

# penlightplus

## Additions to the Penlight Lua Libraries

Kale Ewasiuk (kalekje@gmail.com)

2023-08-27

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 Lua 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

`__PDFmetadata__` a table used to store PDF meta-data

## penlight additions

Some functionality is added to penlight and Lua.

`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)`, `wrtln(x)` write to log  
`help_wrt(s1, s2)` pretty-print something to console. S2 is a flag to help you find., alias is `wrth`  
`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` , but note that no special chars allowed in `cs`(eg. `@`)  
`defmacro(cs, val)` like `\gdef` , allows special characters, but any tokens in `val` must be pre-defined (this uses `token.set_macro` internally)  
`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 the package option `globals` is used, many additional globals are set for easier scripting. All `pl.tex` functions, and variables, `pl.hasval`, `pl.COMP`, `pl.utils.kpairs`, `pl.utils.npairs` become globals. `pl.tablex` is aliased as `TX` (which also includes all native Lua table functions), and `pl.array2d` is aliased as `A2d`.

## Macro helpers

`\MakeluastringCommands [def]{spec}` will let `\pllustring (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. For example, we can control the expansion of args 2 and 3 with arg 1:

```
\NewDocumentCommand{\splitToComma}{ 0{nn} m m }{%
  \MakeluastringCommands[nn]{#1}%
  \luadirect{penlight.tex.split2comma(\pllustringA{#2},\pllustringB{#3})}%
}
```

## Lua boolean expressions for LaTeX conditionals

`\ifluax {<Lua expr>}{<do if true>}{<do if false>}` and  
`\ifluax {<Lua expr>}{<do if true>}{<do if false>}` for truthy (uses `penlight.hasval`)

1	<code>\ifluax{3^3 == 27}{3*3*3 is 27}[WRONG↔</code>	<code>3*3*3 is 27</code>
	<code>]\</code>	
2	<code>\ifluax{abc123 == nil}{Var is nil}[↔</code>	<code>Var is nil</code>
	<code>WRONG]\</code>	<code>fAlSe</code>
3	<code>\ifluax{not true}{tRuE}[fAlSe]\</code>	<code>TRUE</code>
4	<code>\ifluax{''}{TRUE}[FALSE]\</code>	<code>false</code>
5	<code>\ifluaxv{''}{true}[false]\</code>	

## Creating and using Lua tables in LaTeX

penlightplus provides a Lua-table interface. Tables are stored in the `penlight.tbls` table.

`\newtbl {t}` declares a new table with name `t`  
`\chgtbl {t}` changes the 'recent' table  
`\tblfrkv {t}{key-val string}[luakeys opts]` new table from key-vals, used `luakeys`  
`\settbl {i}{v}` sets a value of the table/index `i` to `v`  
`\gettbl {i}` gets the value and `tex.sprint()`'s it  
`\deftbl {i}{d}` pushes the value to a `cs` named `d`  
`\gdeftbl {i}{d}` pushes the value to a global  
`\iftbl {i}{t}[f]` runs code `t` if the item is true else `f`  
`\iftblv {i}{t}[f]` runs code `t` if the item is truthy else `f`

There are 3 ways to use the index (placeholder `{i}` above). `t.key` where `t` is the table name and `key` is a string key, `t/int` where `int` is a numeric indexer (ie uses `t[int]`), or simply `ind` without the table name, where the assumed table is the last one that was created or changed to, (pure-numbers are not treated as strings in this case).

```
1 \tblfrkv{my}{a,b,c,first=john,last=↵
    smith}[defaults={x=0,1=one,n=false↵ d
    ,y=yes}, naked_as_value=true]      e
2                                     0
3 \settbl{my/4}{d}\gettbl{my/4}\\      f
4 \settbl{5}{e}\gettbl{5}\\           f
5 \gettbl{my.x}\\                     t
6 \iftbl{n}{t}[f]\\                   t
7 \iftblv{n}{t}[f]\\
8 \iftbl{my.y}{t}[f]\\
9 \iftblv{y}{t}[f]\\
```

## Splitting strings

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

```

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

```

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

You can do a similar string split but to `\item` instead of commas with `\splitToItems`

```

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

```

- kale
- john
- kale
- john
- someone else
- 1
- 2
- 3
- 4