft, overwrite)` declare a command. If `dft` (default) is `nil`, `cs` is set to a package warning saying 'cs' was declared and used in document, but never set. If `overwrite` is true, it will overwrite an existing command (using `defcmd`), otherwise, it will throw error like `newcmd`.

`get_ref_info(1)` accesses the `\r @label` and returns a table

global extras

If `extraglobals` is used and NOT `extras`, many additional globals are set for short-cuts

All `pl.tex` modules are made global.

`pl.hasval`, `pl.COMP`, `pl.utils.kpairs`, `pl.utils.npairs` become globals with the function name.

Macro helpers

`\MakeluastringCommands [def]{spec}` will let `\plluastring (A|B|C..)` be `\luastring (N|O|T|F)` based on the letters that `spec` is set to (or `def` if nothing is provided) This is useful if you want to write a command with flexibility on argument expansion. The user can specify `n`, `o`, `t`, and `f` (case insensitive) if they want no, once, twice, or full expansion.

Split stuff

Splitting text (or a cmd) into oxford comma format via: `\splitToComma [expansion level]{text}{text to split on}`:

```

1
2 -\splitToComma{ j doe }\and}-\\
3 -\splitToComma{ j doe \and s else \leftarrow
   }\and}-\\
4 -\splitToComma{ j doe \and s else \leftarrow
   and a per }\and}-\\
5 -\splitToComma{ j doe \and s else \leftarrow
   and a per \and f guy}\and}-
6
7 \def\authors{j doe \and s else \and a\leftarrow
   per \and f guy}
8 \splitToComma[o]{\authors}\and}

```

```

-j doe-
-j doe and s else-
-j doe, s else, and a per-
-j doe, s else, a per, and f guy-
j doe, s else, a per, and f guy

```

The expansion level is up to two characters, `n|o|t|f`, to control the expansion of each argument.

`splitToItems:`

```

1 splitToItems:
2 \begin{itemize}
3   \splitToItems{kale\and john}\and}
4   \splitToItems{kale -john -someone \leftarrow
   else}{-}
5 \end{itemize}

```

- kale
- john
- kale
- john
- someone else