penlightplus

Additions to the Penlight Lua Libraries

Kale Ewasiuk (kalekje@gmail.com)

2023–08–23

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/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 modulous

`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.

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

global extras

If `extrasglobals` is used and NOT `extras`, many additional globals are set for shortcuts
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 insensitve) if they want no, once, twice, or full expansion.

Lua boolean expressions for LaTeX conditionals

`\ifluax {<Lua expr>}{<do if true>}[<do if false>]`

```
\ifluax{3^3 == 27}{3*3*3 is 27}[WRONG]\\
\ifluax{abc123 == nil}{Var is nil}[WRONG]\\
\ifluax{not true}{tRuE}[fAlSe]\\
```

3*3*3 is 27
Var is nil
fAlSe

Creating and using Lua tables in LaTeX

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

For the commands below note that:
`t=table name, v=value, k=key (auto-wrapped in ""), K=key (not wrapped)`

```
\newtbl {t}
\tblfrkv {t}{key-val string}[luakeys opts]
\settbl {t}{k}{v}
\gettbl {t}{k}
\idxtbl {t.k} or \idxtbl {t[1]}
```

A use-case is provided below. You may want to use this interface for setting key-vals in commands.

```
\tblfrkv{my}{x=1.5,y}%
    [defaults={x=0,1=one,n=false}]
\gettbl{my}{x}\\
\settbl{my}{y}{kale}
\gettbl{my}{y}\\
\idxtbl{my.x}\\
\idxtbl{my['x']}\\
\idxtbl{my['1']}\\
\iftbl{my}{n}{true}[false]\\
\iftblv{my}{n}{true}[false]\\
\iftbl{my}{y}{true}[false]\\
\iftblv{my}{y}{true}[false]\\
```

```
1.5
kale
1.5
1.5
nil
false
false
true
true
```

Splitting strings

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

```
-\splitToComma{  j doe  }{\and}-\\
-\splitToComma{  j doe \and s else  ↩
   }{\and}-\\
-\splitToComma{  j doe \and s else \↩
   and a per }{\and}-\\
-\splitToComma{  j doe \and s else \↩
   and a per \and f guy}{\and}-

\def\authors{j doe \and s else \and a↩
    per \and f guy}
\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.

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

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

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