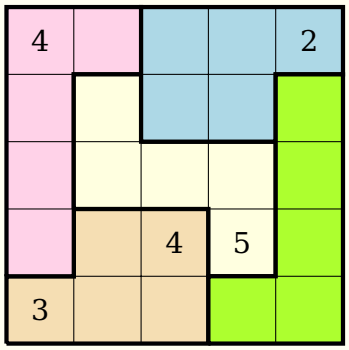
logicpuzzle.sty

v1.8

A style file for typesetting logic puzzles



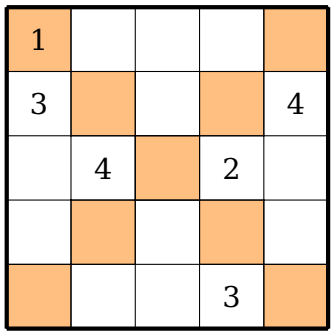
April 10, 2013

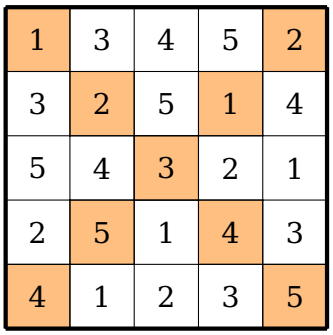
Package author:
Josef Kleber

josef.kleber@gmx.de
https://bitbucket.org/kleberj/logicpuzzle/

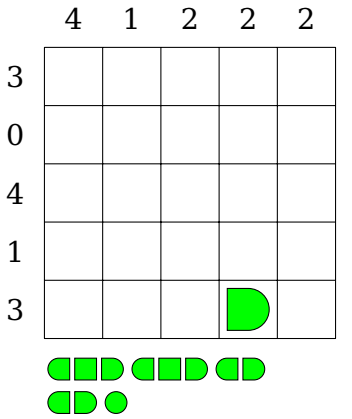
1 Supported puzzles

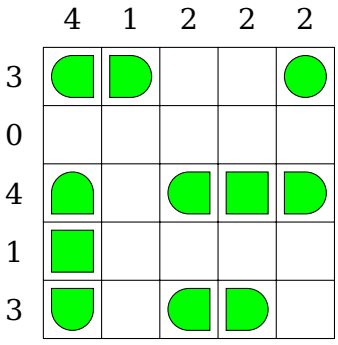
1.1 2D-Sudoku



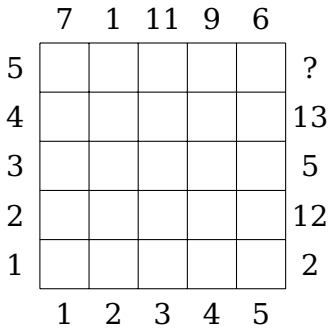


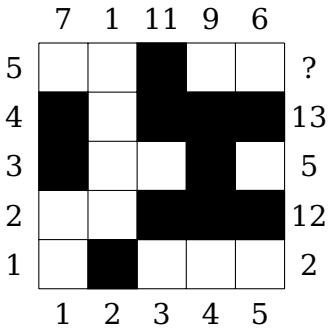
1.2 Battleship



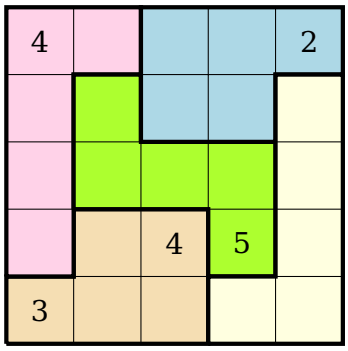


1.3 Bokkusu

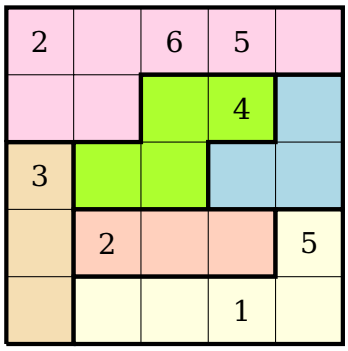


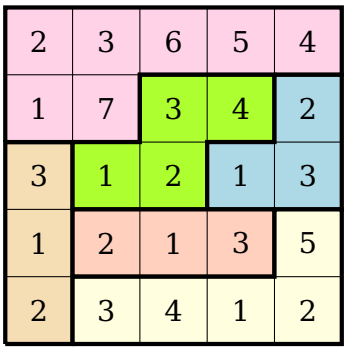


1.4 Chaos Sudoku

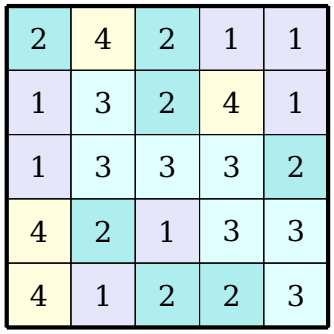


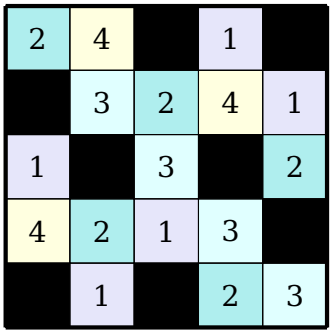
1.5 Hakyuu



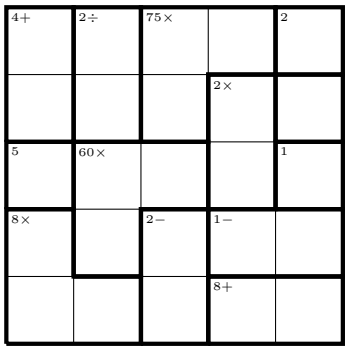


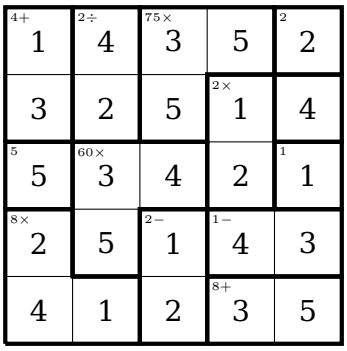
1.6 Hitori



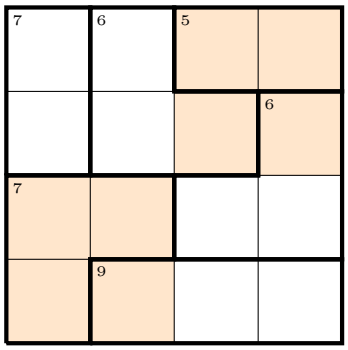


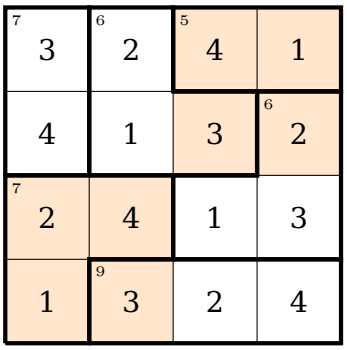
1.7 Kendoku



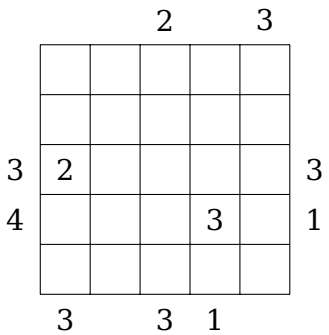


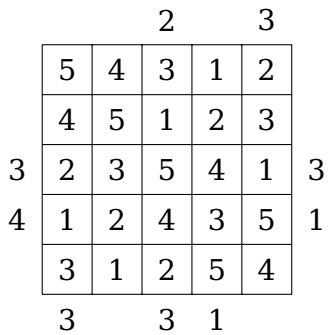
1.8 Killer Sudoku





1.9 Skyline





1.10 Sudoku

	2	6						
						1	7	
		3	1		6			
	6			5		8		3
		9	2	6	1	7		
5		4		8			6	
			8		4	3		
	4	8						
						9	4	

1	2	6	5	7	8	4	3	9
4	8	5	9	3	2	1	7	6
7	9	3	1	4	6	5	8	2
2	6	1	4	5	7	8	9	3
8	3	9	2	6	1	7	5	4
5	7	4	3	8	9	2	6	1
6	5	2	8	9	4	3	1	7
9	4	8	7	1	3	6	2	5
3	1	7	6	2	5	9	4	8

2 Roll out your own grid-based logic puzzle

As an example we take a look at the `bokkusu.sty` package. First, we ignore the LPPL license stuff.

```
\ProvidesPackage{bokkusu}[2013/03/25 bokkusu.sty v1.2 - Josef Kleber (C) 2013]%
\RequirePackage{logicpuzzle}%
```

We wrote a package `bokkusu.sty` with version number v1.2 and date 2013/03/25 and added a copyright remark. We need to load the code base package `logicpuzzle.sty`.

```
\newcommand*\LP@BK@init@prefix{LP@BK}%
\newcommand*\LP@BK@init@package{bokkusu}%
\LP@define@key{\LP@BK@init@prefix}{\LP@BK@init@package}{rows}{5}%
\LP@define@key{\LP@BK@init@prefix}{\LP@BK@init@package}{columns}{5}%
\LP@define@key{\LP@BK@init@prefix}{\LP@BK@init@package}{scale}{1}%
\LP@define@key{\LP@BK@init@prefix}{\LP@BK@init@package}{counterstyle}{none}%
\LP@define@key{\LP@BK@init@prefix}{\LP@BK@init@package}{color}{black}%
\LP@define@key{\LP@BK@init@prefix}{\LP@BK@init@package}{bgcolor}{}%
\LP@define@key{\LP@BK@init@prefix}{\LP@BK@init@package}{width}{6.7cm}%
\LP@define@key{\LP@BK@init@prefix}{\LP@BK@init@package}{cvoffset}{-38pt}%
\LP@define@key{\LP@BK@init@prefix}{\LP@BK@init@package}{title}{}%
\LP@define@key{\LP@BK@init@prefix}{\LP@BK@init@package}{titleindent}{0.75cm}%
\LP@define@key{\LP@BK@init@prefix}{\LP@BK@init@package}{titlewidth}{5.85cm}%
\LP@define@choicekey@fontsize{\LP@BK@init@prefix}{\LP@BK@init@package}{Large}%
\ExecuteOptionsX{rows,columns,width,fontsize,scale,color,bgcolor,cvoffset,
                 counterstyle,title,titleindent,titlewidth}%
\ProcessOptionsX\relax%
\LP@init@counter{\LP@BK@init@prefix}%
```

We save the package prefix and name in a macro for easy change. Then we define the options for package `bokkusu.sty` and the environment `bokkusu`, which are executed afterwards to create the macros for the option values. In the end, we need to initialize the package counters.

```
\let\valueH\LP@bottomrow%
\let\valueV\LP@leftcolumn%
\let\sumH\LP@toprow%
\let\sumV\LP@rightcolumn%
```

We need numbers around the grid. Therefore, we define some aliases for the existing generic commands.

```
\newcommand*\bokkususetup[1]%
{%
  \setkeys{bokkusu.sty}{#1}%
}%
```

We define `\bokkususetup` for resetting the global package options.

Finally, we define the bokkusu environment.

```
\newenvironment{bokkusu}[1][]%
{%
  \setkeys{bokkusu}{#1}%
  \LP@set@package{bokkusu}%
  \LP@set@env@prefix{LP@BK}%
  \setcounter{LP@BK@rows}{\LP@BK@rows}%
  \setcounter{LP@BK@columns}{\LP@BK@columns}%
  \stepcounter{LP@BK@rows}%
  \stepcounter{LP@BK@columns}%
```

We locally set the environment options and the prefix and name of the current puzzle environment. We need to reset the counters for rows and columns, as they might have been altered.

```
  \begin{minipage}[t]{\LP@BK@width}%
    \ifthenelse{\equal{\LP@BK@title}{}}%
    {\par\enspace\par}% empty
    {\enspace\par\noindent\hspace{\LP@BK@titleindent}\parbox{\LP@BK@titlewidth}
      {\strut\LP@titleformat\LP@BK@title}\vspace{3mm}\par}%
    \begin{tikzpicture}[scale=\LP@BK@scale]%
      \LP@drawbackground{1}{1}{\LP@BK@columns}{\LP@BK@rows}{\LP@BK@bgcolor}%
      \LP@drawgrid{1}{1}{\LP@BK@columns}{\LP@BK@rows}{1cm}%
}%
```

We start a minipage with width ⟨*width*⟩. If the user defined a title, we typeset the title and add a vertical space. Then, we draw the puzzle with the help of tikz.sty. We start drawing the background and the grid.

```
{%
    \end{tikzpicture}%
    \LP@drawcounter{\LP@BK@counterstyle}%
    \stepcounter{LP@puzzlecounter}%
  \end{minipage}%
}%
```

Finally, we just end the picture for the puzzle. We draw and step the counter. As last action, we need to close the minipage environment. That's it. Easy, isn't it? You just need to copy this skelton and change or add some code for your specific puzzle.

3 The code

3.1 PGF layers

The logicpuzzle.sty package defines the PGF layers: LPdump, LPbgcolor, LPbackgroundtwo, LPbackground, LPforeground and LPforegroundtwo

Without specifying a special layer, the standard main layer is used. The LPbackground and LPforeground layers can be accessed with the puzzlebackground

[see: 3.2.1] and puzzleforeground [see: 3.2.2] environments. The LPbgcolor is and should only be used for the background color of the grid.

All layers can also be accessed with the generic PGF method:

```
\begin{pgfonlayer}{layer}
  ...
\end{pgfonlayer}{layer}
```

Order: LPdump → LPbgcolor → LPbackgroundtwo → LPbackground → main → LPforeground → LPforegroundtwo

So, if you are in the need to place something behind LPbackground or in front of LPforeground, you can use the LPbackgroundtwo and LPforegroundtwo layers. You can hide elements like help nodes behind the background color on the LPdump layer.

3.2 Environments

3.2.1 puzzlebackground

\begin{puzzlebackground} ... \end{puzzlebackground}

The puzzlebackground environment allows you to place elements behind the main layer on the LPbackground layer [see: 3.1]. This is for example usefull for the \fillarea [see: 3.3.3.15] command.

3.2.2 puzzleforeground

\begin{puzzleforeground} ... \end{puzzleforeground}

The puzzleforeground environment allows you to place elements in front of the main layer on the LPforeground layer [see: 3.1]. This is for example usefull for the \framearea [see: 3.3.3.14] command.

3.3 Commands

3.3.1 Initialization

3.3.1.1 \LP@define@key

\LP@define@key{⟨*prefix*⟩}{⟨*package*⟩}{⟨*option*⟩}{⟨*default*⟩}

With the \LP@define@key command, you can define the options of the package ⟨*package*⟩ **and** of the environment ⟨*package*⟩. A ⟨*prefix*⟩ is needed for creating different name spaces.

```
\LP@define@key{LP@BS}{battleship}{rows}{5}
```

This code snippet defines the option rows as global option for battleship.sty and as local option for environment battleship with the default value 5. This value is stored in \LP@BS@rows.

3.3.1.2 \LP@define@choicekey@fontsize

\LP@define@choicekey@fontsize{⟨prefix⟩}{⟨package⟩}{⟨default⟩}

With the \LP@define@choicekey@fontsize command, you can define the choice key option fontsize of the package ⟨package⟩ **and** of the environment ⟨package⟩. Possible keys are: tiny, scriptsize, footnotesize, small, normalsize, large, Large, LARGE, huge, Huge

3.3.1.3 \LP@init@counter

\LP@init@counter{⟨prefix⟩}

The command \LP@init@counter defines the counters ⟨prefix⟩@rows and ⟨prefix⟩@columns, initialize them with \⟨prefix⟩@rows and \⟨prefix⟩@columns and steps the counters.

3.3.2 Drawing grids

3.3.2.1 \LP@drawgrid

\LP@drawgrid{⟨xmin⟩}{⟨ymin⟩}{⟨xmax⟩}{⟨ymax⟩}{⟨step⟩}

With the \LP@drawgrid command, you can draw the grid (⟨xmin⟩,⟨ymin⟩) to (⟨xmax⟩,⟨ymax⟩) with step ⟨step⟩. For drawing the standard puzzle grid the step must be 1cm.

3.3.2.2 \LP@drawsudokugrid

\LP@drawsudokugrid

The command \LP@drawsudokugrid draws the stnadard Sudoku grid, but just the thicker lines. You will have to overlay the standard grid to get a full Sudoku grid.

3.3.2.3 \LP@drawbackground

\LP@drawbackground{⟨xmin⟩}{⟨ymin⟩}{⟨xmax⟩}{⟨ymax⟩}{⟨color⟩}

With the \LP@drawbackground command, you can draw the background color of the grid.

3.3.3 In the grid

3.3.3.1 \setcell

\setcell{{⟨column⟩}{⟨row⟩}{⟨element⟩}

With the \setcell command, you can set ⟨element⟩ into cell ⟨column⟩⟨row⟩ as central node. It is aware of the current values of the surrounding environment options rows, columns, scale and fontsize. Furthermore, a check if ⟨element⟩ is within the grid is applied.

3.3.3.2 \LP@setcellcontent

\LP@LP@setcellcontent{⟨column⟩}{⟨row⟩}{⟨element⟩}

The command \LP@setcellcontent is the generic command to set an arbitrary ⟨element⟩.

3.3.3.3 \setrow

\setrow{⟨row⟩}{⟨csv list⟩}

With the \setrow command, you can set the contents of a ⟨row⟩. These may be numbers or letters.

3.3.3.4 \LP@setrowcontents

\LP@setrowcontents{⟨csv list⟩} {⟨column⟩}{⟨row⟩}

The command \LP@setrowcontents is the generic command to set row contents. It does not necessarily start with ⟨column⟩ 1!

3.3.3.5 \setcolorrow

\setcolorrow{⟨row⟩}{⟨csv list⟩}

With the \setcolorrow command, you can set the contents of a ⟨row⟩. Furthermore, the background of the cell is filled with color LP@c@romannumber [see: 3.3.3.20]. With the number 0, you can black out the grid cell.

3.3.3.6 \setcolumn

\setcolumn{⟨column⟩}{⟨csv list⟩}

With the \setcolumn command, you can set the contents of a ⟨column⟩. These may be numbers or letters.

3.3.3.7 \LP@setcolumncontents

\LP@setcolumncontents{⟨csv list⟩} {⟨column⟩}{⟨row⟩}

The command \LP@setcolumncontents is the generic command to set column contents. It does not necessarily start with ⟨row⟩ 1!

3.3.3.8 \setcolorcolumn

\setcolorcolumn {⟨column⟩}{⟨csv list⟩}

With the \setcolorcolumn command, you can set the contents of a ⟨column⟩. Furthermore, the background of the cell is filled with color LP@c@romannumber [see: 3.3.3.20].

3.3.3.9 \setrule

\setrule{{⟨column⟩}{⟨row⟩} {⟨rule⟩}

With the \setrule command, you can set a calculation rule ⟨rule⟩ into the top left corner of cell ⟨column⟩⟨row⟩. The rule is typeset in inline math mode. You might consider using the \times and \div commands.

3.3.3.10 \fillcell

\fillcell{{⟨column⟩}{⟨row⟩}

With the \fillcell command, you can fill cell ⟨column⟩⟨row⟩ with the color defined with environment option color[1]. It is aware of the current values of the surrounding envionment options rows, columns, scale and color. Furthermore, a check if the cell is within the grid is applied.

[1] Therefore, you must define an option color in the style file you want to use fill commands

3.3.3.11 \fillrow

\fillrow{⟨row⟩}{⟨csv list⟩} With the \fillrow command, you can fill a ⟨row⟩. In ⟨csv list⟩ '1' means 'fill' and '0' means 'don't fill'. Internally, \fillrow uses \fillcell [see: 3.3.3.10].

3.3.3.12 \fillcolumn

\fillcolumn{⟨column⟩}{⟨csv list⟩} With the \fillcolumn command, you can fill a ⟨column⟩. In ⟨csv list⟩ '1' means 'fill' and '0' means 'don't fill'. Internally, \fillcolumn uses \fillcell [see: 3.3.3.10].

3.3.3.13 \filldiagonals

\filldiagonals[⟨color⟩] With the \filldiagonals command, you can fill the diagonals with the color specified with the optional argument ⟨color⟩ (default: yellow!20). Furthermore, it checks for a quadratic grid, otherwise an error message is issued.

3.3.3.14 \framearea

\framearea{⟨color⟩}{⟨tikz path⟩} The command \framearea frames the area given by ⟨tikz path⟩ with color ⟨color⟩. The reference for coordinates is the bottom left corner of the cell.

```
\framearea{green}{(2,2)--(2,3)--(3,3)--(3,2)--(2,2)}
```

This command will color the frame of the grid cell (2,2) green. You should consider using this command in the puzzleforeground [see: 3.2.2] environment.

3.3.3.15 \fillarea

\fillarea{⟨color⟩}{⟨tikz path⟩} The command \fillarea fills the area given by ⟨tikz path⟩ with color ⟨color⟩. The reference for coordinates is the bottom left corner of the cell. You should consider using this command in the puzzlebackground [see: 3.2.1] environment.

3.3.3.16 \colorarea

\colorarea{⟨color⟩}{⟨tikz path⟩} The command \colorarea fills the area given by ⟨tikz path⟩ with color ⟨color⟩ – just like \framearea without frame.

3.3.3.17 \framepuzzle

\framepuzzle[⟨color⟩] With the \framepuzzle command, you can frame the grid (thicker line) with the color specified with the optional argument ⟨color⟩ (default: black).

3.3.3.18 \tikzpath

\tikzpath{⟨column⟩}{⟨row⟩}{⟨csv list⟩}

With the \tikzpath command, you can easily construct a \tikz path. You just need to define a starting point ⟨column⟩⟨row⟩ (bottom left corner) and a ⟨csv list⟩ with direction indicators relative to the current position.

7:	up left	8:	up	9:	up right
4:	left	5:	no change	6:	right
1:	down left	2:	down	3:	down right

```
\framearea{green}{\tikzpath{2}{2}{8,6,2,4}}
```

This command will frame grid cell (2,2) green.

3.3.3.19 \LP@ingrid

\LP@ingrid{⟨column⟩}{⟨row⟩}{⟨max column⟩}{⟨max row⟩}{⟨package⟩}

With the \LP@ingrid command, you can check if an element – that should be placed – is within the grid. Otherwise an error message is issued.

3.3.3.20 \LP@definecolor

\LP@definecolor{⟨name⟩}{⟨rgb color⟩}

With the \LP@definecolor command, you can define named rgb colors, especially for defining background colors of numbers used in \setcolorrow [see: 3.3.3.5] and \setcolorcolumn [see: 3.3.3.8].

The background color names follow the pattern: LP@c@romannumber

```
\LP@definecolor{LP@c@iv}{.55,1,.88}
```

This command will define the new background color of number 4 !

3.3.4 Around the grid

3.3.4.1 \LP@leftcolumn

\LP@leftcolumn{{⟨csv list⟩}}

With the \LP@leftcolumn command, you can set the contents of the column left to the grid. The skyline.sty package uses for example:

```
\let\skylineL\LP@leftcolumn
```

3.3.4.2 \LP@rightcolumn

\LP@rightcolumn{{⟨csv list⟩}}

With the \LP@rightcolumn command, you can set the contents of the column right to the grid.

3.3.4.3 \LP@toprow

\LP@toprow{{⟨csv list⟩}} With the \LP@toprow command, you can set the contents of the row above the grid.

3.3.4.4 \LP@bottomrow

\LP@bottomrow{{⟨csv list⟩}} With the \LP@bottomrow command, you can set the contents of the row below the grid.

3.3.5 Presentation

3.3.5.1 \titleformat

\titleformat{{⟨format⟩}} With the \titleformat command, you can define the ⟨*format*⟩ of the title. By default, the definition is as follows:

```
\titleformat{\centering\Large\color{blue}}
```

3.3.5.2 \puzzlecounter

\puzzlecounter The command \puzzlecounter provides the counter in textual form to use it for example in \definecounterstyle.

3.3.5.3 \setpuzzlecounter

\setpuzzlecounter{⟨number⟩} With the command \setpuzzlecounter, you can reset the puzzle counter, for example before the solutions.

3.3.5.4 \definecounterstyle

\definecounterstyle{⟨name⟩}{⟨definition⟩} The command \definecounterstyle allows you to define your own styles. For example, the style left is defined as follows:

```
\definecounterstyle{left}{
  \begingroup\reversemarginpar\marginnote{
  \tikz\node[shape=rectangle,fill=yellow!40,inner sep=7pt,
             draw,rounded corners=3pt,thick]
  {\Huge\puzzlecounter};}[\LP@cvoffset]\endgroup}
}
```

To typeset the counter into the margin we use the command \marginnote. We need to use the command \reversemarginpar to set the counter into the left margin. Of course, we must use this command in a group for local scope.

Finally we use `\puzzlecounter` in a `\tikz` node with a vertical offset set with the option `cvoffset`.

3.3.5.5 `\LP@drawcounter`

`\LP@drawcounter{⟨name⟩}` The command `\LP@drawcounter` draws the counter with counter style ⟨*name*⟩.

4 Examples

You can download application examples and their solutions from the project page. The puzzles are originally licensed under CC BY NC SA.