latexindent.pl

Version 3.5.3

Chris Hughes *

October 13, 2018

latexindent.pl is a Perl script that indents .tex (and other) files according to an indentation scheme that the user can modify to suit their taste. Environments, including those with alignment delimiters (such as \texttt{tabular}), and commands, including those that can split braces and brackets across lines, are usually handled correctly by the script. Options for \texttt{verbatim}-like environments and commands, together with indentation after headings (such as \texttt{chapter}, \texttt{section}, etc) are also available. The script also has the ability to modify line breaks, and add comment symbols. All user options are customisable via the switches in the YAML interface; you can find a quick start guide in Section 1.4 on page 8.

Contents

1 Introduction 6
   1.1 Thanks ......................................................... 6
   1.2 License ..................................................... 7
   1.3 About this documentation ................................. 7
   1.4 Quick start .................................................. 8

2 Demonstration: before and after 8

3 How to use the script 9
   3.1 From the command line .................................... 9
   3.2 From arara .................................................. 14

4 indentconfig.yaml, local settings and the \texttt{-y} switch 14
   4.1 indentconfig.yaml and .indentconfig.yaml ................. 14
   4.2 localSettings.yaml .......................................... 15
   4.3 The \texttt{-y|yaml} switch .................................... 16
   4.4 Settings load order ....................................... 16

5 defaultSettings.yaml 17
   5.1 The code blocks known latexindent.pl .................... 31
   5.2 noAdditionalIndent and indentRules ....................... 31
      5.2.1 Environments and their arguments .................... 33
      5.2.2 Environments with items ................................ 39
      5.2.3 Commands with arguments ............................... 40
      5.2.4 ifelsefi code blocks ................................... 42
      5.2.5 specialBeginEnd code blocks ............................ 44
      5.2.6 afterHeading code blocks .............................. 45
      5.2.7 The remaining code blocks .............................. 47
      5.2.8 Summary .................................................. 48
   5.3 Commands and the strings between their arguments ......... 49

* and contributors! See Section 8.2 on page 90. For all communication, please visit [7].
6 The -m (modifylinebreaks) switch
6.1 textWrapOptions: modifying line breaks by text wrapping .......................... 55
  6.1.1 text wrapping on a per-code-block basis .................................................. 57
  6.1.2 Summary of text wrapping ................................................................. 63
6.2 oneSentencePerLine: modifying line breaks for sentences ................................ 63
  6.2.1 sentencesFollow ................................................................. 65
  6.2.2 sentencesBeginWith .......................................................... 66
  6.2.3 sentencesEndWith .......................................................... 66
  6.2.4 Features of the oneSentencePerLine routine ........................................ 68
  6.2.5 text wrapping and indenting sentences ................................................ 69
6.3 removeParagraphLineBreaks: modifying line breaks for paragraphs ................. 71
6.4 Combining removeParagraphLineBreaks and textWrapOptions ....................... 77
6.5 Poly-switches ................................................................. 78
6.6 modifyLineBreaks for environments ......................................................... 79
  6.6.1 Adding line breaks: BeginStartsOnOwnLine and BodyStartsOnOwnLine ...... 79
  6.6.2 Adding line breaks using EndStartsOnOwnLine and EndFinishesWithLineBreak 80
  6.6.3 poly-switches only add line breaks when necessary .................................. 81
  6.6.4 Removing line breaks (poly-switches set to −1) ..................................... 82
  6.6.5 About trailing horizontal space ......................................................... 83
  6.6.6 poly-switch line break removal and blank lines ...................................... 83
6.7 Poly-switches for other code blocks ......................................................... 84
6.8 Partnering BodyStartsOnOwnLine with argument-based poly-switches ............ 86
6.9 Conflicting poly-switches: sequential code blocks ....................................... 87
6.10 Conflicting poly-switches: nested code blocks ........................................... 88

7 Conclusions and known limitations ........................................................ 89

8 References  .......................................................................................... 90
  8.1 External links ................................................................. 90
  8.2 Contributors ................................................................. 90

A Required Perl modules  ........................................................................ 91
  A.1 Module installer script .......................................................... 91
  A.2 Manually installed modules ......................................................... 91

B Updating the path variable  ..................................................................... 92
  B.1 Add to path for Linux .......................................................... 92
  B.2 Add to path for Windows ........................................................ 93

C logFilePreferences ........................................................................... 93

D Differences from Version 2.2 to 3.0 ...................................................... 94

Listings

LISTING 1: demo-tex.tex ............................................................. 7
LISTING 2: fileExtensionPreference .............................................. 7
LISTING 3: modifyLineBreaks .......................................................... 7
LISTING 4: Possible error messages .................................................. 8
LISTING 5: filecontents1.tex .......................................................... 9
LISTING 6: filecontents1.tex default output ...................................... 9
LISTING 7: tikzset.tex ................................................................. 9
LISTING 8: tikzset.tex default output ................................................ 9
LISTING 9: pstricks.tex ............................................................... 9
LISTING 10: pstricks.tex default output .......................................... 9
LISTING 14: fileExtensionPreference ............................................. 17
LISTING 15: logFilePreferences ..................................................... 19
LISTING 16: verbatimEnvironments .............................................. 19
LISTING 17: verbatimCommands .................................................. 19
LISTING 18: noIndentBlock .......................................................... 19
LISTING 19: noIndentBlock demonstration ....................................... 20
LISTING 20: removeTrailingWhitespace .......................................... 20
LISTING 21: Motivating preambleCommandsBeforeEnvironments ........... 21
LISTING 22: fileContentsEnvironments ........................................... 20
LISTING 23: lookForPreamble ........................................................ 20
LISTING 24: tabular1.tex ............................................................. 21
LISTING 25: tabular1.tex default output .......................................... 21
LISTING 28: tabular.yaml ............................................................. 22
LISTING 29: tabular2.tex ............................................................. 22
Introduction

I recently created `latexindent.pl` to help me format chapter files in a big project. After I blogged about it on the *LATEX* Stack Exchange [1] I received some positive feedback and follow-up feature requests. A big thank you to Harish Kumar [9] who helped to develop and test the initial versions of the script.
The YAML-based interface of `latexindent.pl` was inspired by the wonderful `arara` tool; any similarities are deliberate, and I hope that it is perceived as the compliment that it is. Thank you to Paulo Cereda and the team for releasing this awesome tool; I initially worried that I was going to have to make a GUI for `latexindent.pl`, but the release of `arara` has meant there is no need.

There have been several contributors to the project so far (and hopefully more in the future!); thank you very much to the people detailed in Section 8.2 on page 90 for their valued contributions, and thank you to those who report bugs and request features at [7].

1.2 License

`latexindent.pl` is free and open source, and it always will be; it is released under the GNU General Public License v3.0.

Before you start using it on any important files, bear in mind that `latexindent.pl` has the option to overwrite your `.tex` files. It will always make at least one backup (you can choose how many it makes, see page 18) but you should still be careful when using it. The script has been tested on many files, but there are some known limitations (see Section 7). You, the user, are responsible for ensuring that you maintain backups of your files before running `latexindent.pl` on them. I think it is important at this stage to restate an important part of the license here:

*This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.*

There is certainly no malicious intent in releasing this script, and I do hope that it works as you expect it to; if it does not, please first of all make sure that you have the correct settings, and then feel free to let me know at [7] with a complete minimum working example as I would like to improve the code as much as possible.

Before you try the script on anything important (like your thesis), test it out on the sample files in the `test-case` directory [7].

*If you have used any version 2.* of `latexindent.pl`, there are a few changes to the interface; see appendix D on page 94 and the comments throughout this document for details.

1.3 About this documentation

As you read through this documentation, you will see many listings; in this version of the documentation, there are a total of 382. This may seem a lot, but I deem it necessary in presenting the various different options of `latexindent.pl` and the associated output that they are capable of producing.

The different listings are presented using different styles:

<table>
<thead>
<tr>
<th>Listing 1: demo-tex.tex</th>
<th>This type of listing is a <code>.tex</code> file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>demonstration .tex file</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Listing 2: fileExtensionPreference</th>
</tr>
</thead>
<tbody>
<tr>
<td>38 fileExtensionPreference:</td>
</tr>
<tr>
<td>39 .tex: 1</td>
</tr>
<tr>
<td>40 .sty: 2</td>
</tr>
<tr>
<td>41 .cls: 3</td>
</tr>
<tr>
<td>42 .bib: 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Listing 3: modifyLineBreaks</th>
</tr>
</thead>
<tbody>
<tr>
<td>394 modifyLineBreaks:</td>
</tr>
<tr>
<td>395 preserveBlankLines: 1</td>
</tr>
<tr>
<td>396 condenseMultipleBlankLinesInto: 1</td>
</tr>
</tbody>
</table>

This type of listing is a `.yaml` file; when you see line numbers given (as here) it means that the snippet is taken directly from `defaultSettings.yaml`, discussed in detail in Section 5 on page 17.

This type of listing is a `.yaml` file, but it will only be relevant when the `-m` switch is active; see Section 6 on page 53 for more details.

You will occasionally see dates shown in the margin (for example, next to this paragraph!) which detail the date of the version in which the feature was implemented; the ‘N’ stands for ‘new as of the date shown’ and ‘U’ stands for ‘updated as of the date shown’. If you see ‘‡’, it means that the feature...
is either new (N) or updated (U) as of the release of the current version; if you see ✩ attached to a listing, then it means that listing is new (N) or updated (U) as of the current version. If you have not read this document before (and even if you have!), then you can ignore every occurrence of the ✩; they are simply there to highlight new and updated features. The new and updated features in this documentation (V3.5.3) are on the following pages:

- specialBeginEnd verbatim (N) ........................................... 29
- updates to textWrapOptions (U) ...................................... 57
- updates to textWrapOptions (U) ...................................... 63
- oneSentencePerline text wrap and indent (N) ......................... 69
- updates to all in removeParagraphLineBreaks (U) ................. 76
- combine text wrap and remove paragraph line breaks (N) ........ 77

1.4 Quick start
If you'd like to get started with latexindent.pl then simply type

```
cmh:~$ latexindent.pl myfile.tex
```

from the command line. If you receive an error message such as that given in Listing 4, then you need to install the missing perl modules.

```
BEGIN failed--compilation aborted at helloworld.pl line 10.
```

latexindent.pl ships with a script to help with this process; if you run the following script, you should be prompted to install the appropriate modules.

```
cmh:~$ perl latexindent-module-installer.pl
```

You might also like to see https://stackoverflow.com/questions/19590042/error-cant-locate-file-homedir-pm-in-inc, for example, as well as appendix A on page 91.

2 Demonstration: before and after
Let's give a demonstration of some before and after code – after all, you probably won't want to try the script if you don't much like the results. You might also like to watch the video demonstration I made on youtube [17]

As you look at Listings 5 to 10, remember that latexindent.pl is just following its rules, and there is nothing particular about these code snippets. All of the rules can be modified so that you can personalize your indentation scheme.

In each of the samples given in Listings 5 to 10 the 'before' case is a 'worst case scenario' with no effort to make indentation. The 'after' result would be the same, regardless of the leading white space at the beginning of each line which is stripped by latexindent.pl (unless a verbatim-like environment or noIndentBlock is specified – more on this in Section 5).
3. HOW TO USE THE SCRIPT

latexindent.pl ships as part of the TEXLive distribution for Linux and Mac users; latexindent.exe ships as part of the TEXLive and MiKTeX distributions for Windows users. These files are also available from github if you wish to use them without a TEX distribution; in this case, you may like to read appendix B on page 92 which details how the path variable can be updated.

In what follows, we will always refer to latexindent.pl, but depending on your operating system and preference, you might substitute latexindent.exe or simply latexindent.pl.

There are two ways to use latexindent.pl: from the command line, and using arara; we discuss these in Section 3.1 and Section 3.2 respectively. We will discuss how to change the settings and behaviour of the script in Section 5 on page 17.

latexindent.pl ships with latexindent.exe for Windows users, so that you can use the script with or without a Perl distribution. If you plan to use latexindent.pl (i.e., the original Perl script) then you will need a few standard Perl modules – see appendix A on page 91 for details; in particular, note that a module installer helper script is shipped with latexindent.pl.

3.1 From the command line

latexindent.pl has a number of different switches/flags/options, which can be combined in any way that you like, either in short or long form as detailed below. latexindent.pl produces a .log file, indent.log, every time it is run; the name of the log file can be customized, but we will refer to the log file as indent.log throughout this document. There is a base of information that is written to indent.log, but other additional information will be written depending on which of the following options are used.

N: 2018-01-13

-v, --version
3. HOW TO USE THE SCRIPT

$ latexindent.pl -v

This will output only the version number to the terminal.

-h, --help

$ latexindent.pl -h

As above this will output a welcome message to the terminal, including the version number and available options.

$ latexindent.pl myfile.tex

This will operate on myfile.tex, but will simply output to your terminal; myfile.tex will not be changed by latexindent.pl in any way using this command.

-w, --overwrite

$ latexindent.pl -w myfile.tex
$ latexindent.pl --overwrite myfile.tex
$ latexindent.pl myfile.tex --overwrite

This will overwrite myfile.tex, but it will make a copy of myfile.tex first. You can control the name of the extension (default is .bak), and how many different backups are made – more on this in Section 5, and in particular see backupExtension and onlyOneBackUp.

Note that if latexindent.pl can not create the backup, then it will exit without touching your original file; an error message will be given asking you to check the permissions of the backup file.

-o=output.tex, --outputfile=output.tex

$ latexindent.pl -o=output.tex myfile.tex
$ latexindent.pl myfile.tex -o=output.tex
$ latexindent.pl --outputfile=output.tex myfile.tex
$ latexindent.pl --outputfile output.tex myfile.tex

This will indent myfile.tex and output it to output.tex, overwriting it (output.tex) if it already exists. Note that if latexindent.pl is called with both the -w and -o switches, then -w will be ignored and -o will take priority (this seems safer than the other way round).

Note that using -o as above is equivalent to using

$ latexindent.pl myfile.tex > output.tex

You can call the -o switch with the name of the output file without an extension; in this case, latexindent.pl will use the extension from the original file. For example, the following two calls to latexindent.pl are equivalent:

$ latexindent.pl myfile.tex -o=output
$ latexindent.pl myfile.tex -o=output.tex

1Users of version 2.* should note the subtle change in syntax
3. HOW TO USE THE SCRIPT

You can call the \texttt{-o} switch using a \texttt{+} symbol at the beginning; this will concatenate the name of the input file and the text given to the \texttt{-o} switch. For example, the following two calls to \texttt{latexindent.pl} are equivalent:

```
$ latexindent.pl myfile.tex -o=+new
$ latexindent.pl myfile.tex -o=myfilenew.tex
```

You can call the \texttt{-o} switch using a \texttt{++} symbol at the end of the name of your output file; this tells \texttt{latexindent.pl} to search successively for the name of your output file concatenated with 0, 1, \ldots while the name of the output file exists. For example,

```
$ latexindent.pl myfile.tex -o=output++
```

tells \texttt{latexindent.pl} to output to \texttt{output0.tex}, but if it exists then output to \texttt{output1.tex}, and so on.

Calling \texttt{latexindent.pl} with simply

```
$ latexindent.pl myfile.tex -o=++
```

tells it to output to \texttt{myfile0.tex}, but if it exists then output to \texttt{myfile1.tex} and so on.

The \texttt{+} and \texttt{++} feature of the \texttt{-o} switch can be combined; for example, calling

```
$ latexindent.pl myfile.tex -o=+out++
```

tells \texttt{latexindent.pl} to output to \texttt{myfileout0.tex}, but if it exists, then try \texttt{myfileout1.tex}, and so on.

There is no need to specify a file extension when using the \texttt{++} feature, but if you wish to, then you should include it after the \texttt{++} symbols, for example

```
$ latexindent.pl myfile.tex -o=+out++.tex
```

See appendix D on page 94 for details of how the interface has changed from Version 2.2 to Version 3.0 for this flag.

\texttt{-s, -silent}

silent mode: no output will be given to the terminal.

\texttt{-t, -trace}

tracing mode: verbose output will be given to \texttt{indent.log}. This is useful if \texttt{latexindent.pl} has made a mistake and you're trying to find out where and why. You might also be interested in learning about \texttt{latexindent.pl}'s thought process – if so, this switch is for you, although it should be noted that, especially for large files, this does affect performance of the script.

\texttt{-tt, -ttrace}
More detailed tracing mode: this option gives more details to indent.log than the standard trace option (note that, even more so than with -t, especially for large files, performance of the script will be affected).

-1, -local[mymyaml.yaml,other.yaml,...]

latexindent.pl will always load defaultSettings.yaml (rhymes with camel) and if it is called with the -l switch and it finds localSettings.yaml in the same directory as myfile.tex then these settings will be added to the indentation scheme. Information will be given in indent.log on the success or failure of loading localSettings.yaml.

The -l flag can take an optional parameter which details the name (or names separated by commas) of a YAML file(s) that resides in the same directory as myfile.tex; you can use this option if you would like to load a settings file in the current working directory that is not called localSettings.yaml. In fact, you can specify both relative and absolute paths for your YAML files; for example

You will find a lot of other explicit demonstrations of how to use the -l switch throughout this documentation,

You can call the -l switch with a ‘+’ symbol either before or after another YAML file; for example:

which translate, respectively, to

Note that the following is not allowed:

and
will only load `localSettings.yaml`, and `myyaml.yaml` will be ignored. If you wish to use spaces between any of the YAML settings, then you must wrap the entire list of YAML files in quotes, as demonstrated above.

You may also choose to omit the `yaml` extension, such as

```
cmh:~$ latexindent.pl -l=localSettings,myyaml myfile.tex
```

-y, --yaml=yaml settings

```
cmh:~$ latexindent.pl myfile.tex -y="defaultIndent:'\t'"
cmh:~$ latexindent.pl myfile.tex -y="defaultIndent:'\t',maximumIndentation:'\t'"
cmh:~$ latexindent.pl myfile.tex -y="indentRules:one:'\t\t\t\t'

-cmh:~$ latexindent.pl myfile.tex -y='modifyLineBreaks:environments:EndStartsOnOwnLine:3' -m
cmh:~$ latexindent.pl myfile.tex -y='modifyLineBreaks:environments:one:EndStartsOnOwnLine:3' -m
```

You can specify YAML settings from the command line using the -y or --yaml switch; sample demonstrations are given above. Note, in particular, that multiple settings can be specified by separating them via commas. There is a further option to use a ; to separate fields, which is demonstrated in Section 4.3 on page 16.

Any settings specified via this switch will be loaded after any specified using the -l switch. This is discussed further in Section 4.4 on page 16.

-d, --onlydefault

```
cmh:~$ latexindent.pl -d myfile.tex
```

Only defaultSettings.yaml: you might like to read Section 5 before using this switch. By default, latexindent.pl will always search for `indentconfig.yaml` or `.indentconfig.yaml` in your home directory. If you would prefer it not to do so then (instead of deleting or renaming `indentconfig.yaml` or `.indentconfig.yaml`) you can simply call the script with the -d switch; note that this will also tell the script to ignore `localSettings.yaml` even if it has been called with the -l switch; latexindent.pl will also ignore any settings specified from the -y switch.

-c, --cruft=<directory>

```
cmh:~$ latexindent.pl -c=/path/to/directory/ myfile.tex
```

If you wish to have backup files and `indent.log` written to a directory other than the current working directory, then you can send these ‘cruft’ files to another directory.

-g, --logfile=<name of log file>

```
cmh:~$ latexindent.pl -g=other.log myfile.tex
cmh:~$ latexindent.pl -g other.log myfile.tex
cmh:~$ latexindent.pl --logfile other.log myfile.tex
cmh:~$ latexindent.pl myfile.txt -g other.log
```

By default, latexindent.pl reports information to `indent.log`, but if you wish to change the name of this file, simply call the script with your chosen name after the -g switch as demonstrated above.

-sl, --screenlog
Using this option tells \texttt{latexindent.pl} to output the log file to the screen, as well as to your chosen log file.

\texttt{\textasciitilde\$ latexindent.pl -sl myfile.tex}

\texttt{\textasciitilde\$ latexindent.pl -screenlog myfile.tex}

\texttt{\textasciitilde\$ latexindent.pl -m myfile.tex}

\texttt{\textasciitilde\$ latexindent.pl -modifylinebreaks myfile.tex}

One of the most exciting developments in Version 3.0 is the ability to modify line breaks; for full details see Section 6 on page 53

\texttt{latexindent.pl} can also be called on a file without the file extension, for example

\texttt{\textasciitilde\$ latexindent.pl myfile}

and in which case, you can specify the order in which extensions are searched for; see Listing 14 on page 17 for full details.

\texttt{\textasciitilde\$ cat myfile.tex | latexindent.pl}

\texttt{latexindent.pl} will allow input from STDIN, which means that you can pipe output from other commands directly into the script. For example assuming that you have content in \texttt{myfile.tex}, then the above command will output the results of operating upon \texttt{myfile.tex}

Similarly, if you simply type \texttt{latexindent.pl} at the command line, then it will expect (STDIN) input from the command line.

Once you have finished typing your input, you can press

- CTRL+D on Linux
- CTRL+Z followed by ENTER on Windows

to signify that your input has finished. Thanks to [3] for an update to this feature.

\section*{3.2 From \texttt{arara}}

Using \texttt{latexindent.pl} from the command line is fine for some folks, but others may find it easier to use from \texttt{arara}; you can find the \texttt{arara} rule for \texttt{latexindent.pl} and its associated documentation at [2].

\section*{4 indentconfig.yaml, local settings and the -y switch}

The behaviour of \texttt{latexindent.pl} is controlled from the settings specified in any of the YAML files that you tell it to load. By default, \texttt{latexindent.pl} will only load \texttt{defaultSettings.yaml}, but there are a few ways that you can tell it to load your own settings files.

\subsection*{4.1 indentconfig.yaml and \texttt{.indentconfig.yaml}}

\texttt{latexindent.pl} will always check your home directory for \texttt{indentconfig.yaml} and \texttt{.indentconfig.yaml} (unless it is called with the \texttt{-d} switch), which is a plain text file you can create that contains the \textit{absolute} paths for any settings files that you wish \texttt{latexindent.pl} to load. There is no difference between \texttt{indentconfig.yaml} and \texttt{.indentconfig.yaml}, other than the fact that \texttt{.indentconfig.yaml} is a 'hidden' file; thank you to [6] for providing this feature. In what follows, we will use \texttt{indentconfig.yaml},
4. INDENTCONFIG.YAML, LOCAL SETTINGS AND THE -Y SWITCH

but it is understood that this could equally represent .indentconfig.yaml. If you have both files in existence then indentconfig.yaml takes priority.

For Mac and Linux users, their home directory is /username while Windows (Vista onwards) is C:\Users\username² Listing 11 shows a sample indentconfig.yaml file.

<table>
<thead>
<tr>
<th>LISTING 11: indentconfig.yaml (sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td># Paths to user settings for latexindent.pl</td>
</tr>
<tr>
<td># Note that the settings will be read in the order you</td>
</tr>
<tr>
<td># specify here- each successive settings file will overwrite</td>
</tr>
<tr>
<td># the variables that you specify</td>
</tr>
<tr>
<td>paths:</td>
</tr>
<tr>
<td>- /home/cmhughes/Documents/yamlfiles/mysettings.yaml</td>
</tr>
<tr>
<td>- /home/cmhughes/folder/othersettings.yaml</td>
</tr>
<tr>
<td>- /some/other/folder/anynameyouwant.yaml</td>
</tr>
<tr>
<td>- C:\Users\chughes\Documents\mysettings.yaml</td>
</tr>
<tr>
<td>- C:\Users\chughes\Desktop\test spaces\more spaces.yaml</td>
</tr>
</tbody>
</table>

Note that the .yaml files you specify in indentconfig.yaml will be loaded in the order in which you write them. Each file doesn't have to have every switch from defaultSettings.yaml; in fact, I recommend that you only keep the switches that you want to change in these settings files.

To get started with your own settings file, you might like to save a copy of defaultSettings.yaml in another directory and call it, for example, mysettings.yaml. Once you have added the path to indentconfig.yaml you can change the switches and add more code-block names to it as you see fit – have a look at Listing 12 for an example that uses four tabs for the default indent, adds the tabbing environment/command to the list of environments that contains alignment delimiters; you might also like to refer to the many YAML files detailed throughout the rest of this documentation.

<table>
<thead>
<tr>
<th>LISTING 12: mysettings.yaml (example)</th>
</tr>
</thead>
<tbody>
<tr>
<td># Default value of indentation</td>
</tr>
<tr>
<td>defaultIndent: &quot;\t\t\t\t&quot;</td>
</tr>
</tbody>
</table>

You can make sure that your settings are loaded by checking indent.log for details – if you have specified a path that latexindent.pl doesn’t recognize then you’ll get a warning, otherwise you’ll get confirmation that latexindent.pl has read your settings file ³.

⚠️ When editing .yaml files it is extremely important to remember how sensitive they are to spaces. I highly recommend copying and pasting from defaultSettings.yaml when you create your first whatevernameyoulike.yaml file. If latexindent.pl can not read your .yaml file it will tell you so in indent.log.

4.2 localSettings.yaml

The -l switch tells latexindent.pl to look for localSettings.yaml in the same directory as myfile.tex. For example, if you use the following command

²If you're not sure where to put indentconfig.yaml, don’t worry latexindent.pl will tell you in the log file exactly where to put it assuming it doesn’t exist already.

³Windows users may find that they have to end .yaml files with a blank line.
then `latexindent.pl` will (assuming it exists) load `localSettings.yaml` from the same directory as `myfile.tex`.

If you’d prefer to name your `localSettings.yaml` file something different, (say, `mysettings.yaml` as in Listing 12) then you can call `latexindent.pl` using, for example,

```
cmh::~$ latexindent.pl -l=mysettings.yaml myfile.tex
```

Any settings file(s) specified using the `-l` switch will be read after `defaultSettings.yaml` and, assuming they exist, any user setting files specified in `indentconfig.yaml`.

Your settings file can contain any switches that you’d like to change; a sample is shown in Listing 13, and you’ll find plenty of further examples throughout this manual.

**Listing 13: localSettings.yaml (example)**

```yaml
# verbatim environments - environments specified
# here will not be changed at all!
verbatimEnvironments:
  cmhenvironment: 0
  myenv: 1
```

You can make sure that your settings file has been loaded by checking `indent.log` for details; if it can not be read then you receive a warning, otherwise you’ll get confirmation that `latexindent.pl` has read your settings file.

### 4.3 The `-y|yaml` switch

You may use the `-y` switch to load your settings; for example, if you wished to specify the settings from Listing 13 using the `-y` switch, then you could use the following command:

```
cmh::~$ latexindent.pl -y="verbatimEnvironments:cmhenvironment:0;myenv:1" myfile.tex
```

Note the use of `;` to specify another field within `verbatimEnvironments`. This is shorthand, and equivalent, to using the following command:

```
cmh::~$ latexindent.pl
   -y="verbatimEnvironments:cmhenvironment:0,verbatimEnvironments:myenv:1"
myfile.tex
```

You may, of course, specify settings using the `-y` switch as well as, for example, settings loaded using the `-l` switch; for example,

```
cmh::~$ latexindent.pl -l=mysettings.yaml
   -y="verbatimEnvironments:cmhenvironment:0;myenv:1" myfile.tex
```

Any settings specified using the `-y` switch will be loaded after any specified using `indentconfig.yaml` and the `-l` switch.

### 4.4 Settings load order

`latexindent.pl` loads the settings files in the following order:

1. `defaultSettings.yaml` is always loaded, and can not be renamed;
2. `anyUserSettings.yaml` and any other arbitrarily-named files specified in `indentconfig.yaml`;
3. `localSettings.yaml` but only if found in the same directory as `myfile.tex` and called with `-l` switch; this file can be renamed, provided that the call to `latexindent.pl` is adjusted accordingly (see Section 4.2). You may specify both relative and absolute paths to other YAML files using the `-l` switch, separating multiple files using commas;

4. any settings specified in the `-y` switch.

A visual representation of this is given in Figure 1.

![Figure 1](image.png)

**Figure 1**: Schematic of the load order described in Section 4.4; solid lines represent mandatory files, dotted lines represent optional files. `indentconfig.yaml` can contain as many files as you like. The files will be loaded in order; if you specify settings for the same field in more than one file, the most recent takes priority.

5 **defaultSettings.yaml**

`latexindent.pl` loads its settings from `defaultSettings.yaml`. The idea is to separate the behaviour of the script from the internal working – this is very similar to the way that we separate content from form when writing our documents in \LaTeX.

If you look in `defaultSettings.yaml` you’ll find the switches that govern the behaviour of `latexindent.pl`. If you’re not sure where `defaultSettings.yaml` resides on your computer, don’t worry as `indent.log` will tell you where to find it. `defaultSettings.yaml` is commented, but here is a description of what each switch is designed to do. The default value is given in each case; whenever you see `integer` in this section, assume that it must be greater than or equal to 0 unless otherwise stated.

```
fileExtensionPreference: \{fields\}
```

`latexindent.pl` can be called to act on a file without specifying the file extension. For example we can call

```
<cmh:~$ latexindent.pl myfile
```

in which case the script will look for `myfile` with the extensions specified in `fileExtensionPreference` in their numeric order. If no match is found, the script will exit. As with all of the fields, you should change and/or add to this as necessary.

Calling `latexindent.pl` `myfile` with the (default) settings specified in Listing 14 means that the script will first look for `myfile.tex`, then `myfile.sty`, `myfile.cls`, and finally `myfile.bib` in order\(^4\).

```
Listing 14: fileExtensionPreference
fileExtensionPreference:
  .tex: 1
  .sty: 2
  .cls: 3
  .bib: 4
```

\(^4\)Throughout this manual, listings shown with line numbers represent code taken directly from `defaultSettings.yaml`.
If you call `latexindent.pl` with the `-w` switch (to overwrite `myfile.tex`) then it will create a backup file before doing any indentation; the default extension is `.bak`, so, for example, `myfile.bak0` would be created when calling `latexindent.pl myfile.tex` for the first time.

By default, every time you subsequently call `latexindent.pl` with the `-w` to act upon `myfile.tex`, it will create successive backup files: `myfile.bak1`, `myfile.bak2`, etc.

If you don’t want a backup for every time that you call `latexindent.pl` (so you don’t want `myfile.bak1`, `myfile.bak2`, etc) and you simply want `myfile.bak` (or whatever you chose `backupExtension` to be) then change `onlyOneBackUp` to 1; the default value of `onlyOneBackUp` is 0.

Some users may only want a finite number of backup files, say at most 3, in which case, they can change this switch. The smallest value of `maxNumberOfBackUps` is 0 which will not prevent backup files being made; in this case, the behaviour will be dictated entirely by `onlyOneBackUp`. The default value of `maxNumberOfBackUps` is 0.

Some users may wish to cycle through backup files, by deleting the oldest backup file and keeping only the most recent; for example, with `maxNumberOfBackUps: 4` and `cycleThroughBackUps` set to 1 then the copy procedure given below would be obeyed.

```
cmh:∼$ copy myfile.bak1 to myfile.bak0
cmh:∼$ copy myfile.bak2 to myfile.bak1
cmh:∼$ copymyfile.bak3 to myfile.bak2
cmh:∼$ copy myfile.bak4 to myfile.bak3
```

The default value of `cycleThroughBackUps` is 0.

`latexindent.pl` writes information to `indent.log`, some of which can be customized by changing `logFilePreferences`; see Listing 15. If you load your own user settings (see Section 4 on page 14) then `latexindent.pl` will detail them in `indent.log`; you can choose not to have the details logged by switching `showEveryYamlRead` to 0. Once all of your settings have been loaded, you can see the amalgamated settings in the log file by switching `showAmalgamatedSettings` to 1, if you wish.
When either of the trace modes (see page 11) are active, you will receive detailed information in indent.log. You can specify character strings to appear before and after the notification of a found code block using, respectively, showDecorationStartCodeBlockTrace and showDecorationFinishCodeBlockTrace. A demonstration is given in appendix C on page 93.

The log file will end with the characters given in endLogFileWith, and will report the GitHub address of latexindent.pl to the log file if showGitHubInfoFooter is set to 1.

latexindent.pl uses the log4perl module [10] to handle the creation of the logfile. You can specify the layout of the information given in the logfile using any of the Log Layouts detailed at [10].

A field that contains a list of environments that you would like left completely alone – no indentation will be performed on environments that you have specified in this field, see Listing 16.

Note that if you put an environment in verbatimEnvironments and in other fields such as lookForAlignDelims or noAdditionalIndent then latexindent.pl will always prioritize verbatimEnvironments.

A field that contains a list of commands that are verbatim commands, for example \lstinline; any commands populated in this field are protected from line breaking routines (only relevant if the -m is active, see Section 6 on page 53).

If you have a block of code that you don’t want latexindent.pl to touch (even if it is not a verbatim-like environment) then you can wrap it in an environment from noIndentBlock; you can use any name you like for this, provided you populate it as demonstrate in Listing 18.

Of course, you don’t want to have to specify these as null environments in your code, so you use them with a comment symbol, %, followed by as many spaces (possibly none) as you like; see Listing 19 for example.
5. DEFAULTSETTINGS.YAML

LISTING 19: noIndentBlock demonstration

\begin{noindent}
this code won’t be touched by latexindent.pl!
\end{noindent}

removeTrailingWhitespace: {fields}

Trailing white space can be removed both before and after processing the document, as detailed in Listing 20; each of the fields can take the values 0 or 1. See Listings 343 to 345 on page 83 for before and after results. Thanks to [18] for providing this feature.

You can specify removeTrailingWhitespace simply as 0 or 1, if you wish; in this case, latexindent.pl will set both beforeProcessing and afterProcessing to the value you specify; see Listing 21.

fileContentsEnvironments: {field}

Before latexindent.pl determines the difference between preamble (if any) and the main document, it first searches for any of the environments specified in fileContentsEnvironments, see Listing 22. The behaviour of latexindent.pl on these environments is determined by their location (preamble or not), and the value indentPreamble, discussed next.

indentPreamble: 0|1

The preamble of a document can sometimes contain some trickier code for latexindent.pl to operate upon. By default, latexindent.pl won’t try to operate on the preamble (as indentPreamble is set to 0, by default), but if you’d like latexindent.pl to try then change indentPreamble to 1.

lookForPreamble: {fields}

Not all files contain preamble; for example, sty, cls and bib files typically do not. Referencing Listing 23, if you set, for example, .tex to 0, then regardless of the setting of the value of indentPreamble, preamble will not be assumed when operating upon .tex files.

preambleCommandsBeforeEnvironments: 0|1

Assuming that latexindent.pl is asked to operate upon the preamble of a document, when this switch is set to 0 then environment code blocks will be sought first, and then command code blocks. When this switch is set to
1, commands will be sought first. The example that first motivated this switch contained the code given in Listing 24.

**Listing 24: Motivating preambleCommandsBeforeEnvironments**

```latex
... preheadhook={\begin{mdframed}[style=myframedstyle]}, postfoothook=\end{mdframed}, ...
```

**defaultIndent: (horizontal space)**

This is the default indentation (\t means a tab, and is the default value) used in the absence of other details for the command or environment we are working with; see indentRules in Section 5.2 on page 31 for more details.

If you’re interested in experimenting with \texttt{latexindent.pl} then you can remove all indentation by setting defaultIndent: "".

**lookForAlignDelims: (fields)**

This contains a list of environments and/or commands that are operated upon in a special way by \texttt{latexindent.pl} (see Listing 25). In fact, the fields in lookForAlignDelims can actually take two different forms: the \texttt{basic} version is shown in Listing 25 and the \texttt{advanced} version in Listing 28; we will discuss each in turn.

The environments specified in this field will be operated on in a special way by \texttt{latexindent.pl}. In particular, it will try and align each column by its alignment tabs. It does have some limitations (discussed further in Section 7), but in many cases it will produce results such as those in Listings 26 and 27.

If you find that \texttt{latexindent.pl} does not perform satisfactorily on such environments then you can set the relevant key to 0, for example \texttt{tabular: 0}; alternatively, if you just want to ignore specific instances of the environment, you could wrap them in something from \texttt{noIndentBlock} (see Listing 18 on page 19).

**Listing 26: tabular1.tex**

```latex
\begin{tabular}{cccc}
1 & 2 & 3 & 4 \\
5 & & 6 & \\
\end{tabular}
```

**Listing 27: tabular1.tex default output**

```latex
\begin{tabular}{cccc}
1 & 2 & 3 & 4 \\
5 & & 6 & \\
\end{tabular}
```

If, for example, you wish to remove the alignment of the \\ within a delimiter-aligned block, then the advanced form of lookForAlignDelims shown in Listing 28 is for you.
Note that you can use a mixture of the basic and advanced form: in Listing 28 tabular and tabularx are advanced and longtable is basic. When using the advanced form, each field should receive at least 1 sub-field, and can (but does not have to) receive any of the following fields:

- `delims`: binary switch (0 or 1) equivalent to simply specifying, for example, `tabular: 1` in the basic version shown in Listing 25. If `delims` is set to 0 then the align at ampersand routine will not be called for this code block (default: 1);
- `alignDoubleBackSlash`: binary switch (0 or 1) to determine if `\` should be aligned (default: 1);
- `spacesBeforeDoubleBackSlash`: optionally specifies the number (integer \( \geq 0 \)) of spaces to be inserted before `\` (default: 1).
- `multiColumnGrouping`: binary switch (0 or 1) that details if `latexindent.pl` should group columns above and below a `\multicolumn` command (default: 0);
- `alignRowsWithoutMaxDelims`: binary switch (0 or 1) that details if rows that do not contain the maximum number of delimeters should be formatted so as to have the ampersands aligned (default: 1);
- `spacesBeforeAmpersand`: optionally specifies the number (integer \( \geq 0 \)) of spaces to be placed before ampersands (default: 1);
- `spacesAfterAmpersand`: optionally specifies the number (integer \( \geq 0 \)) of spaces to be placed after ampersands (default: 1);
- `justification`: optionally specifies the justification of each cell as either `left` or `right` (default: `left`).

We will explore each of these features using the file `tabular2.tex` in Listing 29 (which contains a `\multicolumn` command), and the YAML files in Listings 30 to 36.

```
\begin{tabular}{cccc}
A & B & C & D \\
AAA & BBB & CCC & DDDD \\
\multicolumn{2}{c}{first heading} & \multicolumn{2}{c}{second heading} \\
one & two & three & four \\
five & six & & \\
seven & & \\
\end{tabular}
```

```
lookForAlignDelims:
    tabular:
        delims: 1
        alignDoubleBackSlash: 0
        spacesBeforeDoubleBackSlash: 0
        multiColumnGrouping: 0
        alignRowsWithoutMaxDelims: 0
        spacesBeforeAmpersand: 1
        spacesAfterAmpersand: 1
    tabularx:
        delims: 1
    longtable: 1
```

```
lookForAlignDelims:
    tabular:
        multiColumnGrouping: 1
```

5Previously this only activated if `alignDoubleBackSlash` was set to 0.
On running the commands

```
cmh:~$ latexindent.pl tabular2.tex

```

we obtain the respective outputs given in Listings 37 to 44.

### Listing 37: `tabular2.tex` default output

```
\begin{tabular}{cccc}
  A \& B \& C \& D \\
  AAA \& BBB \& CCC \& DDD \\
  \hline
  \multicolumn{2}{c}{first heading} \& \multicolumn{2}{c}{second heading} \\
  one \& two \& three \& four \\
  five \& \& six \& \ \\
  seven \& \& \& \\
\end{tabular}
```

### Listing 38: `tabular2.tex` using Listing 30

```
\begin{tabular}{cccc}
  A \& B \& C \& D \\
  AAA \& BBB \& CCC \& DDD \\
  \hline
  \multicolumn{2}{c}{first heading} \& \multicolumn{2}{c}{second heading} \\
  one \& two \& three \& four \\
  five \& \& six \& \\
  seven \& \\
\end{tabular}
```
LISTING 39: tabular2.tex using Listing 31

\begin{tabular}{cccc}
A & B & C & D \\
AAA & BBB & CCC & DDD \\
\multicolumn{2}{c}{first heading} & \multicolumn{2}{c}{second heading} \\
one & two & three & four \\
five & six & \\
seven & \\
\end{tabular}

LISTING 40: tabular2.tex using Listings 30 and 32

\begin{tabular}{cccc}
A & B & C & D \\
AAA & BBB & CCC & DDD \\
\multicolumn{2}{c}{first heading} & \multicolumn{2}{c}{second heading} \\
one & two & three & four \\
five & six & \\
seven & \\
\end{tabular}

LISTING 41: tabular2.tex using Listings 30 and 33

\begin{tabular}{cccc}
A & B & C & D \\
AAA & BBB & CCC & DDD \\
\multicolumn{2}{c}{first heading} & \multicolumn{2}{c}{second heading} \\
one & two & three & four \\
five & six & \\
seven & \\
\end{tabular}

LISTING 42: tabular2.tex using Listings 30 and 34

\begin{tabular}{cccc}
A & B & C & D \\
AAA & BBB & CCC & DDD \\
\multicolumn{2}{c}{first heading} & \multicolumn{2}{c}{second heading} \\
one & two & three & four \\
five & six & \\
seven & \\
\end{tabular}

LISTING 43: tabular2.tex using Listings 30 and 35

\begin{tabular}{cccc}
A & B & C & D \\
AAA & BBB & CCC & DDD \\
\multicolumn{2}{c}{first heading} & \multicolumn{2}{c}{second heading} \\
one & two & three & four \\
five & six & \\
seven & \\
\end{tabular}
LISTING 44: tabular2.tex using Listings 30 and 36

\begin{tabular}{cccc}
  A & B & & C & D \\
  AAA & BBB & & CCC & DDD \\
  \multicolumn{2}{c}{first heading} & \multicolumn{2}{c}{second heading} \\
  one & two & & three & four \\
  five & & six & \\
  seven & & & \\
\end{tabular}

Notice in particular:

- in both Listings 37 and 38 all rows have been aligned at the ampersand, even those that do not contain the maximum number of ampersands (3 ampersands, in this case);
- in Listing 37 the columns have been aligned at the ampersand;
- in Listing 38 the \texttt{\textbackslash multicol} command has grouped the 2 columns beneath and above it, because \texttt{multiColumnGrouping} is set to 1 in Listing 30;
- in Listing 39 rows 3 and 6 have not been aligned at the ampersand, because \texttt{alignRowsWithoutMaxDelims} has been set to 0 in Listing 31; however, the `\ \ ` have still been aligned;
- in Listing 40 the columns beneath and above the \texttt{\textbackslash multicol} commands have been grouped (because \texttt{multiColumnGrouping} is set to 1), and there are at least 4 spaces before each aligned ampersand because \texttt{spacesBeforeAmpersand} is set to 4;
- in Listing 41 the columns beneath and above the \texttt{\textbackslash multicol} commands have been grouped (because \texttt{multiColumnGrouping} is set to 1), and there are at least 4 spaces after each aligned ampersand because \texttt{spacesAfterAmpersand} is set to 4;
- in Listing 42 the `\ \` have not been aligned, because \texttt{alignDoubleBackSlash} is set to 0, otherwise the output is the same as Listing 38;
- in Listing 43 the `\ \` have been aligned, and because \texttt{spacesBeforeDoubleBackSlash} is set to 0, there are no spaces ahead of them; the output is otherwise the same as Listing 38.
- in Listing 44 the cells have been right-justified; note that cells above and below the \texttt{\textbackslash multicol} statements have still been group correctly, because of the settings in Listing 30.

As of Version 3.0, the alignment routine works on mandatory and optional arguments within commands, and also within ‘special’ code blocks (see \texttt{specialBeginEnd} on page 26); for example, assuming that you have a command called \texttt{\textbackslash matrix} and that it is populated within \texttt{lookForAlignDelims} (which it is, by default), and that you run the command

```
cmh:~$ latexindent.pl matrix1.tex
```

then the before-and-after results shown in Listings 45 and 46 are achievable by default.

<table>
<thead>
<tr>
<th>Listing 45: matrix1.tex</th>
<th>Listing 46: matrix1.tex default output</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{\textbackslash matrix} [</td>
<td></td>
</tr>
<tr>
<td>1 &amp; 2 &amp; 3</td>
<td></td>
</tr>
<tr>
<td>4 &amp; 5 &amp; 6</td>
<td></td>
</tr>
<tr>
<td>7 &amp; 8 &amp; 9</td>
<td></td>
</tr>
<tr>
<td>10 &amp; 11 &amp; 12</td>
<td></td>
</tr>
<tr>
<td>]</td>
<td></td>
</tr>
<tr>
<td>\texttt{\textbackslash matrix} [</td>
<td></td>
</tr>
<tr>
<td>1 &amp; 2 &amp; 3</td>
<td></td>
</tr>
<tr>
<td>4 &amp; 5 &amp; 6</td>
<td></td>
</tr>
<tr>
<td>7 &amp; 8 &amp; 9</td>
<td></td>
</tr>
<tr>
<td>10 &amp; 11 &amp; 12</td>
<td></td>
</tr>
<tr>
<td>]</td>
<td></td>
</tr>
</tbody>
</table>

If you have blocks of code that you wish to align at the \& character that are not wrapped in, for example, \texttt{\begin{tabular}...\end{tabular}}, then you can use the mark up illustrated in Listing 47; the default output is shown in Listing 48. Note that the `\*` must be next to each other, but that there
can be any number of spaces (possibly none) between the * and \begin{tabular}; note also that you may use any environment name that you have specified in lookForAlignDelims.

<table>
<thead>
<tr>
<th>Listing 47: align-block.tex</th>
</tr>
</thead>
<tbody>
<tr>
<td>%* \begin{tabular}</td>
</tr>
<tr>
<td>1 &amp; 2 &amp; 3 &amp; 4 \</td>
</tr>
<tr>
<td>5 &amp; &amp; 6 &amp; \</td>
</tr>
<tr>
<td>%* \end{tabular}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Listing 48: align-block.tex default output</th>
</tr>
</thead>
<tbody>
<tr>
<td>%* \begin{tabular}</td>
</tr>
<tr>
<td>1 &amp; 2 &amp; 3 &amp; 4 \</td>
</tr>
<tr>
<td>5 &amp; &amp; 6 &amp; \</td>
</tr>
<tr>
<td>%* \end{tabular}</td>
</tr>
</tbody>
</table>

With reference to Table 1 on page 32 and the, yet undiscussed, fields of noAdditionalIndent and indentRules (see Section 5.2 on page 31), these comment-marked blocks are considered environments.

<table>
<thead>
<tr>
<th>indentAfterItems: (fields)</th>
</tr>
</thead>
</table>

The environment names specified in indentAfterItems tell latexindent.pl to look for \item commands; if these switches are set to 1 then indentation will be performed so as indent the code after each item. A demonstration is given in Listings 50 and 51.

<table>
<thead>
<tr>
<th>Listing 50: items1.tex</th>
</tr>
</thead>
<tbody>
<tr>
<td>\begin{itemize}</td>
</tr>
<tr>
<td>\item some text here</td>
</tr>
<tr>
<td>some more text here</td>
</tr>
<tr>
<td>\item another item</td>
</tr>
<tr>
<td>some more text here</td>
</tr>
<tr>
<td>\item another item</td>
</tr>
<tr>
<td>some more text here</td>
</tr>
<tr>
<td>\end{itemize}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Listing 51: items1.tex default output</th>
</tr>
</thead>
<tbody>
<tr>
<td>\begin{itemize}</td>
</tr>
<tr>
<td>\item some text here</td>
</tr>
<tr>
<td>some more text here</td>
</tr>
<tr>
<td>\item another item</td>
</tr>
<tr>
<td>some more text here</td>
</tr>
<tr>
<td>\item another item</td>
</tr>
<tr>
<td>some more text here</td>
</tr>
<tr>
<td>\end{itemize}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>itemNames: (fields)</th>
</tr>
</thead>
</table>

If you have your own \item commands (perhaps you prefer to use myitem, for example) then you can put populate them in itemNames. For example, users of the exam document class might like to add parts to indentAfterItems and part to itemNames to their user settings (see Section 4 on page 14 for details of how to configure user settings, and Listing 12 on page 16 in particular.)

<table>
<thead>
<tr>
<th>specialBeginEnd: (fields)</th>
</tr>
</thead>
</table>

The fields specified in specialBeginEnd are, in their default state, focused on math mode begin and end statements, but there is no requirement for this to be the case; Listing 53 shows the default settings of specialBeginEnd.
The field `displayMath` represents \[...\], `inlineMath` represents $...$ and `displayMathTex` represents $$...$$. You can, of course, rename these in your own YAML files (see Section 4.2 on page 15); indeed, you might like to set up your own special begin and end statements.

A demonstration of the before-and-after results are shown in Listings 54 and 55.

The function $f$ has formula $f(x)=x^2$. If you like splitting dollars, $g(x)=f(2x)$.

For each field, `lookForThis` is set to 1 by default, which means that `latexindent.pl` will look for this pattern; you can tell `latexindent.pl` not to look for the pattern, by setting `lookForThis` to 0.

There are examples in which it is advantageous to search for `specialBeginEnd` fields before searching for commands, and the `specialBeforeCommand` switch controls this behaviour. For example, consider the file shown in Listing 56.

Upon running the following commands
we receive the respective outputs in Listings 59 and 60.

<table>
<thead>
<tr>
<th>LISTING 59: specialLR.tex using Listing 57</th>
<th>LISTING 60: specialLR.tex using Listings 57 and 58</th>
</tr>
</thead>
<tbody>
<tr>
<td>\begin{equation} \left[ \sqrt{a+b} \right] \end{equation}</td>
<td>\begin{equation} \left[ \sqrt{a+b} \right] \end{equation}</td>
</tr>
</tbody>
</table>

Notice that in:

- Listing 59 the \left has been treated as a command, with one optional argument;
- Listing 60 the specialBeginEnd pattern in Listing 57 has been obeyed because Listing 58 specifies that the specialBeginEnd should be sought before commands.

You can, optionally, specify the middle field for anything that you specify in specialBeginEnd. For example, let’s consider the .tex file in Listing 61.

<table>
<thead>
<tr>
<th>LISTING 61: special2.tex</th>
</tr>
</thead>
<tbody>
<tr>
<td>\If something 0 \ElseIf something 1 \ElseIf something 2 \ElseIf something 3 \Else something 4 \EndIf</td>
</tr>
</tbody>
</table>

Upon saving the YAML settings in Listings 62 and 64 and running the commands

```
cmh:~$ latexindent.pl special2.tex -l=middle
```
```
cmh:~$ latexindent.pl special2.tex -l=middle1
```

then we obtain the output given in Listings 63 and 65.
We note that:

- in Listing 63 the bodies of each of the Elsif statements have been indented appropriately;
- the Else statement has not been indented appropriately in Listing 63 – read on!
- we have specified multiple settings for the middle field using the syntax demonstrated in Listing 64 so that the body of the Else statement has been indented appropriately in Listing 65.

You may specify fields in specialBeginEnd to be treated as verbatim code blocks by changing lookForThis to be verbatim.

For example, beginning with the code in Listing 67 and the YAML in Listing 66, and running

```
cmh:~$ latexindent.pl special3.tex -l=special-verb1
```

then the output in Listing 67 is unchanged.

This field enables the user to specify indentation rules that take effect after heading commands such as \part, \chapter, \section, \subsection, or indeed any user-specified command written in this field.\(^6\)

The default settings do not place indentation after a heading, but you can easily switch them on by changing

```
indentAfterThisHeading: 0
```

The level field tells latexindent.pl the hierarchy of the heading structure in your document.

\(^6\)There is a slight difference in interface for this field when comparing Version 2.2 to Version 3.0; see appendix D on page 94 for details.
You might, for example, like to have both section and subsection set with level: 3 because you do not want the indentation to go too deep.

You can add any of your own custom heading commands to this field, specifying the level as appropriate. You can also specify your own indentation in indentRules (see Section 5.2 on the next page); you will find the default indentRules contains chapter: " " which tells latexindent.pl simply to use a space character after headings (once indent is set to 1 for chapter).

For example, assuming that you have the code in Listing 69 saved into headings1.yaml, and that you have the text from Listing 70 saved into headings1.tex.

**Listing 69: headings1.yaml**

```yaml
indentAfterHeadings:
  subsection:
    indentAfterThisHeading: 1
    level: 1
  paragraph:
    indentAfterThisHeading: 1
    level: 2
```

**Listing 70: headings1.tex**

```latex
\subsection{subsection title}
subsection text
subsection text
\paragraph{paragraph title}
paragraph text
paragraph text
\paragraph{paragraph title}
paragraph text
paragraph text
```

If you run the command

```
cmh:~$ latexindent.pl headings1.tex -l=headings1.yaml
```

then you should receive the output given in Listing 71.

**Listing 71: headings1.tex using Listing 69**

```latex
\subsection{subsection title}
  \subsection text
  \subsection text
  \paragraph{paragraph title}
  \indent  paragraph text
  \indent  paragraph text
  \indent  paragraph text
  \indent  paragraph text
  \indent  paragraph text
```

Now say that you modify the YAML from Listing 69 so that the paragraph level is 1; after running

```
cmh:~$ latexindent.pl headings1.tex -l=headings1.yaml
```

you should receive the code given in Listing 72; notice that the paragraph and subsection are at the same indentation level.

**Listing 72: headings1.tex second modification**

```latex
\subsection{subsection title}
  \subsection text
  \subsection text
  \paragraph{paragraph title}
  \indent  paragraph text
  \indent  paragraph text
  \indent  paragraph text
  \indent  paragraph text
  \indent  paragraph text
```

You can control the maximum indentation given to your file by specifying the maximumIndentation field as horizontal space (but not including tabs). This feature uses the Text::Tabs module [15], and is off by default.

For example, consider the example shown in Listing 73 together with the default output shown in Listing 74.
Now say that, for example, you have the `max-indentation1.yaml` from Listing 75 and that you run the following command:

```bash
cmh:~$ latexindent.pl mult-nested.tex -l=max-indentation1
```

You should receive the output shown in Listing 76.

Comparing the output in Listings 74 and 76 we notice that the (default) tabs of indentation have been replaced by a single space.

In general, when using the `maximumIndentation` feature, any leading tabs will be replaced by equivalent spaces except, of course, those found in `verbatimEnvironments` (see Listing 16 on page 19) or `noIndentBlock` (see Listing 18 on page 19).

### 5.1 The code blocks known latexindent.pl

As of Version 3.0, `latexindent.pl` processes documents using code blocks; each of these are shown in Table 1.

We will refer to these code blocks in what follows.

### 5.2 noAdditionalIndent and indentRules

`latexindent.pl` operates on files by looking for code blocks, as detailed in Section 5.1; for each type of code block in Table 1 on the next page (which we will call a `<thing>` in what follows) it searches YAML fields for information in the following order:

1. `noAdditionalIndent` for the name of the current `<thing>`;
2. `indentRules` for the name of the current `<thing>`;
3. `noAdditionalIndentGlobal` for the type of the current `<thing>`;
4. `indentRulesGlobal` for the type of the current `<thing>`.
### Table 1: Code blocks known to `latexindent.pl`

<table>
<thead>
<tr>
<th>Code block</th>
<th>characters allowed in name</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>environments</td>
<td>a-zA-Z*0-9_</td>
<td>\begin{myenv} body of myenv \end{myenv}</td>
</tr>
<tr>
<td>optionalArguments</td>
<td>inherits name from parent (e.g. environment name)</td>
<td>[ opt arg text ]</td>
</tr>
<tr>
<td>mandatoryArguments</td>
<td>inherits name from parent (e.g. environment name)</td>
<td>{ mand arg text }</td>
</tr>
<tr>
<td>commands</td>
<td>+a-zA-Z*0-9_</td>
<td>\mycommand{arguments}</td>
</tr>
<tr>
<td>keyEqualsValuesBracesBrackets</td>
<td>a-zA-Z*0-9_/.h{}#:</td>
<td>my key/.style={arguments}</td>
</tr>
<tr>
<td>namedGroupingBracesBrackets</td>
<td>0-9.a-zA-Z*&lt;&gt;</td>
<td>in{arguments}</td>
</tr>
<tr>
<td>UnNamedGroupingBracesBrackets</td>
<td>No name!</td>
<td>{ or [ or , or &amp; or ) or ( or $ followed by {arguments}</td>
</tr>
</tbody>
</table>
| ifElseFi                            | @a-zA-Z but must begin with either \if of \@if | \ifnum...
|                                    |                            | ... \else
|                                    |                            | ... \fi                                                                |
| items                               | User specified, see Listings 49 and 52 on page 26 | \begin{enumerate} \item ...
|                                    |                            | \end{enumerate}                                                       |
| specialBeginEnd                     | User specified, see Listing 53 on page 27 | \[
|                                    |                            | ... \]
| afterHeading                        | User specified, see Listing 68 on page 29 | \chapter{title}
|                                    |                            | ... \section{title}                                                    |
| filecontents                        | User specified, see Listing 22 on page 20 | \begin{filecontents}
|                                    |                            | ...
|                                    |                            | \end{filecontents}                                                     |
Using the above list, the first piece of information to be found will be used; failing that, the value of defaultIndent is used. If information is found in multiple fields, the first one according to the list above will be used; for example, if information is present in both indentRules and in noAdditionalIndentGlobal, then the information from indentRules takes priority.

We now present details for the different type of code blocks known to latexindent.pl, as detailed in Table 1 on the preceding page; for reference, there follows a list of the code blocks covered.

5.2.1 Environments and their arguments ................................................. 33
5.2.2 Environments with items ............................................................... 39
5.2.3 Commands with arguments ............................................................. 40
5.2.4 ifelsefi code blocks ................................................................. 42
5.2.5 specialBeginEnd code blocks ......................................................... 44
5.2.6 afterHeading code blocks ............................................................ 45
5.2.7 The remaining code blocks ............................................................ 47
   keyEqualsValuesBracesBrackets ....................................................... 47
   namedGroupingBracesBrackets ......................................................... 47
   UnNamedGroupingBracesBrackets ...................................................... 48
   filecontents ................................................................. 48
5.2.8 Summary ................................................................. 48

5.2.1 Environments and their arguments

There are a few different YAML switches governing the indentation of environments; let's start with the code shown in Listing 77.

```
\begin{outer}
\begin{myenv}
 body of environment
 body of environment
 body of environment
\end{myenv}
\end{outer}
```

If we do not wish myenv to receive any additional indentation, we have a few choices available to us, as demonstrated in Listings 78 and 79.

```
noAdditionalIndent: (fields)
```

On applying either of the following commands,

```
cmh:~$ latexindent.pl myenv.tex -l myenv-noAdd1.yaml
cmh:~$ latexindent.pl myenv.tex -l myenv-noAdd2.yaml
```

we obtain the output given in Listing 80; note in particular that the environment myenv has not received any additional indentation, but that the outer environment has still received indentation.
Upon changing the YAML files to those shown in Listings 81 and 82, and running either

```
cmh:~$ latexindent.pl myenv.tex -l myenv-noAdd3.yaml
```
```
cmh:~$ latexindent.pl myenv.tex -l myenv-noAdd4.yaml
```

we obtain the output given in Listing 83.

---

**Listing 81:**
```
myenv-noAdd3.yaml
```

```
noAdditionalIndent:
  myenv: 0
```

---

**Listing 82:**
```
myenv-noAdd4.yaml
```

```
noAdditionalIndent:
  myenv:
    body: 0
```

---

**Listing 83:**
```
myenv.tex output (using either Listing 81 or Listing 82)
```

```
\begin{outer}
  \begin{myenv}
    body of environment
    body of environment
    body of environment
  \end{myenv}
\end{outer}
```

Let's now allow `myenv` to have some optional and mandatory arguments, as in Listing 84.

---

**Listing 84:**
```
myenv-args.tex
```

```
\begin{outer}
  \begin{myenv}[]%
    optional argument text
    mandatory argument text
  \end{myenv}
```

```
\begin{myenv}[]%
    optional argument text
    mandatory argument text
  \end{myenv}
```

```
body of environment
body of environment
body of environment
\end{myenv}
\end{outer}
```

---

Upon running

```

cmh:~$ latexindent.pl -l=myenv-noAdd1.yaml myenv-args.tex
```

we obtain the output shown in Listing 85; note that the optional argument, mandatory argument
and body all have received no additional indent. This is because, when `noAdditionalIndent` is
specified in `scalar` form (as in Listing 78), then all parts of the environment (body, optional and
mandatory arguments) are assumed to want no additional indent.
We may customise noAdditionalIndent for optional and mandatory arguments of the \texttt{myenv} environment, as shown in, for example, Listings 86 and 87.

Upon running

```
\texttt{cmh:}\$ \texttt{latexindent.pl myenv.tex -l myenv-noAdd5.yaml}
\texttt{cmh:}\$ \texttt{latexindent.pl myenv.tex -l myenv-noAdd6.yaml}
```

we obtain the respective outputs given in Listings 88 and 89. Note that in Listing 88 the text for the optional argument has not received any additional indentation, and that in Listing 89 the mandatory argument has not received any additional indentation; in both cases, the body has not received any additional indentation.

We may also specify indentation rules for environment code blocks using the \texttt{indentRules} field; see, for example, Listings 90 and 91.
On applying either of the following commands,

```bash
$ latexindent.pl myenv.tex -l myenv-rules1.yaml
$ latexindent.pl myenv.tex -l myenv-rules2.yaml
```

we obtain the output given in Listing 92; note in particular that the environment `myenv` has received one tab (from the outer environment) plus three spaces from Listing 90 or 91.

```
\begin{outer}
  \begin{myenv}
    \begin{body}
      \begin{environment}
        \begin{environment}
          \begin{environment}
            \begin{environment}
              \end{myenv}
            \end{environment}
          \end{environment}
        \end{environment}
      \end{body}
    \end{environment}
  \end{myenv}
\end{outer}
```

If you specify a field in `indentRules` using anything other than horizontal space, it will be ignored.

Returning to the example in Listing 84 that contains optional and mandatory arguments. Upon using Listing 90 as in

```bash
$ latexindent.pl myenv-args.tex -l=myenv-rules1.yaml
```

we obtain the output in Listing 93; note that the body, optional argument and mandatory argument of `myenv` have all received the same customised indentation.

```
\begin{outer}
  \begin{myenv}[
    \optional{argument}{text}
    \optional{argument}{text}
    \mandatory{argument}{text}
  ]
    \begin{body}
      \begin{environment}
        \begin{environment}
          \begin{environment}
            \begin{environment}
              \end{myenv}
            \end{environment}
          \end{environment}
        \end{environment}
      \end{body}
    \end{environment}
  \end{myenv}
\end{outer}
```

You can specify different indentation rules for the different features using, for example, Listings 94 and 95

```
LISTING 94: myenv-rules3.yaml
indentRules:
  myenv:
    body: "  "
    optionalArguments: "  "
```

```
LISTING 95: myenv-rules4.yaml
indentRules:
  myenv:
    body: "  "
    mandatoryArguments: "\t\t"
```

After running
then we obtain the respective outputs given in Listings 96 and 97.

**LISTING 96:** myenv-args.tex using Listing 94

```
\begin{outer}
  \begin{myenv}[
    \optional, optional, argument, text
    \optional, optional, argument, text]
    \mandatory, mandatory, argument, text
    \body, of, environment
    \body, of, environment
    \body, of, environment
  \end{myenv}
\end{outer}
```

**LISTING 97:** myenv-args.tex using Listing 95

```
\begin{outer}
  \begin{myenv}[
    \body, of, environment
    \body, of, environment
    \body, of, environment
  \end{myenv}
\end{outer}
```

Note that in Listing 96, the optional argument has only received a single space of indentation, while the mandatory argument has received the default (tab) indentation; the environment body has received three spaces of indentation.

In Listing 97, the optional argument has received the default (tab) indentation, the mandatory argument has received two tabs of indentation, and the body has received three spaces of indentation.

Assuming that your environment name is not found within neither noAdditionalIndent nor indentRules, the next place that latexindent.pl will look is noAdditionalIndentGlobal, and in particular for the environments key (see Listing 98). Let’s say that you change the value of environments to 1 in Listing 98, and that you run

```
cmh:~$ latexindent.pl myenv-args.tex -l env-noAdditionalGlobal.yaml
```

The respective output from these two commands are in Listings 99 and 100; in Listing 99 notice that both environments receive no additional indentation but that the arguments of myenv still do receive indentation. In Listing 100 notice that the outer environment does not receive additional indentation, but because of the settings from myenv-rules1.yaml (in Listing 90 on the previous page), the myenv environment still does receive indentation.
Listing 99: myenv-args.tex using Listing 98
\begin{outer}
\begin{myenv}[%
  optional argument text
  optional argument text]
  { mandatory argument text
    mandatory argument text}
body of environment
body of environment
body of environment
\end{myenv}
\end{outer}

Listing 100: myenv-args.tex using Listings 90 and 98
\begin{outer}
\begin{myenv}[%
  optional argument text
  optional argument text]
  { mandatory argument text
    mandatory argument text}
body of environment
body of environment
body of environment
\end{myenv}
\end{outer}

In fact, noAdditionalIndentGlobal also contains keys that control the indentation of optional and mandatory arguments; on referencing Listings 101 and 102

Listing 101:
opt-args-no-add-glob.yaml
noAdditionalIndentGlobal:
  optionalArguments: 1

Listing 102:
mand-args-no-add-glob.yaml
noAdditionalIndentGlobal:
  mandatoryArguments: 1

we may run the commands

cmh:~/$ latexindent.pl myenv-args.tex -local opt-args-no-add-glob.yaml
cmh:~/$ latexindent.pl myenv-args.tex -local mand-args-no-add-glob.yaml

which produces the respective outputs given in Listings 103 and 104. Notice that in Listing 103 the optional argument has not received any additional indentation, and in Listing 104 the mandatory argument has not received any additional indentation.

Listing 103: myenv-args.tex using Listing 101
\begin{outer}
\begin{myenv}[%
  optional argument text
  optional argument text]
  { mandatory argument text
    mandatory argument text}
body of environment
body of environment
body of environment
\end{myenv}
\end{outer}

indentRulesGlobal: {fields}

The final check that latexindent.pl will make is to look for indentRulesGlobal as detailed in Listing 105; if you change the environments field to anything involving horizontal space, say " ", and then run the following commands

cmh:~/$ latexindent.pl myenv-args.tex -l env-indentRules.yaml
cmh:~/$ latexindent.pl myenv-args.tex -l myenv-rules1.yaml,env-indentRules.yaml

Listing 105:
indentRulesGlobal:
  environments: 0
then the respective output is shown in Listings 106 and 107. Note that in Listing 106, both the environment blocks have received a single-space indentation, whereas in Listing 107 the outer environment has received single-space indentation (specified by indentRulesGlobal), but myenv has received "\", as specified by the particular indentRules for myenv Listing 90 on page 36.

LISTING 106: myenv-args.tex using Listing 105
\begin{outer}
  \begin{myenv}[
    \begin{optional_argument_text}
    \begin{optional_argument_text}
    \begin{mandatory_argument_text}
    \begin{mandatory_argument_text}
    \body_of_environment
    \body_of_environment
    \body_of_environment
    \end{myenv}
  \end{myenv}
\end{outer}

LISTING 107: myenv-args.tex using Listings 90 and 105
\begin{outer}
  \begin{myenv}[
    \begin{optional_argument_text}
    \begin{optional_argument_text}
    \begin{optional_argument_text}
    \begin{mandatory_argument_text}
    \begin{mandatory_argument_text}
    \body_of_environment
    \body_of_environment
    \body_of_environment
    \body_of_environment
    \end{myenv}
  \end{myenv}
\end{outer}

You can specify indentRulesGlobal for both optional and mandatory arguments, as detailed in Listings 108 and 109.

LISTING 108: opt-args-indent-rules-glob.yaml
indentRulesGlobal:
  optionalArguments: "\tt"

LISTING 109: mand-args-indent-rules-glob.yaml
indentRulesGlobal:
  mandatoryArguments: "\tt"

Upon running the following commands

cmh:∼$ latexindent.pl myenv-args.tex -local opt-args-indent-rules-glob.yaml
cmh:∼$ latexindent.pl myenv-args.tex -local mand-args-indent-rules-glob.yaml

we obtain the respective outputs in Listings 110 and 111. Note that the optional argument in Listing 110 has received two tabs worth of indentation, while the mandatory argument has done so in Listing 111.

LISTING 110: myenv-args.tex using Listing 108
\begin{outer}
  \begin{myenv}[
    \begin{optional_argument_text}
    \begin{optional_argument_text}
    \end{myenv}
  \end{myenv}
\end{outer}

LISTING 111: myenv-args.tex using Listing 109
\begin{outer}
  \begin{myenv}[
    \begin{optional_argument_text}
    \begin{optional_argument_text}
    \end{myenv}
  \end{myenv}
\end{outer}

5.2.2 Environments with items

With reference to Listings 49 and 52 on page 26, some commands may contain item commands; for the purposes of this discussion, we will use the code from Listing 50 on page 26.

Assuming that you’ve populated itemNames with the name of your item, you can put the item name into noAdditionalIndent as in Listing 112, although a more efficient approach may be to change the relevant field in itemNames to 0. Similarly, you can customise the indentation that your item receives using indentRules, as in Listing 113.
Upon running the following commands

```
cmh:~$ latexindent.pl items1.tex -local item-noAdd1.yaml

```

the respective outputs are given in Listings 114 and 115; note that in Listing 114 that the text after each item has not received any additional indentation, and in Listing 115, the text after each item has received a single space of indentation, specified by Listing 113.

Alternatively, you might like to populate noAdditionalIndentGlobal or indentRulesGlobal using the items key, as demonstrated in Listings 116 and 117. Note that there is a need to 'reset/remove' the item field from indentRules in both cases (see the hierarchy description given on page 31) as the item command is a member of indentRules by default.

Upon running the following commands,

```
cmh:~$ latexindent.pl items1.tex -local items-noAdditionalGlobal.yaml

```

the respective outputs from Listings 114 and 115 are obtained; note, however, that all such item commands without their own individual noAdditionalIndent or indentRules settings would behave as in these listings.

### 5.2.3 Commands with arguments

Let’s begin with the simple example in Listing 118; when `latexindent.pl` operates on this file, the default output is shown in Listing 119. 

7The command code blocks have quite a few subtleties, described in Section 5.3 on page 49.
As in the environment-based case (see Listings 78 and 79 on page 33) we may specify noAdditionalIndent either in ‘scalar’ form, or in ‘field’ form, as shown in Listings 120 and 121.

After running the following commands,

After running the following commands,

we receive the respective output given in Listings 122 and 123.

Note that in Listing 122 that the ‘body’, optional argument and mandatory argument have all received no additional indentation, while in Listing 123, only the ‘body’ has not received any additional indentation. We define the ‘body’ of a command as any lines following the command name that include its optional or mandatory arguments.

We may further customise noAdditionalIndent for mycommand as we did in Listings 86 and 87 on page 35; explicit examples are given in Listings 124 and 125.

After running the following commands,
we receive the respective output given in Listings 126 and 127.

Listing 126: mycommand.tex using Listing 124

\mycommand
{mand arg text
  mand arg text}
[
opt arg text
  opt arg text
]

Listing 127: mycommand.tex using Listing 125

\mycommand
{mand arg text
  mand arg text}
[
  opt arg text
  opt arg text
]

Attentive readers will note that the body of mycommand in both Listings 126 and 127 has received no additional indent, even though body is explicitly set to 0 in both Listings 124 and 125. This is because, by default, noAdditionalIndentGlobal for commands is set to 1 by default; this can be easily fixed as in Listings 128 and 129.

Listing 128: mycommand-noAdd5.yaml

noAdditionalIndent:
  mycommand:
    body: 0
    optionalArguments: 1
    mandatoryArguments: 0
  commands: 0

Listing 129: mycommand-noAdd6.yaml

noAdditionalIndent:
  mycommand:
    body: 0
    optionalArguments: 0
    mandatoryArguments: 1
  commands: 0

After running the following commands,

cmh:~$ latexindent.pl mycommand.tex -l mycommand-noAdd5.yaml
cmh:~$ latexindent.pl mycommand.tex -l mycommand-noAdd6.yaml

we receive the respective output given in Listings 130 and 131.

Listing 130: mycommand.tex using Listing 128

\mycommand
{mand arg text
  mand arg text}
[
  opt arg text
  opt arg text
]

Listing 131: mycommand.tex using Listing 129

\mycommand
{mand arg text
  mand arg text}
[
  opt arg text
  opt arg text
]

Both indentRules and indentRulesGlobal can be adjusted as they were for environment code blocks, as in Listings 94 and 95 on page 36 and Listings 105, 108 and 109 on pages 38–39.

5.2.4 ifelsefi code blocks

Let’s use the simple example shown in Listing 132; when latexindent.pl operates on this file, the output as in Listing 133; note that the body of each of the \if statements have been indented, and that the \else statement has been accounted for correctly.
It is recommended to specify `noAdditionalIndent` and `indentRules` in the 'scalar' form only for these type of code blocks, although the 'field' form would work, assuming that `body` was specified. Examples are shown in Listings 134 and 135.

After running the following commands,

```bash
cmh:~$ latexindent.pl ifelsefi1.tex -local ifnum-noAdd.yaml
cmh:~$ latexindent.pl ifelsefi1.tex -l ifnum-indent-rules.yaml
```

we receive the respective output given in Listings 136 and 137; note that in Listing 136, the `ifnum` code block has not received any additional indentation, while in Listing 137, the `ifnum` code block has received one tab and two spaces of indentation.

We may specify `noAdditionalIndentGlobal` and `indentRulesGlobal` as in Listings 138 and 139.

Upon running the following commands

```bash
cmh:~$ latexindent.pl ifelsefi1.tex -local ifelsefi-noAdd-glob.yaml
cmh:~$ latexindent.pl ifelsefi1.tex -l ifelsefi-indent-rules-global.yaml
```

we receive the outputs in Listings 140 and 141; notice that in Listing 140 neither of the `ifelsefi` code blocks have received indentation, while in Listing 141 both code blocks have received a single space of indentation.
We can further explore the treatment of `ifElseFi` code blocks in Listing 142, and the associated default output given in Listing 143; note, in particular, that the bodies of each of the ‘or statements’ have been indented.

5.2.5 `specialBeginEnd` code blocks

Let’s use the example from Listing 54 on page 27 which has default output shown in Listing 55 on page 27.

It is recommended to specify `noAdditionalIndent` and `indentRules` in the ‘scalar’ form for these type of code blocks, although the ‘field’ form would work, assuming that body was specified. Examples are shown in Listings 144 and 145.

After running the following commands,

```
cmh:~$ latexindent.pl special1.tex -local displayMath-noAdd.yaml
cmh:~$ latexindent.pl special1.tex -l displayMath-indent-rules.yaml
```

we receive the respective output given in Listings 146 and 147; note that in Listing 146, the displayMath code block has not received any additional indentation, while in Listing 147, the displayMath code block has received three tabs worth of indentation.
We may specify `noAdditionalIndentGlobal` and `indentRulesGlobal` as in Listings 148 and 149.

Upon running the following commands

```
cmh:~$ latexindent.pl special1.tex -local special-noAdd-glob.yaml
cmh:~$ latexindent.pl special1.tex -l special-indent-rules-global.yaml
```

we receive the outputs in Listings 150 and 151; notice that in Listing 150 neither of the special code blocks have received indentation, while in Listing 151 both code blocks have received a single space of indentation.

5.2.6 afterHeading code blocks

Let's use the example Listing 152 for demonstration throughout this Section. As discussed on page 30, by default `latexindent.pl` will not add indentation after headings.

```
\paragraph{paragraph title}
paragraph text
paragraph text
```

On using the YAML file in Listing 154 by running the command

```
cmh:~$ latexindent.pl headings2.tex -l headings3.yaml
```

we obtain the output in Listing 153. Note that the argument of paragraph has received (default) indentation, and that the body after the heading statement has received (default) indentation.

If we specify `noAdditionalIndent` as in Listing 156 and run the command

```
cmh:~$ latexindent.pl headings2.tex -l headings4.yaml
```
then we receive the output in Listing 155. Note that the arguments and the body after the heading of paragraph has received no additional indentation, because we have specified noAdditionalIndent in scalar form.

**Listing 155: headings2.tex using Listing 156**

```
\paragraph{paragraph title}
paragraph text
paragraph text
```

Similarly, if we specify indentRules as in Listing 158 and run analogous commands to those above, we receive the output in Listing 157; note that the body, mandatory argument and content after the heading of paragraph have all received three tabs worth of indentation.

**Listing 157: headings2.tex using Listing 158**

```
\paragraph{paragraph title}
  | \ | \ | \ | \ | \ | \title
  | \ | \ | \ | \ | \ | \paragraph text
  | \ | \ | \ | \ | \ | \paragraph text
```

We may, instead, specify noAdditionalIndent in ‘field’ form, as in Listing 160 which gives the output in Listing 159.

**Listing 159: headings2.tex using Listing 160**

```
\paragraph{paragraph title}
paragraph text
paragraph text
```

Analogously, we may specify indentRules as in Listing 162 which gives the output in Listing 161; note that mandatory argument text has only received a single space of indentation, while the body after the heading has received three tabs worth of indentation.

**Listing 161: headings2.tex using Listing 162**

```
\paragraph{paragraph title}
  | \ | \ | \ | \ | \ | \title
  | \ | \ | \ | \ | \ | \paragraph text
  | \ | \ | \ | \ | \ | \paragraph text
```

Finally, let’s consider noAdditionalIndentGlobal and indentRulesGlobal shown in Listings 164 and 166 respectively, with respective output in Listings 163 and 165. Note that in Listing 164 the mandatory argument of paragraph has received a (default) tab’s worth of indentation, while the body after the heading has received no additional indentation. Similarly, in Listing 165, the argument has received both a (default) tab plus two spaces of indentation (from the global rule specified in Listing 166), and the remaining body after paragraph has received just two spaces of indentation.
5.2.7 The remaining code blocks

Referencing the different types of code blocks in Table 1 on page 32, we have a few code blocks yet to cover; these are very similar to the commands code block type covered comprehensively in Section 5.2.3 on page 40, but a small discussion defining these remaining code blocks is necessary.

keyEqualsValuesBracesBrackets \texttt{latexindent.pl} defines this type of code block by the following criteria:

- it must immediately follow either \{ OR [ OR , with comments and blank lines allowed;
- then it has a name made up of the characters detailed in Table 1 on page 32;
- then an = symbol;
- then at least one set of curly braces or square brackets (comments and line breaks allowed throughout).

An example is shown in Listing 167, with the default output given in Listing 168.

In Listing 168, note that the maximum indentation is three tabs, and these come from:

- the \texttt{\pgfkeys} command's mandatory argument;
- the \texttt{start coordinate/.initial} key's mandatory argument;
- the \texttt{start coordinate/.initial} key's body, which is defined as any lines following the name of the key that include its arguments. This is the part controlled by the \texttt{body} field for noAdditionalIndent and friends from page 31.

namedGroupingBracesBrackets This type of code block is mostly motivated by tikz-based code; we define this code block as follows:

- it must immediately follow either horizontal space OR \texttt{one or more line breaks} OR \{ OR [ OR $ OR ) OR (;
- the name may contain the characters detailed in Table 1 on page 32;
- then at least one set of curly braces or square brackets (comments and line breaks allowed throughout).
A simple example is given in Listing 169, with default output in Listing 170.

<table>
<thead>
<tr>
<th>Listing 169: child1.tex</th>
<th>Listing 170: child1.tex default output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\coordinate</code></td>
<td><code>\coordinate</code></td>
</tr>
<tr>
<td><code>child[grow=down]</code>{</td>
<td><code>child[grow=down]{</code></td>
</tr>
<tr>
<td><code>edge from parent</code> [antiparticle]</td>
<td></td>
</tr>
<tr>
<td><code>node [above=3pt] {$_C$}</code></td>
<td><code>node [above=3pt] {$_C$}</code></td>
</tr>
<tr>
<td>}</td>
<td><code>}</code></td>
</tr>
</tbody>
</table>

In particular, latexindent.pl considers child, parent and node all to be namedGroupingBracesBrackets\(^8\). Referencing Listing 170, note that the maximum indentation is two tabs, and these come from:

- the child's mandatory argument;
- the child's body, which is defined as any lines following the name of the namedGroupingBracesBrackets that include its arguments. This is the part controlled by the body field for noAdditionalIndent and friends from page 31.

**UnNamedGroupingBracesBrackets** occur in a variety of situations; specifically, we define this type of code block as satisfying the following criteria:

- it must immediately follow either \{ OR [ OR , OR & OR ) OR ( OR $;
- then at least one set of curly braces or square brackets (comments and line breaks allowed throughout).

An example is shown in Listing 171 with default output give in Listing 172.

<table>
<thead>
<tr>
<th>Listing 171: psforeach1.tex</th>
<th>Listing 172: psforeach1.tex default output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\psforeach{\row}{%</code></td>
<td><code>\psforeach{\row}{%</code></td>
</tr>
<tr>
<td><code>{3,2.8,2.7,3,3.1},%</code></td>
<td><code>{3,2.8,2.7,3,3.1},%</code></td>
</tr>
<tr>
<td><code>{2.8,1,1.2,2,3},%</code></td>
<td><code>{2.8,1,1.2,2,3},%</code></td>
</tr>
<tr>
<td>}</td>
<td><code>}</code></td>
</tr>
</tbody>
</table>

Referencing Listing 172, there are three sets of unnamed braces. Note also that the maximum value of indentation is three tabs, and these come from:

- the `\psforeach` command's mandatory argument;
- the first un-named braces mandatory argument;
- the first un-named braces body, which we define as any lines following the first opening \{ or [ that defined the code block. This is the part controlled by the body field for noAdditionalIndent and friends from page 31.

Users wishing to customise the mandatory and/or optional arguments on a per-name basis for the UnNamedGroupingBracesBrackets should use always-un-named.

**filecontents** code blocks behave just as environments, except that neither arguments nor items are sought.

5.2.8 **Summary**

Having considered all of the different types of code blocks, the functions of the fields given in Listings 173 and 174 should now make sense.

\(^8\) You may like to verify this by using the `-tt` option and checking `indent.log`!
5. Commands and the strings between their arguments

The command code blocks will always look for optional (square bracketed) and mandatory (curly braced) arguments which can contain comments, line breaks and 'beamer' commands \begin{.*}\end{.*} between them. There are switches that can allow them to contain other strings, which we discuss next.

commandCodeBlocks: (fields)

The commandCodeBlocks field contains a few switches detailed in Listing 175.

roundParenthesesAllowed: 0|1

The need for this field was mostly motivated by commands found in code used to generate images in PSTricks and \LaTeX{}; for example, let's consider the code given in Listing 176.

Notice that the \texttt{defFunction} command has an optional argument, followed by a mandatory argument, followed by a round-parenthesis argument, \( (u,v) \).

By default, because roundParenthesesAllowed is set to 1 in Listing 175, then \texttt{latexindent.pl} will allow round parenthesis between optional and mandatory arguments. In the case of the code in Listing 176, \texttt{latexindent.pl} finds \texttt{all} the arguments of \texttt{defFunction}, both before and after \( (u,v) \).
The default output from running `latexindent.pl` on Listing 176 actually leaves it unchanged (see Listing 177); note in particular, this is because of `noAdditionalIndentGlobal` as discussed on page 42.

Upon using the YAML settings in Listing 179, and running the command

```
$ latexindent.pl pstricks1.tex -l noRoundParentheses.yaml
```

we obtain the output given in Listing 178.

<table>
<thead>
<tr>
<th>Listing 178: pstricks1.tex using Listing 179</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\defFunction{torus}(u,v)</code></td>
</tr>
<tr>
<td><code>\{(2+\cos(u))\cos(v+\Pi)\}</code></td>
</tr>
<tr>
<td><code>\{(2+\cos(u))\sin(v+\Pi)\}</code></td>
</tr>
<tr>
<td><code>\{\sin(u)\}</code></td>
</tr>
</tbody>
</table>

Notice the difference between Listing 177 and Listing 178; in particular, in Listing 178, because round parentheses are not allowed, `latexindent.pl` finds that the `\defFunction` command finishes at the first opening round parenthesis. As such, the remaining braced, mandatory, arguments are found to be `UnnamedGroupingBracesBrackets` (see Table 1 on page 32) which, by default, assume indentation for their body, and hence the tabbed indentation in Listing 178.

Let’s explore this using the YAML given in Listing 181 and run the command

```
$ latexindent.pl pstricks1.tex -l defFunction.yaml
```

then the output is as in Listing 180.

<table>
<thead>
<tr>
<th>Listing 180: pstricks1.tex using Listing 181</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\defFunction{torus}(u,v)</code></td>
</tr>
<tr>
<td><code>\{(2+\cos(u))\cos(v+\Pi)\}</code></td>
</tr>
<tr>
<td><code>\{(2+\cos(u))\sin(v+\Pi)\}</code></td>
</tr>
<tr>
<td><code>\{\sin(u)\}</code></td>
</tr>
</tbody>
</table>

Notice in Listing 180 that the body of the `defFunction` command i.e, the subsequent lines containing arguments after the command name, have received the single space of indentation specified by Listing 181.

```
stringsAllowedBetweenArguments: (fields)
```

tikz users may well specify code such as that given in Listing 182; processing this code using `latexindent.pl` gives the default output in Listing 183.

<table>
<thead>
<tr>
<th>Listing 182: tikz-node1.tex</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\draw[thin]</code></td>
</tr>
<tr>
<td><code>(c)\to[in=110,out=-90]</code></td>
</tr>
<tr>
<td><code>++(0,-0.5cm)</code></td>
</tr>
<tr>
<td><code>node[below,align=left,scale=0.5]</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Listing 183: tikz-node1 default output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\draw[thin]</code></td>
</tr>
<tr>
<td><code>(c)\to[in=110,out=-90]</code></td>
</tr>
<tr>
<td><code>++(0,-0.5cm)</code></td>
</tr>
<tr>
<td><code>node[below,align=left,scale=0.5]</code></td>
</tr>
</tbody>
</table>

With reference to Listing 175 on the preceding page, we see that the strings

to, node, ++

are all allowed to appear between arguments; importantly, you are encouraged to add further names to this field as necessary. This means that when `latexindent.pl` processes Listing 182, it consumes:
• the optional argument [thin]
• the round-bracketed argument (c) because roundParenthesesAllowed is 1 by default
• the string to (specified in stringsAllowedBetweenArguments)
• the optional argument [in=110,out=-90]
• the string ++ (specified in stringsAllowedBetweenArguments)
• the round-bracketed argument (0,-0.5cm) because roundParenthesesAllowed is 1 by default
• the string node (specified in stringsAllowedBetweenArguments)
• the optional argument [below,align=left,scale=0.5]

We can explore this further, for example using Listing 185 and running the command

```
cmh:~$ latexindent.pl tikz-node1.tex -l draw.yaml
```

we receive the output given in Listing 184.

**Listing 184:** tikz-node1.tex using Listing 185

\draw[thin]
\( (c) \) to[in=110,out=-90]
++(0,-0.5cm)
node[below,align=left,scale=0.5]

**Listing 185:** draw.yaml

indentRules:
draw:
  body: " "

Notice that each line after the \draw command (its ‘body’) in Listing 184 has been given the appropriate two-spaces worth of indentation specified in Listing 185.

Let’s compare this with the output from using the YAML settings in Listing 187, and running the command

```
cmh:~$ latexindent.pl tikz-node1.tex -l no-strings.yaml
```

given in Listing 186.

**Listing 186:** tikz-node1.tex using Listing 187

\draw[thin]
\( (c) \) to[in=110,out=-90]
++(0,-0.5cm)
node[below,align=left,scale=0.5]

**Listing 187:** no-strings.yaml

commandCodeBlocks:
  stringsAllowedBetweenArguments: 0

In this case, latexindent.pl sees that:

• the \draw command finishes after the (c), as stringsAllowedBetweenArguments has been set to 0 so there are no strings allowed between arguments;

• it finds a namedGroupingBracesBrackets called to (see Table 1 on page 32) with argument [in=110,out=-90]

• it finds another namedGroupingBracesBrackets but this time called node with argument [below,align=left,scale=0.5]

Referencing Listing 175 on page 49, we see that the first field in the stringsAllowedBetweenArguments is amalgamate and is set to 1 by default. This is for users who wish to specify their settings in multiple YAML files. For example, by using the settings in either Listing 188 or Listing 189 is equivalent to using the settings in Listing 190.
We specify `amalgamate` to be set to 0 and in which case any settings loaded prior to those specified, including the default, will be overwritten. For example, using the settings in Listing 191 means that only the strings specified in that field will be used.

It is important to note that the `amalgamate` field, if used, must be in the first field, and specified using the syntax given in Listings 189 to 191.

We may explore this feature further with the code in Listing 192, whose default output is given in Listing 193.

Let’s compare this with the output from using the YAML settings in Listing 195, and running the command

```bash
cmh:~$ latexindent.pl for-each.tex -l foreach.yaml
```

given in Listing 194.

You might like to compare the output given in Listing 193 and Listing 194. Note, in particular, in Listing 193 that the `foreach` command has not included any of the subsequent strings, and that the braces have been treated as a `namedGroupingBracesBrackets`. In Listing 194 the `foreach`
command has been allowed to have \x/\y and \ in between arguments because of the settings given in Listing 195.

There are some special command names that do not fit within the names recognized by latexindent.pl, the first one of which is \@ifnextchar\[. From the perspective of latexindent.pl, the whole of the text \@ifnextchar\[ is a command, because it is immediately followed by sets of mandatory arguments. However, without the commandNameSpecial field, latexindent.pl would not be able to label it as such, because the [ is, necessarily, not matched by a closing ].

For example, consider the sample file in Listing 196, which has default output in Listing 197.

**Listing 196:** ifnextchar.tex
```
\parbox{
  \@ifnextchar[\{arg 1\}{arg 2}
}
```

**Listing 197:** ifnextchar.tex default output
```
\parbox{
  \@ifnextchar[\{arg 1\}{arg 2}
}
```

Notice that in Listing 197 the parbox command has been able to indent its body, because latexindent.pl has successfully found the command \@ifnextchar first; the pattern-matching of latexindent.pl starts from the inner most <thing> and works outwards, discussed in more detail on page 88.

For demonstration, we can compare this output with that given in Listing 198 in which the settings from Listing 199 have dictated that no special command names, including the \@ifnextchar[ command, should not be searched for specially; as such, the parbox command has been unable to indent its body successfully, because the \@ifnextchar[ command has not been found.

**Listing 198:** ifnextchar.tex using Listing 199
```
\parbox{
  \@ifnextchar[\{arg 1\}{arg 2}
}
```

**Listing 199:** no-ifnextchar.yaml
```
commandCodeBlocks:
  commandNameSpecial: 0
```

The amalgamate field can be used for commandNameSpecial, just as for stringsAllowedBetweenArguments. The same condition holds as stated previously, which we state again here:

- It is important to note that the amalgamate field, if used, in either commandNameSpecial or stringsAllowedBetweenArguments must be in the first field, and specified using the syntax given in Listings 189 to 191.

6 **The -m (modifylinebreaks) switch**

All features described in this section will only be relevant if the -m switch is used.

6.1 **textWrapOptions**: modifying line breaks by text wrapping ............................... 55
  6.1.1 text wrapping on a per-code-block basis .................................................. 57
  6.1.2 Summary of text wrapping ................................................................. 63
6.2 **oneSentencePerLine**: modifying line breaks for sentences ............................... 63
  6.2.1 sentencesFollow ......................................................................................... 65
  6.2.2 sentencesBeginWith ..................................................................................... 66
  6.2.3 sentencesEndWith ........................................................................................ 66
  6.2.4 Features of the oneSentencePerLine routine ............................................. 68
  6.2.5 text wrapping and indenting sentences ....................................................... 69
6.3 **removeParagraphLineBreaks**: modifying line breaks for paragraphs .................. 71
As of Version 3.0, latexindent.pl has the \texttt{-m} switch, which permits latexindent.pl to modify line breaks, according to the specifications in the \texttt{modifyLineBreaks} field. \textit{The settings in this field will only be considered if the \texttt{-m} switch has been used.} A snippet of the default settings of this field is shown in Listing 200.

Having read the previous paragraph, it should sound reasonable that, if you call latexindent.pl using the \texttt{-m} switch, then you give it permission to modify line breaks in your file, but let’s be clear:

\begin{itemize}
\item If you call latexindent.pl with the \texttt{-m} switch, then you are giving it permission to modify line breaks. By default, the only thing that will happen is that multiple blank lines will be condensed into one blank line; many other settings are possible, discussed next.
\end{itemize}

This field is directly related to \textit{poly-switches}, discussed below. By default, it is set to 1, which means that blank lines will be protected from removal; however, regardless of this setting, multiple blank lines can be condensed if \texttt{condenseMultipleBlankLinesInto} is greater than 0, discussed next.

Assuming that this switch takes an integer value greater than 0, latexindent.pl will condense multiple blank lines into the number of blank lines illustrated by this switch. As an example, Listing 201 shows a sample file with blank lines; upon running

\begin{verbatim}
cmh:$ latexindent.pl myfile.tex -m
\end{verbatim}

the output is shown in Listing 202; note that the multiple blank lines have been condensed into one blank line, and note also that we have used the \texttt{-m} switch!
6. THE -M (MODIFYLINEBREAKS) SWITCH

6.1 textWrapOptions: modifying line breaks by text wrapping

When the -m switch is active latexindent.pl has the ability to wrap text using the options specified in the textWrapOptions field, see Listing 203. The value of columns specifies the column at which the text should be wrapped. By default, the value of columns is 0, so latexindent.pl will not wrap text; if you change it to a value of 2 or more, then text will be wrapped after the character in the specified column.

For example, consider the file given in Listing 204.

Using the file textwrap1.yaml in Listing 206, and running the command

```
cmh:~$ latexindent.pl -m textwrap1.tex -o textwrap1-mod1.tex -l textwrap1.yaml
```

we obtain the output in Listing 205.

The text wrapping routine is performed after verbatim environments have been stored, so verbatim environments and verbatim commands are exempt from the routine. For example, using the file in Listing 207,
6. THE -M (MODIFYLINEBREAKS) SWITCH

<table>
<thead>
<tr>
<th>Listing 207: textwrap2.tex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Here is a line of text that will be wrapped by latexindent.pl. Each line is quite long.</td>
</tr>
</tbody>
</table>

```
\begin{verbatim}
  a long line in a verbatim environment, which will not be broken by latexindent.pl
\end{verbatim}
```

Here is a verb command: \verb!this will not be text wrapped!

and running the following command and continuing to use textwrap1.yaml from Listing 206,

```
cmh:~$ latexindent.pl -m textwrap2.tex -o textwrap2-mod1.tex -l textwrap1.yaml
```

then the output is as in Listing 208.

<table>
<thead>
<tr>
<th>Listing 208: textwrap2-mod1.tex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Here is a line of text that will be wrapped by latexindent.pl. Each line is quite long.</td>
</tr>
</tbody>
</table>

```
\begin{verbatim}
  a long line in a verbatim environment, which will not be broken by latexindent.pl
\end{verbatim}
```

Here is a verb command: \verb!this will not be text wrapped!

Furthermore, the text wrapping routine is performed after the trailing comments have been stored, and they are also exempt from text wrapping. For example, using the file in Listing 209

<table>
<thead>
<tr>
<th>Listing 209: textwrap3.tex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Here is a line of text that will be wrapped by latexindent.pl. Each line is quite long.</td>
</tr>
</tbody>
</table>

```
Here is a line % text wrapping does not apply to comments by latexindent.pl
```

and running the following command and continuing to use textwrap1.yaml from Listing 206,

```
cmh:~$ latexindent.pl -m textwrap3.tex -o textwrap3-mod1.tex -l textwrap1.yaml
```

then the output is as in Listing 210.

<table>
<thead>
<tr>
<th>Listing 210: textwrap3-mod1.tex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Here is a line of text that will be wrapped by latexindent.pl. Each line is quite long.</td>
</tr>
</tbody>
</table>

```
Here is a line % text wrapping does not apply to comments by latexindent.pl
```
The text wrapping routine of latexindent.pl is performed by the Text::Wrap module, which provides a separator feature to separate lines with characters other than a new line (see [16]). By default, the separator is empty which means that a new line token will be used, but you can change it as you see fit.

For example starting with the file in Listing 211

```latex
Here is a line of text.
```

and using textwrap2.yaml from Listing 213 with the following command

```
cmb:~$ latexindent.pl -m textwrap4.tex -o textwrap4-mod2.tex -l textwrap2.yaml
```

then we obtain the output in Listing 212.

```latex
Here||is a||line||of||text||.
```

### 6.1.1 text wrapping on a per-code-block basis

By default, if the value of `columns` is greater than 0 and the `-m` switch is active, then the text wrapping routine will operate before the code blocks have been searched for. This behaviour is customisable; in particular, you can instead instruct latexindent.pl to apply `textWrap` on a per-code-block basis. Thanks to [19] for their help in testing and shaping this feature.

The full details of `textWrapOptions` are shown in Listing 214. In particular, note the field `perCodeBlockBasis`: 0.

```yaml
modifyLineBreaks:  # -m
  textWrapOptions:
    columns: 0
    separator: ""
    perCodeBlockBasis: 0
    all: 0
    alignAtAmpersandTakesPriority: 1
    environments:
      quotation: 0
      ifElseFi: 0
      optionalArguments: 0
      mandatoryArguments: 0
      items: 0
      specialBeginEnd: 0
      afterHeading: 0
      preamble: 0
      filecontents: 0
      masterDocument: 0
```

The code blocks detailed in Listing 214 are with direct reference to those detailed in Table 1 on page 32. The only special case is the `masterDocument` field; this is designed for ‘chapter’-type files that may contain paragraphs that are not within any other code-blocks. The same notation is used between this feature and the removeParagraphLineBreaks described in Listing 275 on page 72; in fact, the two features can even be combined (this is detailed in Section 6.4 on page 77).

Let’s explore these switches with reference to the code given in Listing 215; the text outside of the environment is considered part of the `masterDocument`.

---

[git]  •  master @ 162e869  •  2018-10-13  •  ☛  •  V3.5.3
6. **THE -M (MODIFYLINEBREAKS) SWITCH**

### Listing 215: textwrap5.tex

Before the environment; here is a line of text that can be wrapped by latexindent.pl.

```latex
\begin{myenv}
Within the environment; here is a line of text that can be wrapped by latexindent.pl.
\end{myenv}
```

After the environment; here is a line of text that can be wrapped by latexindent.pl.

With reference to this codeblock, the settings given in Listings 216 to 218 each give the same output.

---

### Listing 216: textwrap3.yaml

```yaml
modifyLineBreaks:
  textWrapOptions:
    columns: 30
    perCodeBlockBasis: 1
    all: 1
```

### Listing 217: textwrap4.yaml

```yaml
modifyLineBreaks:
  textWrapOptions:
    columns: 30
    perCodeBlockBasis: 1
    environments: 1
    masterDocument: 1
```

### Listing 218: textwrap5.yaml

```yaml
modifyLineBreaks:
  textWrapOptions:
    columns: 30
    perCodeBlockBasis: 1
    environments:
      myenv: 1
      masterDocument: 1
```

Let’s explore the similarities and differences in the equivalent (with respect to Listing 215) syntax specified in Listings 216 to 218:

- in each of Listings 216 to 218 notice that columns: 30;
- in each of Listings 216 to 218 notice that perCodeBlockBasis: 1;
- in Listing 216 we have specified all: 1 so that the text wrapping will operate upon all code blocks;
- in Listing 217 we have not specified all, and instead, have specified that text wrapping should be applied to each of environments and masterDocument;
- in Listing 218 we have specified text wrapping for masterDocument and on a per-name basis for environments code blocks.

Upon running the following commands

```bash
cmh:~$ latexindent.pl -s textwrap5.tex -l=textwrap3.yaml -m
cmh:~$ latexindent.pl -s textwrap5.tex -l=textwrap4.yaml -m
cmh:~$ latexindent.pl -s textwrap5.tex -l=textwrap5.yaml -m
```

we obtain the output shown in Listing 219.

```latex
Before the environment; here is a line of text that can be wrapped by latexindent.pl.

\begin{myenv}
  Within the environment; here is a line of text that can be wrapped by latexindent.pl.
\end{myenv}
```

After the environment; here is a line of text that can be wrapped by latexindent.pl.

We can explore the idea of per-name text wrapping given in Listing 218 by using Listing 220.
6. THE -M (MODIFYLINEBREAKS) SWITCH

LISTING 220: textwrap6.tex

Before the environment; here is a line of text that can be wrapped by latexindent.pl.

\begin{myenv}
Within the environment; here is a line of text that can be wrapped by latexindent.pl.
\end{myenv}

\begin{another}
Within the environment; here is a line of text that can be wrapped by latexindent.pl.
\end{another}

After the environment; here is a line of text that can be wrapped by latexindent.pl.

In particular, upon running

```
cmh:~$ latexindent.pl -s textwrap6.tex -l=textwrap5.yaml -m
```

we obtain the output given in Listing 221.

LISTING 221: textwrap6.tex using Listing 218

Before the environment; here is a line of text that can be wrapped by latexindent.pl.

\begin{myenv}
  Within the environment; here is a line of text that can be wrapped by latexindent.pl.
\end{myenv}

\begin{another}
  Within the environment; here is a line of text that can be wrapped by latexindent.pl.
\end{another}

After the environment; here is a line of text that can be wrapped by latexindent.pl.

Notice that, because environments has been specified only for myenv (in Listing 218) that the environment named another has not had text wrapping applied to it.

The all field can be specified with exceptions which can either be done on a per-code-block or per-name basis; we explore this in relation to Listing 220 in the settings given in Listings 222 to 224.

LISTING 222: textwrap6.yaml

```
modifyLineBreaks:
  textWrapOptions:
    columns: 30
    perCodeBlockBasis: 1
    all:
      except:
        - environments
```

LISTING 223: textwrap7.yaml

```
modifyLineBreaks:
  textWrapOptions:
    columns: 30
    perCodeBlockBasis: 1
    all:
      except:
        - myenv
```

LISTING 224: textwrap8.yaml

```
modifyLineBreaks:
  textWrapOptions:
    columns: 30
    perCodeBlockBasis: 1
    all:
      except:
        - masterDocument
```

Upon running the commands
6. THE -M (MODIFYLINEBREAKS) SWITCH

we receive the respective output given in Listings 225 to 227.

<table>
<thead>
<tr>
<th>Listing 225: textwrap6.tex using Listing 222</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before the environment; here \</td>
</tr>
<tr>
<td>is a line of text that can be \</td>
</tr>
<tr>
<td>wrapped by latexindent.pl. \</td>
</tr>
<tr>
<td>\begin{myenv} \</td>
</tr>
<tr>
<td>\hspace{3em} Within the environment; here \</td>
</tr>
<tr>
<td>is a line of text that can be \</td>
</tr>
<tr>
<td>wrapped by latexindent.pl. \</td>
</tr>
<tr>
<td>\end{myenv} \</td>
</tr>
<tr>
<td>\begin{another} \</td>
</tr>
<tr>
<td>\hspace{3em} Within the environment; here \</td>
</tr>
<tr>
<td>is a line of text that can be \</td>
</tr>
<tr>
<td>wrapped by latexindent.pl. \</td>
</tr>
<tr>
<td>\end{another} \</td>
</tr>
<tr>
<td>After the environment; here \</td>
</tr>
<tr>
<td>is a line of text that can be \</td>
</tr>
<tr>
<td>wrapped by latexindent.pl. \</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Listing 226: textwrap6.tex using Listing 223</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before the environment; here \</td>
</tr>
<tr>
<td>is a line of text that can be \</td>
</tr>
<tr>
<td>wrapped by latexindent.pl. \</td>
</tr>
<tr>
<td>\begin{myenv} \</td>
</tr>
<tr>
<td>\hspace{3em} Within the environment; here \</td>
</tr>
<tr>
<td>is a line of text that can be \</td>
</tr>
<tr>
<td>wrapped by latexindent.pl. \</td>
</tr>
<tr>
<td>\end{myenv} \</td>
</tr>
<tr>
<td>\begin{another} \</td>
</tr>
<tr>
<td>\hspace{3em} Within the environment; here \</td>
</tr>
<tr>
<td>is a line of text that can be \</td>
</tr>
<tr>
<td>wrapped by latexindent.pl. \</td>
</tr>
<tr>
<td>\end{another} \</td>
</tr>
<tr>
<td>After the environment; here \</td>
</tr>
<tr>
<td>is a line of text that can be \</td>
</tr>
<tr>
<td>wrapped by latexindent.pl. \</td>
</tr>
</tbody>
</table>
6. THE -M (MODIFYLINEBREAKS) SWITCH

Before the environment; here is a line of text that can be wrapped by \texttt{latexindent.pl}.

\begin{myenv}
\begin{verbatim}
\text{Within the environment; here is a line of text that can be wrapped by latexindent.pl.}
\end{verbatim}
\end{myenv}

\begin{another}
\begin{verbatim}
\text{Within the environment; here is a line of text that can be wrapped by latexindent.pl.}
\end{verbatim}
\end{another}

After the environment; here is a line of text that can be wrapped by \texttt{latexindent.pl}.

Notice that:

- in Listing 225 the text wrapping routine has not been applied to any environments because it has been switched off (per-code-block) in Listing 222;
- in Listing 226 the text wrapping routine has not been applied to \texttt{myenv} because it has been switched off (per-name) in Listing 223;
- in Listing 227 the text wrapping routine has not been applied to \texttt{masterDocument} because of the settings in Listing 224.

The \texttt{columns} field has a variety of different ways that it can be specified; we've seen two basic ways already: the default (set to 0) and a positive integer (see Listing 220 on page 59, for example). We explore further options in Listings 228 to 230.

Listing 228 and Listing 229 are equivalent. Upon running the commands

\begin{verbatim}
\texttt{cmh:\sim \$ latexindent.pl}\ -s\ \texttt{textwrap6.tex}\ -l=textwrap9.yaml\ -m
\texttt{cmh:\sim \$ latexindent.pl}\ -s\ \texttt{textwrap6.tex}\ -l=textwrap10.yaml\ -m
\end{verbatim}

we receive the respective output given in Listings 231 and 232.
Notice that:

- in Listing 231 the text for the `masterDocument` has been wrapped using 30 columns, while `environments` has been wrapped using 50 columns;
- in Listing 232 the text for `myenv` has been wrapped using 50 columns, the text for `another` has been wrapped using 15 columns, and `masterDocument` has been wrapped using 30 columns.

If you don't specify a default value on per-code-block basis, then the default value from `columns` will be inherited; if you don't specify a default value for `columns` then 80 will be used.

`alignAtAmpersandTakesPriority` is set to 1 by default; assuming that text wrapping is occurring on a per-code-block basis, and the current environment/code block is specified within Listing 25 on page 21 then text wrapping will be disabled for this code block.

If you wish to specify `afterHeading` commands (see Listing 68 on page 29) on a per-name basis,
then you need to append the name with :heading, for example, you might use section:heading.

### 6.1.2 Summary of text wrapping
It is important to note the following:

- Verbatim environments (Listing 16 on page 19) and verbatim commands (Listing 17 on page 19) will *not* be affected by the text wrapping routine (see Listing 208 on page 56);  
- comments will *not* be affected by the text wrapping routine (see Listing 210 on page 56);  
- it is possible to wrap text on a per-code-block and a per-name basis;  
- the text wrapping routine sets preserveBlankLines as 1;  
- indentation is performed *after* the text wrapping routine; as such, indented code will likely exceed any maximum value set in the columns field.

### 6.2 oneSentencePerLine: modifying line breaks for sentences
You can instruct latexindent.pl to format your file so that it puts one sentence per line. Thank you to [11] for helping to shape and test this feature. The behaviour of this part of the script is controlled by the switches detailed in Listing 233, all of which we discuss next.

**Listing 233: oneSentencePerLine**

```bash
397  oneSentencePerLine:
398      manipulateSentences: 0
399      removeSentenceLineBreaks: 1
400      textWrapSentences: 0
401      sentenceIndent: ""
402      sentencesFollow:
403          par: 1
404          blankLine: 1
405          fullStop: 1
406          exclamationMark: 1
407          questionMark: 1
408          rightBrace: 1
409          commentOnPreviousLine: 1
410          other: 0
411  sentencesBeginWith:
412      A-Z: 1
413      a-z: 0
414      other: 0
415  sentencesEndWith:
416      basicFullStop: 0
417      betterFullStop: 1
418      exclamationMark: 1
419      questionMark: 1
420      other: 0
```

**manipulateSentences: 0|1**

This is a binary switch that details if latexindent.pl should perform the sentence manipulation routine; it is *off* (set to 0) by default, and you will need to turn it on (by setting it to 1) if you want the script to modify line breaks surrounding and within sentences.

**removeSentenceLineBreaks: 0|1**

When operating upon sentences latexindent.pl will, by default, remove internal linebreaks as removeSentenceLineBreaks is set to 1. Setting this switch to 0 instructs latexindent.pl not to do so.

For example, consider multiple-sentences.tex shown in Listing 234.
6. THE -M (MODIFYLINEBREAKS) SWITCH

<table>
<thead>
<tr>
<th>LISTING 234: multiple-sentences.tex</th>
</tr>
</thead>
<tbody>
<tr>
<td>This is the first sentence. This is the; second, sentence. This is the third sentence.</td>
</tr>
<tr>
<td>This is the fourth sentence! This is the fifth sentence? This is the sixth sentence.</td>
</tr>
</tbody>
</table>

If we use the YAML files in Listings 236 and 238, and run the commands

```
cmh:~$ latexindent.pl multiple-sentences -m -l=manipulate-sentences.yaml
```

```
cmh:~$ latexindent.pl multiple-sentences -m -l=keep-sen-line-breaks.yaml
```

then we obtain the respective output given in Listings 235 and 237.

<table>
<thead>
<tr>
<th>LISTING 235: multiple-sentences.tex</th>
</tr>
</thead>
<tbody>
<tr>
<td>using Listing 236</td>
</tr>
<tr>
<td>This is the first sentence. This is the; second, sentence. This is the third sentence.</td>
</tr>
<tr>
<td>This is the fourth sentence! This is the fifth sentence? This is the sixth sentence.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LISTING 236: manipulate-sentences.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>modifyLineBreaks:</td>
</tr>
<tr>
<td>oneSentencePerLine:</td>
</tr>
<tr>
<td>manipulateSentences: 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LISTING 237: multiple-sentences.tex</th>
</tr>
</thead>
<tbody>
<tr>
<td>using Listing 238</td>
</tr>
<tr>
<td>This is the first sentence. This is the; second, sentence. This is the third sentence.</td>
</tr>
<tr>
<td>This is the fourth sentence! This is the fifth sentence? This is the sixth sentence.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LISTING 238: keep-sen-line-breaks.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>modifyLineBreaks:</td>
</tr>
<tr>
<td>oneSentencePerLine:</td>
</tr>
<tr>
<td>manipulateSentences: 1</td>
</tr>
<tr>
<td>removeSentenceLineBreaks: 0</td>
</tr>
</tbody>
</table>

Notice, in particular, that the ‘internal’ sentence line breaks in Listing 234 have been removed in Listing 235, but have not been removed in Listing 237.

The remainder of the settings displayed in Listing 233 on the previous page instruct latexindent.pl on how to define a sentence. From the perspective of latexindent.pl a sentence must:

- **follow** a certain character or set of characters (see Listing 239); by default, this is either `\par`, a blank line, a full stop/period (.), exclamation mark (!), question mark (?) right brace (}), or a comment on the previous line;
- **begin** with a character type (see Listing 240); by default, this is only capital letters;
- **end** with a character (see Listing 241); by default, these are full stop/period (.), exclamation mark (!) and question mark (?).

In each case, you can specify the `other` field to include any pattern that you would like; you can specify anything in this field using the language of regular expressions.
6. THE -M (MODIFYLINEBREAKS) SWITCH

6.2.1 sentencesFollow

Let’s explore a few of the switches in sentencesFollow; let’s start with Listing 234 on the preceding page, and use the YAML settings given in Listing 243. Using the command

```
cmh:~$ latexindent.pl multiple-sentences -m -l=sentences-follow1.yaml
```

we obtain the output given in Listing 242.

```
<table>
<thead>
<tr>
<th>Listing 242: multiple-sentences.tex using Listing 243</th>
</tr>
</thead>
<tbody>
<tr>
<td>This is the first sentence.</td>
</tr>
<tr>
<td>This is the; second, sentence.</td>
</tr>
<tr>
<td>This is the third sentence.</td>
</tr>
<tr>
<td>This is the fourth sentence!</td>
</tr>
<tr>
<td>This is the fifth sentence?</td>
</tr>
<tr>
<td>This is the sixth sentence.</td>
</tr>
</tbody>
</table>
```

Notice that, because blankLine is set to 0, latexindent.pl will not seek sentences following a blank line, and so the fourth sentence has not been accounted for.

We can explore the other field in Listing 239 with the .tex file detailed in Listing 244.

```
<table>
<thead>
<tr>
<th>Listing 244: multiple-sentences1.tex</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Some sentences stand alone in brackets.) This is the first sentence. This is the; second, sentence. This is the third sentence.</td>
</tr>
</tbody>
</table>
```

Upon running the following commands

```
cmh:~$ latexindent.pl multiple-sentences1 -m -l=manipulate-sentences.yaml
```

```
cmh:~$ latexindent.pl multiple-sentences1 -m -l=manipulate-sentences.yaml,sentences-follow2.yaml
```

then we obtain the respective output given in Listings 245 and 246.

```
<table>
<thead>
<tr>
<th>Listing 245: multiple-sentences1.tex using Listing 236 on the previous page</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Some sentences stand alone in brackets.) This is the first sentence. This is the; second, sentence. This is the third sentence.</td>
</tr>
</tbody>
</table>
```
6. THE -M (MODIFYLINEBREAKS) SWITCH

6.2.2 sentencesBeginWith

By default, latexindent.pl will only assume that sentences begin with the upper case letters A-Z; you can instruct the script to define sentences to begin with lower case letters (see Listing 240), and we can use the other field to define sentences to begin with other characters.

Upon running the following commands

```
cmh:~$ latexindent.pl multiple-sentences2 -m -l=manipulate-sentences.yaml
```
```
cmh:~$ latexindent.pl multiple-sentences2 -m -l=manipulate-sentences.yaml,sentences-begin1.yaml
```
```
cmh:~$
```

then we obtain the respective output given in Listings 249 and 250.

Notice that in Listing 249, the first sentence has been accounted for but that the subsequent sentences have not. In Listing 250, all of the sentences have been accounted for, because the other field in Listing 251 has defined sentences to begin with either $ or any numeric digit, 0 to 9.

6.2.3 sentencesEndWith

Let's return to Listing 234 on page 64; we have already seen the default way in which latexindent.pl will operate on the sentences in this file in Listing 235 on page 64. We can populate the other field with any character that we wish; for example, using the YAML specified in Listing 253 and the command
then we obtain the output in Listing 252.

```
\begin{verbatim}
\texttt{cmh:~}\$ latexindent.pl multiple-sentences -m -l=sentences-end1.yaml
\texttt{cmh:~}\$ latexindent.pl multiple-sentences -m -l=sentences-end2.yaml
\end{verbatim}
```

```
\begin{verbatim}
\texttt{\textbackslash listing252: multiple-sentences.tex using Listing 253}
This is the first sentence.
This is the; second, sentence.
This is the third sentence.
This is the fourth sentence!
This is the fifth sentence?
This is the sixth sentence.
\end{verbatim}
```

```
\begin{verbatim}
\texttt{\textbackslash listing253: sentences-end1.yaml}
modifyLineBreaks:
oneSentencePerLine:
manipulateSentences: 1
sentencesEndWith:
other: ";|;|,`
\end{verbatim}
```

```
\begin{verbatim}
\texttt{\textbackslash listing254: multiple-sentences.tex using Listing 255}
This is the first sentence.
This is the; second, sentence.
This is the third sentence.
This is the fourth sentence!
This is the fifth sentence?
This is the sixth sentence.
\end{verbatim}
```

```
\begin{verbatim}
\texttt{\textbackslash listing255: sentences-end2.yaml}
modifyLineBreaks:
oneSentencePerLine:
manipulateSentences: 1
sentencesEndWith:
other: ";|;|,`
sentencesBeginWith:
a-z: 1
\end{verbatim}
```

There is a subtle difference between the output in Listings 252 and 254; in particular, in Listing 252 the word sentence has not been defined as a sentence, because we have not instructed latexindent.pl to begin sentences with lower case letters. We have changed this by using the settings in Listing 255, and the associated output in Listing 254 reflects this.

Referencing Listing 241 on page 65, you’ll notice that there is a field called basicFullStop, which is set to 0, and that the betterFullStop is set to 1 by default.

Let’s consider the file shown in Listing 256.

```
\begin{verbatim}
\texttt{\textbackslash listing256: url.tex}
This sentence, \url{tex.stackexchange.com/} finishes here. Second sentence.
\end{verbatim}
```

Upon running the following commands

```
\texttt{\textbackslash cmh:~}\$ latexindent.pl url -m -l=manipulate-sentences.yaml
\texttt{\textbackslash cmh:~}\$ latexindent.pl url -m -l=manipulate-sentences.yaml
\texttt{\textbackslash cmh:~}\$ latexindent.pl url -m -l=manipulate-sentences.yaml
\texttt{\textbackslash cmh:~}\$ latexindent.pl url -m -l=manipulate-sentences.yaml
\texttt{\textbackslash cmh:~}\$ latexindent.pl url -m -l=manipulate-sentences.yaml
\texttt{\textbackslash cmh:~}\$ latexindent.pl url -m -l=manipulate-sentences.yaml
\texttt{\textbackslash cmh:~}\$ latexindent.pl url -m -l=manipulate-sentences.yaml
```

```
\begin{verbatim}
\texttt{\textbackslash listing257: url.tex using Listing 236 on page 64}
This sentence, \url{tex.stackexchange.com/} finishes here. Second sentence.
\end{verbatim}
```

we obtain the output given in Listing 257.

```
\begin{verbatim}
\texttt{\textbackslash listing258: url.tex}
This sentence, \url{tex.stackexchange.com/} finishes here. Second sentence.
\end{verbatim}
```

Notice that the full stop within the url has been interpreted correctly. This is because, within the betterFullStop, full stops at the end of sentences have the following properties:

- they are ignored within e.g. and i.e.;
- they can not be immediately followed by a lower case or upper case letter;
- they can not be immediately followed by a hyphen, comma, or number.
If you find that the `betterFullStop` does not work for your purposes, then you can switch it off by setting it to 0, and you can experiment with the other field.

The `basicFullStop` routine should probably be avoided in most situations, as it does not accommodate the specifications above. For example, using the following command:

```
cmh:~$ latexindent.pl url -m -l=alt-full-stop1.yaml
```

and the YAML in Listing 259 gives the output in Listing 258.

---

**Listing 258: url.tex using Listing 259**

This sentence, \url{tex.stackexchange.com/} finishes here. Second sentence.

---

Notice that the full stop within the URL has not been accommodated correctly because of the non-default settings in Listing 259.

### 6.2.4 Features of the `oneSentencePerLine` routine

The sentence manipulation routine takes place after verbatim environments, preamble and trailing comments have been accounted for; this means that any characters within these types of code blocks will not be part of the sentence manipulation routine.

For example, if we begin with the `.tex` file in Listing 260, and run the command:

```
cmh:~$ latexindent.pl multiple-sentences3 -m -l=manipulate-sentences.yaml
cmh:~$
```

then we obtain the output in Listing 261.

---

**Listing 260: multiple-sentences3.tex**

The first sentence continues after the verbatim `\begin{verbatim} there are sentences within this. These will not be operated upon by latexindent.pl. \end{verbatim}` and finishes here. Second sentence % a commented full stop. contains trailing comments, which are ignored.

---

**Listing 261: multiple-sentences3.tex using Listing 236 on page 64**

The first sentence continues after the verbatim `\begin{verbatim} there are sentences within this. These will not be operated upon by latexindent.pl. \end{verbatim}` and finishes here. Second sentence contains trailing comments, which are ignored. % a commented full stop.

Furthermore, if sentences run across environments then, by default, the line breaks internal to the sentence will be removed. For example, if we use the `.tex` file in Listing 262 and run the commands:

```cmh:~$ latexindent.pl url -m -l=alt-full-stop1.yaml
```
then we obtain the output in Listings 263 and 264.

\begin{itemize}
\item continues
\end{itemize}
across itemize and finishes here.

Once you've read Section 6.5, you will know that you can accommodate the removal of internal sentence line breaks by using the YAML in Listing 266 and the command

\begin{itemize}
\item continues
\end{itemize}
across itemize and finishes here.

the output of which is shown in Listing 265.

\begin{itemize}
\item continues
\end{itemize}
across itemize and finishes here.

6.2.5 text wrapping and indenting sentences

The oneSentencePerLine can be instructed to perform text wrapping and indentation upon sentences.

Let's use the code in Listing 267.
A distinção entre conteúdo \textit{real} e conteúdo \textit{intencional} está relacionada, ainda, à distinção entre o conceito husserliano de \textit{experiência} e o uso popular desse termo. No sentido comum, o \texttt{experimentado} é um complexo de eventos exteriores, e o \texttt{experimentar} consiste em percepções (além de julgamentos e outros atos) nas quais tais eventos aparecem como objetos, e objetos frequentemente relacionados ao ego empírico.

Referencing Listing 269, and running the following command

\begin{verbatim}
    cmh:\$ latexindent.pl multiple-sentences5 -m -l=sentence-wrap1.yaml
\end{verbatim}

we receive the output given in Listing 268.

If you wish to specify the columns field on a per-code-block basis for sentences, then you would use sentence; explicitly, starting with Listing 228 on page 61, for example, you would replace /append environments with, for example, sentence: 50.

The indentation of sentences requires that sentences are stored as code blocks. This means that you may need to tweak Listing 241 on page 65. Let's explore this in relation to Listing 270.

Consider the following:
\begin{itemize}
  \item firstly.
  \item secondly.
\end{itemize}

By default, latexindent.pl will find the full-stop within the first item, which means that, upon running the following commands

\begin{verbatim}
    cmh:\$ latexindent.pl multiple-sentences6 -m -l=sentence-wrap1.yaml
    cmh:\$ latexindent.pl multiple-sentences6 -m -l=sentence-wrap1.yaml
    -y="modifyLineBreaks:oneSentencePerLine:sentenceIndent:/'""
\end{verbatim}

we receive the respective output in Listing 271 and Listing 272.
6. THE \texttt{-M (MODIFYLINEBREAKS)} SWITCH

Consider the following:

\begin{itemize}
  \item firstly.
  \item secondly.
\end{itemize}

We note that Listing 271 the itemize code block has not been indented appropriately. This is because the oneSentencePerLine has been instructed to store sentences (because Listing 269); each sentence is then searched for code blocks.

We can tweak the settings in Listing 241 on page 65 to ensure that full stops are not followed by item commands, and that the end of sentences contains \texttt{\end{itemize}} as in Listing 273 (if you intend to use this, ensure that you remove the line breaks from the other field).

Upon running

```
\texttt{cmh:~}\$ latexindent.pl multiple-sentences6 -m -l=sentence-wrap1.yaml,itemize.yaml
```

we receive the output in Listing 274.

Consider the following:

\begin{itemize}
  \item firstly. \item secondly.
\end{itemize}

Notice that the sentence has received indentation, and that the itemize code block has been found and indented correctly.

6.3 \texttt{removeParagraphLineBreaks (fields)}

When the \texttt{-m} switch is active latexindent.pl has the ability to remove line breaks from within paragraphs; the behaviour is controlled by the \texttt{removeParagraphLineBreaks} field, detailed in Listing 275. Thank you to [12] for shaping and assisting with the testing of this feature.

This feature is considered complimentary to the oneSentencePerLine feature described in Section 6.2 on page 63.
This routine can be turned on *globally* for *every* code block type known to `latexindent.pl` (see Table 1 on page 32) by using the `all` switch; by default, this switch is *off*. Assuming that the `all` switch is off, then the routine can be controlled on a per-code-block-type basis, and within that, on a per-name basis. We will consider examples of each of these in turn, but before we do, let’s specify what `latexindent.pl` considers as a paragraph:

- it must begin on its own line with either an alphabetic or numeric character, and not with any of the code-block types detailed in Table 1 on page 32;
- it can include line breaks, but finishes when it meets either a blank line, a \par command, or any of the user-specified settings in the `paragraphsStopAt` field, detailed in Listing 292 on page 76.

Let’s start with the `.tex` file in Listing 276, together with the YAML settings in Listing 277.

Upon running the command

```bash
$ latexindent.pl -m shortlines.tex -o shortlines1.tex -l remove-para1.yaml
```

then we obtain the output given in Listing 278.
Keen readers may notice that some trailing white space must be present in the file in Listing 276 which has crept in to the output in Listing 278. This can be fixed using the YAML file in Listing 343 on page 83 and running, for example,

```
cmh:~$ latexindent.pl -m shortlines.tex -o shortlines1-tws.tex -l remove-para1.yaml,removeTWS-before.yaml
```

in which case the output is as in Listing 279; notice that the double spaces present in Listing 278 have been addressed.

```
Listing 279: shortlines1-tws.tex

\begin{myenv}
  The lines in this environment are very short and contain many linebreaks.

  Another paragraph.
\end{myenv}
```

Keeping with the settings in Listing 277, we note that the all switch applies to all code block types. So, for example, let's consider the files in Listings 280 and 281

```
Listing 280: shortlines-mand.tex
\mycommand{
  The lines in this command are very short and contain many linebreaks.

  Another paragraph.
}
```

```
Listing 281: shortlines-opt.tex
\mycommand[
  The lines in this command are very short and contain many linebreaks.

  Another paragraph.
]
```

Upon running the commands

```
cmh:~$ latexindent.pl -m shortlines-mand.tex -o shortlines-mand1.tex -l remove-para1.yaml
cmh:~$ latexindent.pl -m shortlines-opt.tex -o shortlines-opt1.tex -l remove-para1.yaml
```

then we obtain the respective output given in Listings 282 and 283.

```
Listing 282: shortlines-mand1.tex
\mycommand{
  The lines in this command are very short and contain many linebreaks.

  Another paragraph.
}
```

```
Listing 283: shortlines-opt1.tex
\mycommand[
  The lines in this command are very short and contain many linebreaks.

  Another paragraph.
]
```

Assuming that we turn off the all switch (by setting it to 0), then we can control the behaviour of removeParagraphLineBreaks either on a per-code-block-type basis, or on a per-name basis.
For example, let’s use the code in Listing 284, and consider the settings in Listings 285 and 286; note that in Listing 285 we specify that every environment should receive treatment from the routine, while in Listing 286 we specify that only the one environment should receive the treatment.

**Listing 284: shortlines-envs.tex**
```
\begin{one}
The lines in this environment are very short and contain many linebreaks.

Another paragraph.
\end{one}
```
```
\begin{two}
The lines in this environment are very short and contain many linebreaks.

Another paragraph.
\end{two}
```

Upon running the commands
```
cmh:~$ latexindent.pl -m shortlines-envs.tex -o shortlines-envs2.tex -l remove-para2.yaml
cmh:~$ latexindent.pl -m shortlines-envs.tex -o shortlines-envs3.tex -l remove-para3.yaml
```
then we obtain the respective output given in Listings 287 and 288.

**Listing 287: shortlines-envs2.tex**
```
\begin{one}
The lines in this environment are very short and contain many linebreaks.

Another paragraph.
\end{one}
```
```
\begin{two}
The lines in this environment are very short and contain many linebreaks.

Another paragraph.
\end{two}
```
The remaining code-block types can be customized in analogous ways, although note that commands, keyEqualsValuesBracesBrackets, namedGroupingBracesBrackets, UnNamedGroupingBracesBrackets are controlled by the optionalArguments and the mandatoryArguments.

The only special case is the masterDocument field; this is designed for ‘chapter’-type files that may contain paragraphs that are not within any other code-blocks. For example, consider the file in Listing 289, with the YAML settings in Listing 290.

Listing 289: \texttt{shortlines-md.tex}

\begin{myenv}
\begin{one}
The lines in this environment are very short and contain many linebreaks.

Another paragraph.
\end{one}
\begin{two}
The lines in this environment are very short and contain many linebreaks.

Another paragraph.
\end{two}
\end{myenv}

Upon running the following command

\texttt{cmh:~$ latexindent.pl \(-m\) shortlines-md.tex \(-o\) shortlines-md4.tex \(-l\) remove-para4.yaml}

then we obtain the output in Listing 291.
The lines in this document are very short and contain many linebreaks.

Another paragraph.

\begin{myenv}
  The lines in this document are very short and contain many linebreaks.
\end{myenv}

Note that the all field can take the same exceptions detailed in Listing \ref{lst:texwrap8-yaml}.

The paragraph line break routine considers blank lines and the \texttt{\textbackslash par} command to be the end of a paragraph; you can fine tune the behaviour of the routine further by using the paragraphsStopAt fields, shown in Listing \ref{lst:paragraphsStopAt}.

The fields specified in paragraphsStopAt tell \texttt{latexindent.pl} to stop the current paragraph when it reaches a line that begins with any of the code-block types specified as 1 in Listing \ref{lst:paragraphsStopAt}. By default, you'll see that the paragraph line break routine will stop when it reaches an environment or verbatim code block at the beginning of a line. It is not possible to specify these fields on a per-name basis.

Let's use the \texttt{.tex} file in Listing \ref{lst:sl-stop.tex}; we will, in turn, consider the settings in Listings \ref{lst:stop-comment.yaml} and \ref{lst:stop-comment.yaml}.

Upon using the settings from Listing \ref{lst:modifyLineBreaks} on the preceding page and running the commands
we obtain the respective outputs in Listings 296 to 298; notice in particular that:

- in Listing 296 the paragraph line break routine has included commands and comments;
- in Listing 297 the paragraph line break routine has stopped at the \texttt{emph} command, because in Listing 294 we have specified commands to be 1, and \texttt{emph} is at the beginning of a line;
- in Listing 298 the paragraph line break routine has stopped at the comments, because in Listing 295 we have specified comments to be 1, and the comment is at the beginning of a line.

In all outputs in Listings 296 to 298 we notice that the paragraph line break routine has stopped at \texttt{\begin{myenv}} because, by default, environments is set to 1 in Listing 292 on the previous page.

---

**Listing 296:** \texttt{sl-stop4.tex}

These lines are very short \texttt{\texttt{\emph{and}}} contain many linebreaks.

\begin{myenv}
  Body of myenv
\end{myenv}

Another paragraph. \% a comment \% a comment

**Listing 297:** \texttt{sl-stop4-command.tex}

These lines are very short \texttt{\texttt{\emph{and}}} contain many linebreaks.

\begin{myenv}
  Body of myenv
\end{myenv}

Another paragraph. \% a comment \% a comment

**Listing 298:** \texttt{sl-stop4-comment.tex}

These lines are very short \texttt{\texttt{\emph{and}}} contain many linebreaks.

\begin{myenv}
  Body of myenv
\end{myenv}

Another paragraph. \% a comment \% a comment

---

### 6.4 Combining removeParagraphLineBreaks and textWrapOptions

The text wrapping routine (Section 6.1 on page 55) and remove paragraph line breaks routine (Section 6.3 on page 71) can be combined.

We motivate this feature with the code given in Listing 299.

**Listing 299:** \texttt{textwrap7.tex}

This paragraph has line breaks throughout its paragraph; we would like to combine the textwrapping and paragraph removal routine.
Applying the text wrap routine from Section 6.1 on page 55 with, for example, Listing 216 on page 58 gives the output in Listing 300.

**Listing 300: textwrap7.tex using Listing 216**

This paragraph has line breaks throughout its paragraph; we would like to combine the textwrapping and paragraph removal routine.

The text wrapping routine has behaved as expected, but it may be desired to remove paragraph line breaks before performing the text wrapping routine. The desired behaviour can be achieved by employing the beforeTextWrap switch.

Explicitly, using the settings in Listing 302 and running the command

```
cmh:~$ latexindent.pl -m textwrap7.tex -l=textwrap12.yaml -o=+-mod12
```

we obtain the output in Listing 301.

**Listing 301: textwrap7-mod12.tex**

This paragraph has line breaks throughout its paragraph; we would like to combine the textwrapping and paragraph removal routine.

In Listing 301 the paragraph linebreaks have first been removed from Listing 299, and then the text wrapping routine has been applied. It is envisaged that variants of Listing 302 will be among the most useful settings for these two features.

### 6.5 Poly-switches

Every other field in the modifyLineBreaks field uses poly-switches, and can take one of five integer values:

- **-1 remove mode**: line breaks before or after the `</part of thing>` can be removed (assuming that preserveBlankLines is set to 0);
- **0 off mode**: line breaks will not be modified for the `</part of thing>` under consideration;
- **1 add mode**: a line break will be added before or after the `</part of thing>` under consideration, assuming that there is not already a line break before or after the `</part of thing>`;
- **2 comment then add mode**: a comment symbol will be added, followed by a line break before or after the `</part of thing>` under consideration, assuming that there is not already a comment and line break before or after the `</part of thing>`;
- **3 add then blank line mode**: a line break will be added before or after the `</part of thing>` under consideration, assuming that there is not already a line break before or after the `</part of thing>`, followed by a blank line.

In the above, `</part of thing>` refers to either the `begin statement`, `body` or `end statement` of the code blocks detailed in Table 1 on page 32. All poly-switches are off by default; `latexindent.pl` searches first of all for per-name settings, and then followed by global per-thing settings.
6.6 modifyLineBreaks for environments

We start by viewing a snippet of defaultSettings.yaml in Listing 303; note that it contains global settings (immediately after the environments field) and that per-name settings are also allowed – in the case of Listing 303, settings for equation* have been specified. Note that all poly-switches are off by default.

<table>
<thead>
<tr>
<th>Listing 303: environments</th>
</tr>
</thead>
<tbody>
<tr>
<td>environments:</td>
</tr>
<tr>
<td>BeginStartsOnOwnLine: 0</td>
</tr>
<tr>
<td>BodyStartsOnOwnLine: 0</td>
</tr>
<tr>
<td>EndStartsOnOwnLine: 0</td>
</tr>
<tr>
<td>EndFinishesWithLineBreak: 0</td>
</tr>
<tr>
<td>equation*:</td>
</tr>
<tr>
<td>BeginStartsOnOwnLine: 0</td>
</tr>
<tr>
<td>BodyStartsOnOwnLine: 0</td>
</tr>
<tr>
<td>EndStartsOnOwnLine: 0</td>
</tr>
<tr>
<td>EndFinishesWithLineBreak: 0</td>
</tr>
</tbody>
</table>

Let's begin with the simple example given in Listing 304; note that we have annotated key parts of the file using ♠, ♥, ♦ and ♣, these will be related to fields specified in Listing 303.

<table>
<thead>
<tr>
<th>Listing 304: env-mlb1.tex</th>
</tr>
</thead>
<tbody>
<tr>
<td>before words♠ \begin{myenv}body of myenv\end{myenv}♣ after words</td>
</tr>
</tbody>
</table>

6.6.1 Adding line breaks; BeginStartsOnOwnLine and BodyStartsOnOwnLine

Let's explore BeginStartsOnOwnLine and BodyStartsOnOwnLine in Listings 305 and 306, and in particular, let's allow each of them in turn to take a value of 1.

<table>
<thead>
<tr>
<th>Listing 305: env-mlb1.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>modifyLineBreaks:</td>
</tr>
<tr>
<td>environments:</td>
</tr>
<tr>
<td>BeginStartsOnOwnLine: 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Listing 306: env-mlb2.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>modifyLineBreaks:</td>
</tr>
<tr>
<td>environments:</td>
</tr>
<tr>
<td>BodyStartsOnOwnLine: 1</td>
</tr>
</tbody>
</table>

After running the following commands,

```
$ latexindent.pl -m env-mlb.tex -l env-mlb1.yaml
$ latexindent.pl -m env-mlb.tex -l env-mlb2.yaml
```

the output is as in Listings 307 and 308 respectively.

<table>
<thead>
<tr>
<th>Listing 307: env-mlb.tex using Listing 305</th>
</tr>
</thead>
<tbody>
<tr>
<td>before words♠ \begin{myenv}body of myenv\end{myenv}♣ after words</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Listing 308: env-mlb.tex using Listing 306</th>
</tr>
</thead>
<tbody>
<tr>
<td>before words \begin{myenv}body of myenv\end{myenv} after words</td>
</tr>
</tbody>
</table>

There are a couple of points to note:

- in Listing 307 a line break has been added at the point denoted by ♠ in Listing 304; no other line breaks have been changed;
- in Listing 308 a line break has been added at the point denoted by ♥ in Listing 304; furthermore, note that the body of myenv has received the appropriate (default) indentation.

Let's now change each of the 1 values in Listings 305 and 306 so that they are 2 and save them into env-mlb3.yaml and env-mlb4.yaml respectively (see Listings 309 and 310).

<table>
<thead>
<tr>
<th>Listing 309: env-mlb3.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>modifyLineBreaks:</td>
</tr>
<tr>
<td>environments:</td>
</tr>
<tr>
<td>BeginStartsOnOwnLine: 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Listing 310: env-mlb4.yaml</th>
</tr>
</thead>
<tbody>
<tr>
<td>modifyLineBreaks:</td>
</tr>
<tr>
<td>environments:</td>
</tr>
<tr>
<td>BodyStartsOnOwnLine: 2</td>
</tr>
</tbody>
</table>

Upon running commands analogous to the above, we obtain Listings 311 and 312.
6. THE -M (MODIFYLINEBREAKS) SWITCH

Note that line breaks have been added as in Listings 307 and 308, but this time a comment symbol has been added before adding the line break; in both cases, trailing horizontal space has been stripped before doing so.

Let’s now change each of the 1 values in Listings 305 and 306 so that they are 3 and save them into env-mlb5.yaml and env-mlb6.yaml respectively (see Listings 313 and 314).

Upon running commands analogous to the above, we obtain Listings 315 and 316.

There are a couple of points to note:

• in Listing 319 a line break has been added at the point denoted by ♦ in Listing 304 on the previous page; no other line breaks have been changed and the \end{myenv} statement has not received indentation (as intended);

• in Listing 320 a line break has been added at the point denoted by ♣ in Listing 304 on the preceding page.

Let’s now change each of the 1 values in Listings 317 and 318 so that they are 2 and save them into env-mlb9.yaml and env-mlb10.yaml respectively (see Listings 321 and 322).
6. THE -M (MODIFYLINEBREAKS) SWITCH

Upon running commands analogous to the above, we obtain Listings 323 and 324.

**Listing 323: env-mlb.tex using Listing 321**

```latex
\begin{myenv}
body of myenv
\end{myenv}
```

Note that line breaks have been added as in Listings 319 and 320, but this time a comment symbol has been added before adding the line break; in both cases, trailing horizontal space has been stripped before doing so.

Let’s now change each of the 1 values in Listings 317 and 318 so that they are 3 and save them into `env-mlb11.yaml` and `env-mlb12.yaml` respectively (see Listings 325 and 326).

**Listing 325: env-mlb11.yaml**

```yaml
modifyLineBreaks:
environments:
  EndStartsOnOwnLine: 3
```

Upon running commands analogous to the above, we obtain Listings 327 and 328.

**Listing 327: env-mlb.tex using Listing 325**

```latex
\begin{myenv}
body of myenv
\end{myenv}
```

**Listing 328: env-mlb.tex using Listing 326**

```latex
\begin{myenv}
body of myenv
\end{myenv}
```

Note that line breaks have been added as in Listings 319 and 320, and that a blank line has been added after the line break.

### 6.6.3 poly-switches only add line breaks when necessary

If you ask `latexindent.pl` to add a line break (possibly with a comment) using a poly-switch value of 1 (or 2), it will only do so if necessary. For example, if you process the file in Listing 329 using any of the YAML files presented so far in this section, it will be left unchanged.

**Listing 329: env-mlb2.tex**

```latex
\begin{myenv}
body of myenv
\end{myenv}
```

In contrast, the output from processing the file in Listing 330 will vary depending on the poly-switches used; in Listing 331 you’ll see that the comment symbol after the `\begin{myenv}` has been moved to the next line, as `BodyStartsOnOwnLine` is set to 1. In Listing 332 you’ll see that the comment has been accounted for correctly because `BodyStartsOnOwnLine` has been set to 2, and the comment symbol has not been moved to its own line. You’re encouraged to experiment with Listing 330 and by setting the other poly-switches considered so far to 2 in turn.

**Listing 331: env-mlb3.tex using Listing 306 on page 79**

```latex
\begin{myenv}
\% body of myenv
\end{myenv}\%
```

**Listing 332: env-mlb3.tex using Listing 310 on page 79**

```latex
\begin{myenv}
\% body of myenv
\end{myenv}\%
```
The details of the discussion in this section have concerned global poly-switches in the environments field; each switch can also be specified on a per-name basis, which would take priority over the global values; with reference to Listing 303 on page 79, an example is shown for the equation* environment.

### 6.6.4 Removing line breaks (poly-switches set to −1)

Setting poly-switches to −1 tells \texttt{latexindent.pl} to remove line breaks of the \texttt{<part of the thing>}, if necessary. We will consider the example code given in Listing 333, noting in particular the positions of the line break highlighters, ♠, ♥, ♦ and ♣, together with the associated YAML files in Listings 334 to 337.

After running the commands

\begin{verbatim}
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb13.yaml
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb14.yaml
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb15.yaml
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb16.yaml
\end{verbatim}

we obtain the respective output in Listings 338 to 341.

Notice that in:

- Listing 338 the line break denoted by ♠ in Listing 333 has been removed;
- Listing 339 the line break denoted by ♥ in Listing 333 has been removed;

```yaml
Listing 334: env-mlb13.yaml
modifyLineBreaks:
environments:
  BeginStartsOnOwnLine: -1

Listing 335: env-mlb14.yaml
modifyLineBreaks:
environments:
  BodyStartsOnOwnLine: -1

Listing 336: env-mlb15.yaml
modifyLineBreaks:
environments:
  EndStartsOnOwnLine: -1

Listing 337: env-mlb16.yaml
modifyLineBreaks:
environments:
  EndFinishesWithLineBreak: -1
```
• Listing 340 the line break denoted by ♦ in Listing 333 has been removed;
• Listing 341 the line break denoted by ♣ in Listing 333 has been removed.

We examined each of these cases separately for clarity of explanation, but you can combine all of
the YAML settings in Listings 334 to 337 into one file; alternatively, you could tell latexindent.pl
to load them all by using the following command, for example

```
```

which gives the output in Listing 304 on page 79.

### 6.6.5 About trailing horizontal space

Recall that on page 20 we discussed the YAML field removeTrailingWhitespace, and that it has
two (binary) switches to determine if horizontal space should be removed beforeProcessing and
afterProcessing. The beforeProcessing is particularly relevant when considering the -m switch;
let’s consider the file shown in Listing 342, which highlights trailing spaces.

**Listing 342: env-mlb5.tex**

```
\begin{myenv}/body/\text{of/myenv/after/words}
```

**Listing 343: removeTWS-before.yaml**

```
modifyLineBreaks:
  preserveBlankLines: 0
```

The output from the following commands

```
cmh:~$ latexindent.pl -m env-mlb5.tex -l env-mlb13.yaml,env-mlb14.yaml,env-mlb15.yaml,env-mlb16.yaml,removeTWS-before.yaml
```

is shown, respectively, in Listings 344 and 345; note that the trailing horizontal white space has
been preserved (by default) in Listing 344, while in Listing 345, it has been removed using the
switch specified in Listing 343.

**Listing 344: env-mlb5.tex using Listings 338 to 341**

```
before_words\begin{myenv}body_of_myenv\text{of/myenv/after/words}
```

**Listing 345: env-mlb5.tex using Listings 338 to 341 and Listing 343**

```
before_words\begin{myenv}body_of_myenv\text{of/myenv/after/words}
```

### 6.6.6 poly-switch line break removal and blank lines

Now let’s consider the file in Listing 346, which contains blank lines.

**Listing 346: env-mlb6.tex**

```
\begin{myenv}/body_of_myenv/after/words
```

**Listing 347: UnpreserveBlankLines.yaml**

```
modifyLineBreaks:
  preserveBlankLines: 0
```

Upon running the following commands
we receive the respective outputs in Listings 348 and 349. In Listing 348 we see that the multiple blank lines have each been condensed into one blank line, but that blank lines have not been removed by the poly-switches – this is because, by default, preserveBlankLines is set to 1. By contrast, in Listing 349, we have allowed the poly-switches to remove blank lines because, in Listing 347, we have set preserveBlankLines to 0.

We can explore this further using the blank-line poly-switch value of 3; let’s use the file given in Listing 350.

We notice that in:
• Listing 351 that \end{one} has added a blank line, because of the value of EndFinishesWithLineBreak in Listing 326 on page 81, and even though the line break ahead of \begin{two} should have been removed (because of BeginStartsOnOwnLine in Listing 334 on page 82), the blank line has been preserved by default;
• Listing 352, by contrast, has had the additional line-break removed, because of the settings in Listing 347.

6.7 Poly-switches for other code blocks
Rather than repeat the examples shown for the environment code blocks (in Section 6.6 on page 79), we choose to detail the poly-switches for all other code blocks in Table 2; note that each and every one of these poly-switches is off by default, i.e., set to 0. Note also that, by design, line breaks involving
verbatim, filecontents and ‘comment-marked’ code blocks (Listing 47 on page 26) can not be modified using latexindent.pl.

<table>
<thead>
<tr>
<th>Code block</th>
<th>Sample</th>
<th>Poly-switch mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>environment</td>
<td>before words♠  \begin{myenv}♡ body of myenv♢ \end{myenv}♣ after words</td>
<td>♠ BeginStartsOnOwnLine ♡ BodyStartsOnOwnLine ♢ EndStartsOnOwnLine ♣ EndFinishesWithLineBreak</td>
</tr>
<tr>
<td>ifelsefi</td>
<td>before words♠  \if...♥ body of if/or statement▲ \or▼ body of if/or statement★ else□ body of else statement◇ \fi♣ after words</td>
<td>♠ IfStartsOnOwnLine ♡ BodyStartsOnOwnLine ♢ OrStartsOnOwnLine ♣ OrFinishesWithLineBreak ★ ElseStartsOnOwnLine □ ElseFinishesWithLineBreak ♦ FiStartsOnOwnLine ♣ FiFinishesWithLineBreak</td>
</tr>
<tr>
<td>optionalArguments</td>
<td>...♠  [♥ body of opt arg♦ ]♣ ...</td>
<td>♠ LSqBStartsOnOwnLine(^9) ♡ OptArgBodyStartsOnOwnLine ♢ RSqBStartsOnOwnLine ♣ RSqBFinishesWithLineBreak</td>
</tr>
<tr>
<td>mandatoryArguments</td>
<td>...♠  {♥ body of mand arg♦ }♣ ...</td>
<td>♠ LCuBStartsOnOwnLine(^10) ♡ MandArgBodyStartsOnOwnLine ♢ RCuBStartsOnOwnLine ♣ RCuBFinishesWithLineBreak</td>
</tr>
<tr>
<td>commands</td>
<td>before words♠  \mycommand♡  (\text{arguments})</td>
<td>♠ CommandStartsOnOwnLine ♡ CommandNameFinishesWithLineBreak</td>
</tr>
<tr>
<td>namedGroupingBraces Brackets</td>
<td>before words♠  myname•  (\text{braces/brackets})</td>
<td>♠ NameStartsOnOwnLine ♡ NameFinishesWithLineBreak</td>
</tr>
<tr>
<td>keyEqualsValuesBraces Brackets</td>
<td>before words♠  \key⋆=•  (\text{braces/brackets})</td>
<td>♠ KeyStartsOnOwnLine ♡ EqualsStartsOnOwnLine ♢ EqualsFinishesWithLineBreak</td>
</tr>
<tr>
<td>items</td>
<td>before words♠  \item♡   (\text{items})</td>
<td>♠ ItemStartsOnOwnLine ♡ ItemFinishesWithLineBreak</td>
</tr>
<tr>
<td>specialBeginEnd</td>
<td>before words♠  [♥ body of special/middle★ \middle□ body of special/middle ◊ ]♣ after words</td>
<td>♠ SpecialBeginStartsOnOwnLine ♡ SpecialBodyStartsOnOwnLine ♢ SpecialMiddleStartsOnOwnLine □ SpecialMiddleFinishesWithLineBreak ◊ SpecialEndStartsOnOwnLine ♣ SpecialEndFinishesWithLineBreak</td>
</tr>
</tbody>
</table>

\(^9\)LSqB stands for Left Square Bracket  

\(^10\)LCuB stands for Left Curly Brace
6.8 Partnering BodyStartsOnOwnLine with argument-based poly-switches

Some poly-switches need to be partnered together, in particular, when line breaks involving the first argument of a code block need to be accounted for using both BodyStartsOnOwnLine (or its equivalent, see Table 2 on the preceding page) and LCuBStartsOnOwnLine for mandatory arguments, and LSqBStartsOnOwnLine for optional arguments.

Let's begin with the code in Listing 353 and the YAML settings in Listing 355; with reference to Table 2 on the previous page, the key CommandNameFinishesWithLineBreak is an alias for BodyStartsOnOwnLine.

Listing 353: mycommand1.tex
```
\mycommand
{ 
  mand arg text 
  mand arg text }
{ 
  mand arg text 
  mand arg text }
```

Upon running the command

```
cmh:~$ latexindent.pl -m -l=mycom-mlb1.yaml mycommand1.tex
```

we obtain Listing 354; note that the second mandatory argument beginning brace { has had its leading line break removed, but that the first brace has not.

Listing 354: mycommand1.tex using Listing 355
```
\mycommand
{ 
  mand arg text 
  mand arg text }
{ 
  mand arg text 
  mand arg text }
```

Listing 355: mycom-mlb1.yaml
```
modifyLineBreaks:
  commands:
    CommandNameFinishesWithLineBreak: 0
    mandatoryArguments:
      LCuBStartsOnOwnLine: -1
```

Now let's change the YAML file so that it is as in Listing 357; upon running the analogous command to that given above, we obtain Listing 356; both beginning braces { have had their leading line breaks removed.

Listing 356: mycommand1.tex using Listing 357
```
\mycommand{
  mand arg text 
  mand arg text}{
  mand arg text 
  mand arg text}
```

Listing 357: mycom-mlb2.yaml
```
modifyLineBreaks:
  commands:
    CommandNameFinishesWithLineBreak: -1
    mandatoryArguments:
      LCuBStartsOnOwnLine: -1
```

Now let's change the YAML file so that it is as in Listing 359; upon running the analogous command to that given above, we obtain Listing 358.

Listing 358: mycommand1.tex using Listing 359
```
\mycommand{
  mand arg text 
  mand arg text}
{ 
  mand arg text 
  mand arg text}
```

Listing 359: mycom-mlb3.yaml
```
modifyLineBreaks:
  commands:
    CommandNameFinishesWithLineBreak: -1
    mandatoryArguments:
      LCuBStartsOnOwnLine: 1
```
6.9 Conflicting poly-switches: sequential code blocks

It is very easy to have conflicting poly-switches; if we use the example from Listing 353 on the preceding page, and consider the YAML settings given in Listing 361. The output from running

```bash
latexindent.pl -m -l=mycom-mlb4.yaml mycommand1.tex
```

is given in Listing 361.

```
\mycommand
{  
  mand arg text
  mand arg text}{
  mand arg text
  mand arg text}
```

Studying Listing 361, we see that the two poly-switches are at opposition with one another:

- on the one hand, `LCuBStartsOnOwnLine` should not start on its own line (as poly-switch is set to -1);
- on the other hand, `RCuBFinishesWithLineBreak` should finish with a line break.

So, which should win the conflict? As demonstrated in Listing 360, it is clear that `LCuBStartsOnOwnLine` won this conflict, and the reason is that the second argument was processed after the first – in general, the most recently-processed code block and associated poly-switch takes priority.

We can explore this further by considering the YAML settings in Listing 363; upon running the command

```bash
latexindent.pl -m -l=mycom-mlb5.yaml mycommand1.tex
```

we obtain the output given in Listing 362.

```
\mycommand
{  
  mand arg text
  mand arg text}{
  mand arg text
  mand arg text}
```

As previously, the most-recently-processed code block takes priority – as before, the second (i.e, last) argument. Exploring this further, we consider the YAML settings in Listing 365, which give associated output in Listing 364.

```
\mycommand
{  
  mand arg text
  mand arg text}%
{  
  mand arg text
  mand arg text}
```

Note that a `%` has been added to the trailing first `}; this is because:
• while processing the first argument, the trailing line break has been removed (RCuBFinishesWithLineBreak set to −1);
• while processing the second argument, latexindent.pl finds that it does not begin on its own line, and so because LCuBStartsOnOwnLine is set to 2, it adds a comment, followed by a line break.

6.10 Conflicting poly-switches: nested code blocks
Now let’s consider an example when nested code blocks have conflicting poly-switches; we’ll use the code in Listing 366, noting that it contains nested environments.

<table>
<thead>
<tr>
<th>Listing 366: nested-env.tex</th>
</tr>
</thead>
</table>
\begin{one}
  one text
\begin{two}
  two text
\end{two}
\end{one}

Let’s use the YAML settings given in Listing 368, which upon running the command

```
cmh:~$ latexindent.pl -m -l=nested-env-mlb1.yaml nested-env.tex
```

gives the output in Listing 367.

<table>
<thead>
<tr>
<th>Listing 367: nested-env.tex using Listing 368</th>
</tr>
</thead>
</table>
\begin{one}
  one text
\begin{two}
  two text\end{two}\end{one}

In Listing 367, let’s first of all note that both environments have received the appropriate (default) indentation; secondly, note that the poly-switch EndStartsOnOwnLine appears to have won the conflict, as \end{one} has had its leading line break removed.

To understand it, let’s talk about the three basic phases of latexindent.pl:

1. Phase 1: packing, in which code blocks are replaced with unique ids, working from the inside to the outside, and then sequentially – for example, in Listing 366, the two environment is found before the one environment; if the -m switch is active, then during this phase:
   • line breaks at the beginning of the body can be added (if BodyStartsOnOwnLine is 1 or 2) or removed (if BodyStartsOnOwnLine is −1);
   • line breaks at the end of the body can be added (if EndStartsOnOwnLine is 1 or 2) or removed (if EndStartsOnOwnLine is −1);
   • line breaks after the end statement can be added (if EndFinishesWithLineBreak is 1 or 2).
2. Phase 2: indentation, in which white space is added to the begin, body, and end statements;
3. Phase 3: unpacking, in which unique ids are replaced by their indented code blocks; if the -m switch is active, then during this phase,
   • line breaks before begin statements can be added or removed (depending upon BeginStartsOnOwnLine);
   • line breaks after end statements can be removed but NOT added (see EndFinishesWithLineBreak).

With reference to Listing 367, this means that during Phase 1:
• the two environment is found first, and the line break ahead of the \end{two} statement is removed because EndStartsOnOwnLine is set to −1. Importantly, because, at this stage, \end{two} does finish with a line break, EndFinishesWithLineBreak causes no action.

• next, the one environment is found; the line break ahead of \end{one} is removed because EndStartsOnOwnLine is set to −1.

The indentation is done in Phase 2; in Phase 3 there is no option to add a line break after the end statements. We can justify this by remembering that during Phase 3, the one environment will be found and processed first, followed by the two environment. If the two environment were to add a line break after the \end{two} statement, then latexindent.pl would have no way of knowing how much indentation to add to the subsequent text (in this case, \end{one}).

We can explore this further using the poly-switches in Listing 370; upon running the command

```bash
cmh:~$ latexindent.pl -m -l=nested-env-mlb2.yaml nested-env.tex
```

we obtain the output given in Listing 369.

The main limitation is to do with the alignment routine discussed on page 21; for example, consider the file given in Listing 371.

```latex
\begin{one}
one text
\begin{two}
two text
\end{two}\end{one}
```

The default output is given in Listing 372, and it is clear that the alignment routine has not worked as hoped, but it is expected.

7 Conclusions and known limitations

There are a number of known limitations of the script, and almost certainly quite a few that are unknown!

The main limitation is to do with the alignment routine discussed on page 21; for example, consider the file given in Listing 371.

```latex
\matrix (A)\{ 
c01 & c02 & c03 & c0q \\ 
c_{-11} & c12 & \ldots & c1q \\ 
};
```

The default output is given in Listing 372, and it is clear that the alignment routine has not worked as hoped, but it is expected.
The reason for the problem is that when latexindent.pl stores its code blocks (see Table 1 on page 32) it uses replacement tokens. The alignment routine is using the length of the replacement token in its measuring – I hope to be able to address this in the future.

There are other limitations to do with the multicolumn alignment routine (see Listing 38 on page 23); in particular, when working with codeblocks in which multicolumn commands overlap, the algorithm can fail.

Another limitation is to do with efficiency, particularly when the -m switch is active, as this adds many checks and processes. The current implementation relies upon finding and storing every code block (see the discussion on page 88); it is hoped that, in a future version, only nested code blocks will need to be stored in the ‘packing’ phase, and that this will improve the efficiency of the script.

You can run latexindent on .sty, .cls and any file types that you specify in fileExtensionPreference (see Listing 14 on page 17); if you find a case in which the script struggles, please feel free to report it at [7], and in the meantime, consider using a noIndentBlock (see page 20).

I hope that this script is useful to some; if you find an example where the script does not behave as you think it should, the best way to contact me is to report an issue on [7]; otherwise, feel free to find me on the http://tex.stackexchange.com/users/6621/cmhughes.

8 References

8.1 External links

8. Text::Wrap Perl module. URL: http://perldoc.perl.org/Text/Wrap.html (visited on 05/01/2017).

8.2 Contributors

A Required Perl modules

If you intend to use `latexindent.pl` and not one of the supplied standalone executable files, then you will need a few standard Perl modules – if you can run the minimum code in Listing 373 (`perl helloworld.pl`) then you will be able to run `latexindent.pl`, otherwise you may need to install the missing modules – see appendices A.1 and A.2.

**Listing 373: helloworld.pl**

```perl
#!/usr/bin/perl

use strict;
use warnings;
use utf8;
use PerlIO::encoding;
use Unicode::GCString;
use open ':std', ':encoding(UTF-8)';
use Text::Wrap;
use Text::Tabs;
use FindBin;
use YAML::Tiny;
use File::Copy;
use File::Basename;
use File::HomeDir;
use Getopt::Long;
use Data::Dumper;
use List::Util qw(max);
use Log::Log4perl qw(get_logger :levels);

print "hello/world"
exit;
```

A.1 Module installer script

`latexindent.pl` ships with a helper script that will install any missing perl modules on your system; if you run

```
cmh:~$ perl latexindent-module-installer.pl
```

or

```
C:\Users\cmh>perl latexindent-module-installer.pl
```

then, once you have answered Y, the appropriate modules will be installed onto your distribution.

A.2 Manually installed modules

Manually installing the modules given in Listing 373 will vary depending on your operating system and Perl distribution. For example, Ubuntu users might visit the software center, or else run

```
cmh:~$ sudo perl -MCPAN -e 'install/File::HomeDir''
```
Linux users may be interested in exploring Perlbrew \cite{13}; possible installation and setup options follow for Ubuntu (other distributions will need slightly different commands).

```
cmh:~$ sudo apt-get install perlbrew
cmh:~$ perlbrew install perl-5.22.1
cmh:~$ perlbrew switch perl-5.22.1
cmh:~$ sudo apt-get install curl
cmh:~$ curl -L http://cpanmin.us | perl - App::cpanminus
cmh:~$ cpanm YAML::Tiny
cmh:~$ cpanm File::HomeDir
cmh:~$ cpanm Unicode::GCString
cmh:~$ cpanm Log::Log4perl
cmh:~$ cpanm Log::Dispatch
```

Strawberry Perl users on Windows might use CPAN client. All of the modules are readily available on CPAN \cite{4}.

indent.log will contain details of the location of the Perl modules on your system. latexindent.exe is a standalone executable for Windows (and therefore does not require a Perl distribution) and caches copies of the Perl modules onto your system; if you wish to see where they are cached, use the trace option, e.g.

```
C:\Users\cmh>latexindent.exe -t myfile.tex
```

**B Updating the path variable**

latexindent.pl has a few scripts (available at \cite{7}) that can update the path variables. Thank you to \cite{8} for this feature. If you're on a Linux or Mac machine, then you'll want CMakeLists.txt from \cite{7}.

**B.1 Add to path for Linux**

To add latexindent.pl to the path for Linux, follow these steps:

1. download latexindent.pl and its associated modules, defaultSettings.yaml, to your chosen directory from \cite{7};
2. within your directory, create a directory called path-helper-files and download CMakeLists.txt and cmake_uninstall.cmake.in from \cite{7}/path-helper-files to this directory;
3. run
   ```
cmh:~$ ls /usr/local/bin
```
to see what is currently in there;
4. run the following commands
   ```
cmh:~$ sudo apt-get install cmake
cmh:~$ sudo apt-get update && sudo apt-get install build-essential
cmh:~$ mkdir build && cd build
cmh:~$ cmake ../path-helper-files
```
5. run
   ```
cmh:~$ ls /usr/local/bin
```

C. LOGFILEPREFERENCES

again to check that \texttt{latexindent.pl}, its modules and \texttt{defaultSettings.yaml} have been added.

To remove the files, run

cmh:~$ sudo make uninstall

B.2 Add to path for Windows

To add \texttt{latexindent.exe} to the path for Windows, follow these steps:

1. download \texttt{latexindent.exe}, \texttt{defaultSettings.yaml}, \texttt{add-to-path.bat} from [7] to your chosen directory;
2. open a command prompt and run the following command to see what is currently in your \texttt{%path%} variable;

   \[
   C:\Users\cmh>echo %path%
   \]

3. right click on \texttt{add-to-path.bat} and Run as administrator;
4. log out, and log back in;
5. open a command prompt and run

   \[
   C:\Users\cmh>echo %path%
   \]

   to check that the appropriate directory has been added to your \texttt{%path%}.

To remove the directory from your \texttt{%path%}, run \texttt{remove-from-path.bat} as administrator.

C logFilePreferences

Listing 15 on page 19 describes the options for customising the information given to the log file, and we provide a few demonstrations here. Let’s say that we start with the code given in Listing 374, and the settings specified in Listing 375.

\begin{lstlisting}[language=TeX]
\begin{myenv}
  body of myenv
\end{myenv}
\end{lstlisting}

\begin{lstlisting}[language=yaml]
logFilePreferences:
  showDecorationStartCodeBlockTrace: "+++++
  showDecorationFinishCodeBlockTrace: "-----"
\end{lstlisting}

If we run the following command (noting that -t is active)

cmh:~$ latexindent.pl -t -l=logfile-prefs1.yaml simple.tex

then on inspection of indent.log we will find the snippet given in Listing 376.
D. DIFFERENCES FROM VERSION 2.2 TO 3.0

There are a few (small) changes to the interface when comparing Version 2.2 to Version 3.0. Explicitly, in previous versions you might have run, for example,

```bash
cmh:~$ latexindent.pl -o myfile.tex outputfile.tex
```

whereas in Version 3.0 you would run any of the following, for example,

```bash
cmh:~$ latexindent.pl -o=myfile.tex outputfile.tex
cmh:~$ latexindent.pl -o outputfile.tex myfile.tex
cmh:~$ latexindent.pl myfile.tex -o outputfile.tex
cmh:~$ latexindent.pl myfile.tex -o=outputfile.tex
cmh:~$ latexindent.pl myfile.tex -o outputfile outputfile.tex
```

noting that the `output` file is given next to the `-o` switch.

The fields given in Listing 377 are obsolete from Version 3.0 onwards.

```yml
alwaysLookforSplitBrackets
alwaysLookforSplitBrackets
checkunmatched
checkunmatchedELSE
checkunmatchedbracket
constructIfElseFi
```

There is a slight difference when specifying indentation after headings; specifically, we now write `indentAfterThisHeading` instead of `indent`. See Listings 378 and 379.

```yml
indentAfterHeadings:
  part:
    indent: 0
    level: 1

indentAfterHeadings:
  part:
    indentAfterThisHeading: 0
    level: 1
```
To specify `noAdditionalIndent` for display-math environments in Version 2.2, you would write YAML as in Listing 380; as of Version 3.0, you would write YAML as in Listing 381 or, if you're using `-m` switch, Listing 382.

**Listing 380: `noAdditionalIndent` in Version 2.2**

```yaml
noAdditionalIndent:
  \[
  \]: 0
  \]: 0
```

**Listing 381: `noAdditionalIndent` for displayMath in Version 3.0**

```yaml
specialBeginEnd:
  displayMath:
    begin: '```'
    end: '```'
lookForThis: 0
```

**Listing 382: `noAdditionalIndent` for displayMath in Version 3.0**

```yaml
noAdditionalIndent:
  displayMath: 1
```

---

End