

# penlight

Lua libraries for use in LuaLaTeX

Kale Ewasiuk (kalekje@gmail.com)

2022-10-22

Documentation for the Lua library this includes can be found here:

<https://lunarmodules.github.io/Penlight>

This package uses version 1.13.1

## Required Package Option

The first option sent to this package MUST be one of:

[penlight] or [pl].

All Penlight sub-modules are then available under this global variable by either `penlight.XYZ` or `pl.XYZ`

## texlua usage

If you want to use Penlight (and extras) with the `texlua` interpreter (no document made, only for Lua files, useful for testing), you can access it by setting `__SKIP_TEX__ = true` and adding the package to path. For example:

```
package.path = package.path .. ';'..'path/to/texmf/tex/latex/penlight/?..lua'
penlight = require'penlight'
-- below is optional
__SKIP_TEX__ = true --only required if you want to use
                    --penlightextras without a LaTeX run
__PL_GLOBALS__ = true -- optional, include global definitions
__SKIP_LUAKEYS__ = true -- luakeys is loaded in penlightextras by default, you may skip :
require'penlightextras'
```

## Additional Package Options

<code>stringx</code>	will import additional string functions into the string meta table. this will be ran in pre-amble: <code>require('pl.stringx').import()</code>
<code>format</code>	allows <code>%</code> operator for Python-style string formatting this will be ran in pre-amble: <code>require('pl.stringx').format_operator()</code> <a href="https://lunarmodules.github.io/Penlight/libraries/pl.stringx.html#format_operator">https://lunarmodules.github.io/Penlight/libraries/pl.stringx.html#format_operator</a>
<code>func</code>	allows placeholder expressions eg. <code>_1+1</code> to be used this will be ran in pre-amble: <code>penlight.utils.import('pl.func')</code> <a href="https://lunarmodules.github.io/Penlight/libraries/pl.func">https://lunarmodules.github.io/Penlight/libraries/pl.func</a>
<code>extras</code>	does the above three ( <code>func</code> , <code>stringx</code> , <code>format</code> ); adds some additional functions to <code>penlight</code> module; and adds the <code>pl.tex</code> sub-module.
<code>extrasglobals</code>	does the above <code>extras</code> but makes many of the functions global variables as well.

## Extras

If `extras` is used, the following Lua globals will be defined:

## Misc stuff

`__SKIP_TEX__` If using package with `texlua`, 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`  
`__SKIP_LUAKEYS__`  
`__PL_NO_HYPERREF__`  
`__PL_EXTRAS__` false, 1 or 2

`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.  
`kpairs(t)`, `npairs(t)` iterate over keys only, or include nil value from table ipairs

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

### **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.

`prtl(1)`, `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.

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

### **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	<code>-\splitToComma{ j doe }\and}-\\</code>	
3	<code>-\splitToComma{ j doe \and s else \&lt;</code>	
	<code>}\and}-\\</code>	<code>-j doe-</code>
4	<code>-\splitToComma{ j doe \and s else \&lt;</code>	<code>-j doe and s else-</code>
	<code>and a per }\and}-\\</code>	<code>-j doe, s else, and a per-</code>
5	<code>-\splitToComma{ j doe \and s else \&lt;</code>	<code>-j doe, s else, a per, and f guy-</code>
	<code>and a per \and f guy}\and}-</code>	<code>j doe, s else, a per, and f guy</code>
6		
7	<code>\def\authors{j doe \and s else \and a&lt;</code>	
	<code>per \and f guy}</code>	
8	<code>\splitToComma[o]{\authors}{\and}</code>	

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

`spliToItems`:

1	<code>spliToItems:</code>	• kale
2	<code>\begin{itemize}</code>	
3	<code>\splitToItems{kale\and john}{\and}</code>	• john
4	<code>\splitToItems{kale -john -someone &lt;</code>	• kale
	<code>else}{-}</code>	
5	<code>\end{itemize}</code>	• john
		• someone else

## global extras

If `extraglobals` is used and NOT `extras`, many globals are set.

All `pl.tex` modules are made global.

`hasval`, `COMP`, `kpairs`, `npairs` are globals.

Disclaimer: I am not the author of the Lua Penlight library. Penlight is Copyright ©2009-2016 Steve Donovan, David Manura. The distribution of Penlight used for this library is: <https://github.com/lunarmodules/penlight>

The author of this library has merged all Lua sub-modules into one file for this package.