

*» The Monty Pythons, were they \TeX users,
could have written the `chickenize` macro.«*
Paul Isambert

chickenize

Arno Trautmann
arno.trautmann@gmx.de

October 22, 2011

This is the package `chickenize`. It allows you to substitute or change the contents of a \LuaTeX document,¹ but is actually just for fun. Please *never* use any of the functionality of this package for a production document. The following table informs you shortly about some of your possibilities and provides links to the Lua functions. The \TeX interface is presented [below](#).

function/command	effect
chickenize	replaces every word with “chicken”
colorstretch	shows grey boxes that depict the badness and font expansion of each line
leetspeak	translates the (latin-based) input into 1337 5p34k
randomucl	changes randomly between uppercase and lowercase
randomfont	changes the font randomly between every letter
randomchar	randomizes the whole input
randomcolor	prints every letter in a random color
rainbowcolor	changes the color of letters slowly according to a rainbow
uppercasecolor	makes every uppercase letter colored

If you have any suggestions or comments, just drop me a mail, I’ll be happy to get any response!

¹The code is based on pure \LuaTeX features, so don't even try to use it with any other \TeX flavour. The package is tested under \LuaTeX , and should be working fine with plain \LuaTeX . If you tried it with Con \TeX t, please share your experience!

Contents

I	User Documentation	3
1	How It Works	3
2	Commands – How You Can Use It	3
2.1	T_EX Commands – Document Wide	3
2.2	How to Deactivate It	4
2.3	\text-Versions	4
2.4	Lua functions	5
3	Options – How to Adjust It	5
II	Implementation	7
4	T_EX file	7
5	L^AT_EX package	10
5.1	Definition of User-Level Macros	10
6	Lua Module	11
6.1	chickenize	11
6.2	leetspeak	13
6.3	pancakenize	14
6.4	randomfonts	14
6.5	randomucl	15
6.6	randomchars	15
6.7	randomcolor and rainbowcolor	15
6.8	rickroll	17
6.9	uppercasecolor	17
6.10	colorstretch	18
6.11	draw a chicken	21
7	Known Bugs	23
8	To Dos	23

Part I

User Documentation

1 How It Works

We make use of Lua \TeX s callbacks, especially the `pre_linebreak_filter` and the `post_linebreak_filter`. Hooking a function into these, we can nearly arbitrarily change the contents of the document. If the changes should be on the input-side (replacing with `chicken`), one can use the `pre_linebreak_filter`. However, changes like inserting color are best made after the linebreak is finalized, so `post_linebreak_filter` is used for such things.

All functions traverse the node list of a paragraph and manipulate the nodes' properties (like `.font` or `.char`) or insert nodes (like color push/pop nodes) and return this changed node list.

2 Commands – How You Can Use It

There are several ways to make use of this package – you can either stay on the \TeX side or use the Lua functions directly. In fact, the \TeX macros are simple wrappers around the functions.

2.1 \TeX Commands – Document Wide

You have a number of commands at your hand, each of which does some manipulation of the input or output. In fact, the code is easy and straightforward, but be careful, especially when combining things. Apply features step by step so your brain won't be damaged ...

The effect of the commands can be influenced, not with arguments, but only via the `\chickenizesetup` described [below](#).

`\chickenize` Replaces every word of the input with the word “chicken”. Maybe sometime the replaced word can be changed, but up to now, it's only chicken. To be a bit less static, about every 10th chicken is uppercase. However, the beginning of a sentence is not recognized automatically.²

`\uppercasecolor` Makes every uppercase character in the input colored. At the moment, the color is randomized over the full rgb scale, but that will be adjustable once options are well implemented.

`\randomuclc` Changes every character of the input into its uppercase or lowercase variant. Well, guess what the “random” means ...

²If you have a nice implementation idea, I'd love to include this!

`\randomfonts` Changes the font randomly for every character. If no parameters are given, all fonts that have been loaded are used, especially including math fonts.

`\randomcolor` Does what it's name says.

`\rainbowcolor` Instead of random colors, this command causes the text color to change slowly according to the colors of a rainbow. Do not mix this with `randomcolor`, as that doesn't make any sense.

`\pancakenize` This is a dummy so far, as I have no idea what it should do. If you have suggestions, please tell me.

`\nyanize` A synonym for `rainbowcolor`.

`\leetspeak` Translates the input into 1337 speak. If you don't understand that, lern it, n00b.

`\colorstretch` Inspired by Paul Isambert's code, this command prints boxes instead of lines. The greyness of the first (left-hand) box corresponds to the badness of the line, i. e. it is a measure for how much the space between words has been extended to get proper paragraph justification. The second box on the right-hand side shows the amount of stretching/shrinking when font expansion is used. Together the box greyness give you information about how well the overall greyness of the typeset page is.

This functionality is actually the only really usefull implementation of this package ...

2.2 How to Deactivate It

Every command has a `\un-`version that deactivates it's functionality. So once you used `\chickenize`, it will chickenize the whole document up to `\unchickenize`. However, the paragraph in which `\unchickenize` appears, will *not* be chickenized. The same is true for all other manipulations. Take care that you don't `\un-`anything bevor activating it, as this will result in an error.³

If you want to manipulate only a part of a paragraph, you have use the `\text-`version of the function, see below. However, feel free to set and unset every function at will at any place in your document.

2.3 \text-Versions

The functions of this package might be much more useful if applied only to a short sequence of words or single words instead of the whole document or paragraph. Therefore,

³Which is so far not catchable due to missing functionality in `luatexbase`.

most of the above-mentioned commands have⁴ a `\text-version` that takes an argument. `\textrandomcolor{foo}` results in a colored `foo` while the rest of the document keeps its color. However, to achieve this effect, still the whole node list has to be traversed, so it may slow down your document, even if you use `\textrandomcolor` only once. Fortunately, the effect is very small and mostly negligible.⁵

Please don't fool around by mixing a `\text-version` with the non-`\text-version`. If you feel like and are not please with the result, it is up to *you* to provide a stable and working solution.

2.4 Lua functions

As all features are implemented on the Lua side, you can use these functions on their own. If you do so, please consult the corresponding subsections in the [implementation](#) part, because there are some variables that can be adapted to your need.

You can use the following code inside a `\directlua` statement or in a `luacode` environment (or the corresponding thing in your format):

```
luatexbase.add_to_callback("pre_linebreak_filter",chickenize,"chickenize")
```

Replace `pre` by `post` to register into the post linebreak filter. The second argument gives the function name; find a list of available functions below. You can give a label as you like in the third argument, and the last argument gives the order in which the functions in the callback are used. If you have no fancy stuff going on, you can safely use 1.

3 Options – How to Adjust It

There are several ways to change the behaviour of `chickenize` and its macros. Most of the options are Lua variables and can be set using `\chickenizesetup`. But be *careful*! The argument of `\chickenizesetup` is parsed directly to Lua, therefore you are *not* using a comma-separated key-value list, but uncorrelated Lua commands. The argument must have the syntax `{randomfontslower = 1 randomfontsupper = 0}` instead of `{randomfontslower = 1, randomfontsupper = 0}`. Alright?

However, `\chickenizesetup` is a macro on the \TeX side meaning that you can use *only* `%` as comment string. If you use `--`, all of the argument will be ignored as \TeX does not pass an eol to `\directlua`. If you don't understand that, just ignore it and go on as usual.

The following list tries to keep kind of track to the options and variables. There is no guarantee for this list, and if you find something that is missing or doesn't work as described here, please inform me!

⁴If they don't have, I did miss that, sorry. Please inform me about such cases.

⁵On a 500 pages text-only \LaTeX document the dilation is on the order of 10% with `\textrandomcolor`, but other manipulations can take much more time. However, you are not supposed to make such long documents with `chickenize`!

`randomfontslower, randomfontsupper = <int>` These two integer variables determine the span of fonts used for the font randomization. Just play with them a bit to find out what they are doing.

`chickenstring = <table>` The string that is printed when using `\chickenize`. In fact, `chickenstring` is a table which allows for some more random action. To specify the default string, say `chickenstring[1] = 'chicken'`. For more than one animal, just step the index: `chickenstring[2] = 'rabbit'`. All existing table entries will be used randomly. Remember that we are dealing with Lua strings here, so use `' '` to mark them. (`" "` can cause problems with `babel`.)

`chickenizefraction = <float> 1` Gives the fraction of words that get replaced by the `chickenstring`. The default means that every word is substituted. However, with a value of, say, `0.0001`, only one word in ten thousand will be `chickenstring`. `chickenizefraction` must be specified *after* `\begin{document}`. No idea, why ...

`colorstretchnumbers = <true>` If true, the amount of stretching or shrinking of each line is printed into the margin as a green, red or black number.

`leettable = <table>` From this table, the substitution for 1337 is taken. If you want to add or change an entry, you have to provide the unicode numbers of the characters, e.g. `leettable[101] = 50` replaces every `e` (101) with the number `3` (50).

`uclcratio = <float> 0.5` Gives the fraction of uppercases to lowercases in the `\randomucl` mode. A higher number (up to 1) gives more uppercase letters. Guess what a lower number does.

`randomcolor_grey = <bool> false` For a printer-friendly version, this offers a grey scale instead of an `rgb` value for `\randomcolor`.

`rainbow_step = <float> 0.005` This indicates the relative change of color using the rainbow functionality. A value of 1 changes the color in one step from red to yellow, while a value of `0.005` takes 200 letters for this change. Useful values are below `0.05`, but it depends on the amount of text. The longer the text and the lower the step, the nicer your rainbow will be.

`Rgb_lower, rGb_upper = <int>` To specify the color space that is used for `\randomcolor`, you can specify six values, the upper and lower value for each color. The uppercase letter in the variable denotes the color, so `rGb_upper` gives the upper value for green etc. Possible values are between 1 and 254. If you enter anything outside this, your pdf will become invalid and break. For grey scale, use `grey_lower` and `grey_upper`, with values between 0 (black) and 1000 (white), included. Default is 0 to 900 to prevent white letters.

`keeptext = <bool> false` This is for the `\colorstretch` command. If set to true, the text of your document will be kept. This way, it is easier to identify bad lines and the reason for the badness.

`colorexpan = <bool> true` If true, two bars are shown of which the second one denotes the font expansion. Only usefull if font expansion is used. (You *do* use font expansion, do you?)

Part II

Implementation

4 TeX file

```
1 \input{luatexbase.sty}
2 % read the Lua code first
3 \directlua{dofile("chickenize.lua")}
4 % then define the global macros. These affect the whole document and will stay active until the f
5 \def\chickenize{
6   \directlua{luatexbase.add_to_callback("pre_linebreak_filter",chickenize,"chickenize")}
7   luatexbase.add_to_callback("start_page_number",
8     function() texio.write("[..status.total_pages) end ,"cstartpage")
9     luatexbase.add_to_callback("stop_page_number",
10      function() texio.write(" chickens]") end,"cstoppage")}
11
12   luatexbase.add_to_callback("stop_run",nicetext,"a nice text")
13 }
14 }
15 \def\unchickenize{
16   \directlua{luatexbase.remove_from_callback("pre_linebreak_filter","chickenize")}
17   luatexbase.remove_from_callback("start_page_number","cstartpage")
18   luatexbase.remove_from_callback("stop_page_number","cstoppage")}}
19
20 \def\colorstretch{
21   \directlua{luatexbase.add_to_callback("post_linebreak_filter",colorstretch,"stretch_expansion")}
22 }
23 \def\uncolorstretch{
24   \directlua{luatexbase.remove_from_callback("post_linebreak_filter","stretch_expansion")}}
25
26 \def\leetspeak{
27   \directlua{luatexbase.add_to_callback("post_linebreak_filter",leet,"1337")}}
28 \def\unleetspeak{
29   \directlua{luatexbase.remove_from_callback("post_linebreak_filter","1337")}}
30
31 \def\rainbowcolor{
```

```

31 \directlua{luatexbase.add_to_callback("post_linebreak_filter",randomcolor,"rainbowcolor")
32         rainbowcolor = true}}
33 \def\unrainbowcolor{
34 \directlua{luatexbase.remove_from_callback("post_linebreak_filter","rainbowcolor")
35         rainbowcolor = false}}
36 \let\nyanize\rainbowcolor
37 \let\unnyanize\unrainbowcolor
38
39 \def\pancakenize{
40 \directlua{}}
41 \def\unpancakenize{
42 \directlua{}}
43
44 \def\coffeestainize{
45 \directlua{}}
46 \def\uncoffeestainize{
47 \directlua{}}
48
49 \def\randomcolor{
50 \directlua{luatexbase.add_to_callback("post_linebreak_filter",randomcolor,"randomcolor")}}
51 \def\unrandomcolor{
52 \directlua{luatexbase.remove_from_callback("post_linebreak_filter","randomcolor")}}
53
54 \def\randomfonts{
55 \directlua{luatexbase.add_to_callback("post_linebreak_filter",randomfonts,"randomfonts")}}
56 \def\unrandomfonts{
57 \directlua{luatexbase.remove_from_callback("post_linebreak_filter","randomfonts")}}
58
59 \def\randomuclc{
60 \directlua{luatexbase.add_to_callback("pre_linebreak_filter",randomuclc,"randomuclc")}}
61 \def\unrandomuclc{
62 \directlua{luatexbase.remove_from_callback("pre_linebreak_filter","randomuclc")}}
63
64 \def\uppercasecolor{
65 \directlua{luatexbase.add_to_callback("post_linebreak_filter",uppercasecolor,"uppercasecolor")}}
66 \def\unuppercasecolor{
67 \directlua{luatexbase.remove_from_callback("post_linebreak_filter","uppercasecolor")}}

```

Now the setup for the \text-versions. We utilize LuaTeX's attributes to mark all nodes that should be manipulated. The macros should be \long to allow arbitrary input.

```

68 \newluaTeXattribute\leetattr
69 \newluaTeXattribute\randcolorattr
70 \newluaTeXattribute\randfontsattrib
71 \newluaTeXattribute\randuclcattrib
72
73 \long\def\textleetspeak#1%

```



```

74 {\setluatexattribute\leetattr{42}#1\unsetluatexattribute\leetattr}
75 \long\def\textrandomcolor#1%
76 {\setluatexattribute\randcolorattr{42}#1\unsetluatexattribute\randcolorattr}
77 \long\def\textrandomfonts#1%
78 {\setluatexattribute\randfontsattr{42}#1\unsetluatexattribute\randfontsattr}
79 \long\def\textrandomfonts#1%
80 {\setluatexattribute\randfontsattr{42}#1\unsetluatexattribute\randfontsattr}
81 \long\def\textrandomuclc#1%
82 {\setluatexattribute\randuclcatr{42}#1\unsetluatexattribute\randuclcatr}

```

Finally, a macro to control the setup. So far, it's only a wrapper that allows T_EX-style comments to make the user feel more at home.

```

83 \def\chickenizesetup#1{\directlua{#1}}
84 \long\def\luadraw#1#2{%
85   \vbox to #1bp{%
86     \vfil
87     \luatexlatalua{pdf_print("q") #2 pdf_print("Q")}%
88   }%
89 }
90 \long\def\drawchicken{
91 \luadraw{90}{
92 kopf = {200,50} % Kopfmitte
93 kopf_rad = 20
94
95 d = {215,35} % Halsansatz
96 e = {230,10} %
97
98 korper = {260,-10}
99 korper_rad = 40
100
101 bein1 = {260,-50}
102 bein2 = {250,-70}
103 bein3 = {235,-70}
104
105 bein21 = {270,-50}
106 bein22 = {260,-75}
107 bein23 = {245,-75}
108
109 schnabel_oben = {185,55}
110 schnabel_vorne = {165,45}
111 schnabel_unten = {185,35}
112
113 flugel_vorne = {260,-10}
114 flugel_unten = {280,-40}
115 flugel_hinten = {275,-15}
116

```

```

117 sloppycircle(kopf,kopf_rad)
118 sloppyline(d,e)
119 sloppycircle(korper,korper_rad)
120 sloppyline(bein11,bein12) sloppyline(bein12,bein13)
121 sloppyline(bein21,bein22) sloppyline(bein22,bein23)
122 sloppyline(schnabel_vorne,schnabel_oben) sloppyline(schnabel_vorne,schnabel_unten)
123 sloppyline(flugel_vorne,flugel_unten) sloppyline(flugel_hinten,flugel_unten)
124
125 }
126 }

```

5 L^AT_EX package

I have decided to keep the L^AT_EX-part of this package as small as possible. So far, it does ... nothing usefull, but it provides a `chickenize.sty` that loads `chickenize.tex`. Some code might be implemented to manipulate figures for full chickenization. However, I will *not* load any packages at this place, as loading of `expl3` or `TikZ` or whatever takes too much time for such a tiny package like this one. If you want to use anything of the features presented here, you have to load the packages on your own. Maybe this will change.

```

127 \input{chickenize}

```

5.1 Definition of User-Level Macros

```

128 %% We want to "chickenize" figures, too. So ...
129 \iffalse
130 \DeclareDocumentCommand\includegraphics{0}{m}{
131     \fbox{Chicken} %% actually, I'd love to draw a mp graph showing a chicken ...
132 }
133 %%% specials: the balmerpeak. A tribute to http://xkcd.com/323/.
134 %% So far, you have to load pgfplots yourself.
135 %% As it is a mighty package, I don't want the user to force loading it.
136 \NewDocumentCommand\balmerpeak{G{}0{-4cm}}{
137     \begin{tikzpicture}
138     \hspace*{#2} %% anyhow necessary to fix centering ... strange :(
139     \begin{axis}
140     [width=10cm,height=7cm,
141     xmin=-0.005,xmax=0.28,ymin=-0.05,ymax=1,
142     xtick={0,0.02,...,0.27},ytick=\empty,
143     /pgf/number format/precision=3,/pgf/number format/fixed,
144     tick label style={font=\small},
145     label style = {font=\Large},
146     xlabel = \fontspec{Punk Nova} BLOOD ALCOHOL CONCENTRATION (\%),
147     ylabel = \fontspec{Punk Nova} \rotatebox{-90}{\parbox{3cm}{\center programming\ skills}}]
148     \addplot
149     [domain=-0.01:0.27,color=red,samples=250]

```

```

150      {0.8*exp(-0.5*((x-0.1335)^2)/.00002)+
151      0.5*exp(-0.5*((x+0.015)^2)/0.01)
152      };
153   \end{axis}
154   \end{tikzpicture}
155 }
156 \fi

```

6 Lua Module

This file contains all the necessary functions, sorted alphabetically, not by sense.

First, we set up some constants. These are made global so the code can be manipulated on document level, too.

```

157 Hhead = node.id("hhead")
158 RULE = node.id("rule")
159 GLUE = node.id("glue")
160 WHAT = node.id("whatsit")
161 COL = node.subtype("pdf_colorstack")
162 GLYPH = node.id("glyph")

```

Now we set up the nodes used for all color things. The nodes are whatsits of subtype pdf_colorstack.

```

163 color_push = node.new(WHAT,COL)
164 color_pop = node.new(WHAT,COL)
165 color_push.stack = 0
166 color_pop.stack = 0
167 color_push.cmd = 1
168 color_pop.cmd = 2

```

6.1 chickenize

The infamous \chickenize macro. Substitutes every word of the input with the given string. This can be elaborated arbitrarily, and whenever I feel like, I might add functionality. So far, only the string replaces the word, and even hyphenation is not possible.

```

169 chicken_pagenumbers = true
170
171 chickenstring = {}
172 chickenstring[1] = "Chicken" -- chickenstring is a table, please remeber this!
173
174 chickenizefraction = 0.5
175 -- set this to a small value to fool somebody, or to see if your text has been read carefully. Th
176
177 local tbl = font.getfont(font.current())
178 local space = tbl.parameters.space
179 local shrink = tbl.parameters.space_shrink

```

```

180 local stretch = tbl.parameters.space_stretch
181 local match = unicode.utf8.match
182 chickenize_ignore_word = false
183
184 chickenize_real_stuff = function(i,head)
185     while ((i.next.id == 37) or (i.next.id == 11) or (i.next.id == 7) or (i.next.id == 0)) do --
186         i.next = i.next.next
187     end
188
189     chicken = {} -- constructing the node list.
190
191 -- Should this be done only once? No, then we loose the freedom to change the string in-document.
192 --but it could be done only once each paragraph as in-paragraph changes are not possible!
193
194     chickenstring_tmp = chickenstring[math.random(1,#chickenstring)]
195     chicken[0] = node.new(37,1) -- only a dummy for the loop
196     for i = 1,string.len(chickenstring_tmp) do
197         chicken[i] = node.new(37,1)
198         chicken[i].font = font.current()
199         chicken[i-1].next = chicken[i]
200     end
201
202     j = 1
203     for s in string.utfvalues(chickenstring_tmp) do
204         local char = unicode.utf8.char(s)
205         chicken[j].char = s
206         if match(char,"%s") then
207             chicken[j] = node.new(10)
208             chicken[j].spec = node.new(47)
209             chicken[j].spec.width = space
210             chicken[j].spec.shrink = shrink
211             chicken[j].spec.stretch = stretch
212         end
213         j = j+1
214     end
215
216     node.slide(chicken[1])
217     lang.hyphenate(chicken[1])
218     chicken[1] = node.kerning(chicken[1]) -- FIXME: does not work
219     chicken[1] = node.ligaturing(chicken[1]) -- dito
220
221     node.insert_before(head,i,chicken[1])
222     chicken[1].next = chicken[2] -- seems to be necessary ... to be fixed
223     chicken[string.len(chickenstring_tmp)].next = i.next
224     return head
225 end

```

```

226
227 chickenize = function(head)
228   for i in node.traverse_id(37,head) do --find start of a word
229     if (chickenize_ignore_word == false) then -- normal case: at the beginning of a word, we jump
230       head = chickenize_real_stuff(i,head)
231     end
232
233 -- At the end of the word, the ignoring is reset. New chance for everyone.
234     if not((i.next.id == 37) or (i.next.id == 7) or (i.next.id == 22) or (i.next.id == 11)) then
235       chickenize_ignore_word = false
236     end
237
238 -- and the random determination of the chickenization of the next word:
239     if math.random() > chickenizefraction then
240       chickenize_ignore_word = true
241     end
242   end
243   return head
244 end
245
246 nicetext = function()
247   texio.write_nl("Output written on "..tex.jobname..".pdf ("..status.total_pages.." chicken,".." e
248   texio.write_nl(" ")
249   texio.write_nl("-----")
250   texio.write_nl("Hello my dear user,")
251   texio.write_nl("good job, now go outside and enjoy the world!")
252   texio.write_nl(" ")
253   texio.write_nl("And don't forget to feet your chicken!")
254   texio.write_nl("-----")
255 end

```

6.2 leetspeak

The leettable is the substitution scheme. Just add items if you feel to. Maybe we will differ between a light-weight version and a hardcore 1337.

```

256 leet_onlytext = false
257 leettable = {
258   [101] = 51, -- E
259   [105] = 49, -- I
260   [108] = 49, -- L
261   [111] = 48, -- O
262   [115] = 53, -- S
263   [116] = 55, -- T
264
265   [101-32] = 51, -- e

```

```

266 [105-32] = 49, -- i
267 [108-32] = 49, -- l
268 [111-32] = 48, -- o
269 [115-32] = 53, -- s
270 [116-32] = 55, -- t
271 }

```

And here the function itself. So simple that I will not write any

```

272 leet = function(head)
273   for line in node.traverse_id(Hhead,head) do
274     for i in node.traverse_id(GLYPH,line.head) do
275       if not(leetspeak_onlytext) or
276         node.has_attribute(i,luatexbase.attributes.leetattr)
277       then
278         if leettable[i.char] then
279           i.char = leettable[i.char]
280         end
281       end
282     end
283   end
284   return head
285 end

```

6.3 pancakenize

Not yet completely decided what this should do, but it might come down to inserting a cooking receipe for a ... well, guess what. Possible implementations are: Substitute a whole sentence, from full-stop to full-stop. OR: Substitute word-by-word at a random place. OR (expert-freak-1337-level): Substitute the n-th word of each page to a word of the receipe. That would be totally awesome!!

6.4 randomfonts

Traverses the output and substitutes fonts randomly. A check is done so that the font number is existing. One day, the fonts should be easily given explicately in terms of \bf etc.

```

286 randomfontslower = 1
287 randomfontsupper = 0
288 %
289 randomfonts = function(head)
290   if (randomfontsupper > 0) then -- fixme: this should be done only once, no? Or at every paragr
291     rfub = randomfontsupper -- user-specified value
292   else
293     rfub = font.max() -- or just take all fonts
294   end
295   for line in node.traverse_id(Hhead,head) do

```

```

296     for i in node.traverse_id(GLYPH,line.head) do
297         if not(randomfonts_onlytext) or node.has_attribute(i,luatexbase.attributes.randfontsattrib) then
298             i.font = math.random(randomfontslower,rfub)
299         end
300     end
301 end
302 return head
303 end

```

6.5 randomuclc

Traverses the input list and changes lowercase/uppercase codes.

```

304 uclcratio = 0.5 -- ratio between uppercase and lower case
305 randomuclc = function(head)
306     for i in node.traverse_id(37,head) do
307         if not(randomuclc_onlytext) or node.has_attribute(i,luatexbase.attributes.randuclcattrib) then
308             if math.random() < uclcratio then
309                 i.char = tex.uccode[i.char]
310             else
311                 i.char = tex.lccode[i.char]
312             end
313         end
314     end
315     return head
316 end

```

6.6 randomchars

```

317 randomchars = function(head)
318     for line in node.traverse_id(Hhead,head) do
319         for i in node.traverse_id(GLYPH,line.head) do
320             i.char = math.floor(math.random()*512)
321         end
322     end
323     return head
324 end

```

6.7 randomcolor and rainbowcolor

Setup of the boolean for grey/color or rainbowcolor, and boundaries for the colors. rgb space is fully used, but greyscale is only used in a visible range, i. e. to 90% instead of 100% white.

```

325 randomcolor_grey = false
326 randomcolor_onlytext = false --switch between local and global colorization
327 rainbowcolor = false
328

```

```

329 grey_lower = 0
330 grey_upper = 900
331
332 Rgb_lower = 1
333 rGb_lower = 1
334 rgB_lower = 1
335 Rgb_upper = 254
336 rGb_upper = 254
337 rgB_upper = 254

```

Variables for the rainbow. $1/\text{rainbow_step} \times 5$ is the number of letters used for one cycle, the color changes from red to yellow to green to blue to purple.

```

338 rainbow_step = 0.005
339 rainbow_Rgb = 1-rainbow_step -- we start in the red phase
340 rainbow_rGb = rainbow_step -- values x must always be  $0 < x < 1$ 
341 rainbow_rgB = rainbow_step
342 rainind = 1 -- 1:red,2:yellow,3:green,4:blue,5:purple

```

This function produces the string needed for the pdf color stack. We need values 0]..[1 for the colors.

```

343 randomcolorstring = function()
344   if randomcolor_grey then
345     return (0.001*math.random(grey_lower, grey_upper)).." g"
346   elseif rainbowcolor then
347     if rainind == 1 then -- red
348       rainbow_rGb = rainbow_rGb + rainbow_step
349       if rainbow_rGb >= 1-rainbow_step then rainind = 2 end
350     elseif rainind == 2 then -- yellow
351       rainbow_Rgb = rainbow_Rgb - rainbow_step
352       if rainbow_Rgb <= rainbow_step then rainind = 3 end
353     elseif rainind == 3 then -- green
354       rainbow_rgB = rainbow_rgB + rainbow_step
355       rainbow_rGb = rainbow_rGb - rainbow_step
356       if rainbow_rGb <= rainbow_step then rainind = 4 end
357     elseif rainind == 4 then -- blue
358       rainbow_Rgb = rainbow_Rgb + rainbow_step
359       if rainbow_Rgb >= 1-rainbow_step then rainind = 5 end
360     else -- purple
361       rainbow_rgB = rainbow_rgB - rainbow_step
362       if rainbow_rgB <= rainbow_step then rainind = 1 end
363     end
364     return rainbow_Rgb.." "..rainbow_rGb.." "..rainbow_rgB.." rg"
365   else
366     Rgb = math.random(Rgb_lower, Rgb_upper)/255
367     rGb = math.random(rGb_lower, rGb_upper)/255
368     rgB = math.random(rgB_lower, rgB_upper)/255
369     return Rgb.." "..rGb.." "..rgB.." " rg

```



```

370 end
371 end

```

The function that does all the colorizing action. It goes through the whole paragraph and looks at every glyph. If the boolean `randomcolor_onlytext` is set, only glyphs with the set attribute will be colored. Elsewise, all glyphs are taken.

```

372 randomcolor = function(head)
373   for line in node.traverse_id(0,head) do
374     for i in node.traverse_id(37,line.head) do
375       if not(randomcolor_onlytext) or
376         (node.has_attribute(i,luatexbase.attributes.randcolorattr))
377       then
378         color_push.data = randomcolorstring() -- color or grey string
379         line.head = node.insert_before(line.head,i,node.copy(color_push))
380         node.insert_after(line.head,i,node.copy(color_pop))
381       end
382     end
383   end
384   return head
385 end

```

6.8 rickroll

Another tribute to pop culture. Either: substitute word-by-word as in pancake. OR: substitute each link to a youtube-rickroll ...

6.9 uppercasecolor

Loop through all the nodes and checking whether it is uppercase. If so (and also for small caps), color it.

```

386 uppercasecolor = function (head)
387   for line in node.traverse_id(Hhead,head) do
388     for upper in node.traverse_id(GLYPH,line.head) do
389       if (((upper.char > 64) and (upper.char < 91)) or
390         ((upper.char > 57424) and (upper.char < 57451))) then -- for small caps! nice
391         color_push.data = randomcolorstring() -- color or grey string
392         line.head = node.insert_before(line.head,upper,node.copy(color_push))
393         node.insert_after(line.head,upper,node.copy(color_pop))
394       end
395     end
396   end
397   return head
398 end

```

6.10 colorstretch

This function displays the amount of stretching that has been done for each line of an arbitrary document. A well-typeset document should be equally grey over all lines, which is not always possible.

In fact, two boxes are drawn: The first (left) box shows the badness, i. e. the amount of stretching the spaces between words. Too much space results in light grey, whereas a too dense line is indicated by a dark grey box.

The second box is only useful if microtypographic extensions are used, e. g. with the `microtype` package under \LaTeX . The box color then corresponds to the amount of font expansion in the line. This can be greatly used to show the positive effect of font expansion on the badness of a line!

The base structure of the following code is written by Paul Isambert. Thanks for the code and support, Paul!

Two booleans, `keeptext`, and `colorexpansion`, are used to control the behaviour of the function.

```
399 keeptext = true
400 colorexpansion = true
401
402 colorstretch_coloroffset = 0.5
403 colorstretch_colorrange = 0.5
404 chickenize_rule_bad_height = 4/5 -- height and depth of the rules
405 chickenize_rule_bad_depth = 1/5
406
407
408 colorstretchnumbers = true
409 drawstretchthreshold = 0.1
410 drawexpansionthreshold = 0.9
```

After setting the constants, the function starts. It receives the vertical list of the typeset paragraph as head, and loops through all horizontal lists.

If font expansion should be shown (`colorexpansion == true`), then the first glyph node is determined and its width compared with the width of the unexpanded glyph. This gives a measure for the expansion factor and is translated into a grey scale.

```
411 colorstretch = function (head)
412
413   local f = font.getfont(font.current()).characters
414   for line in node.traverse_id(Hhead,head) do
415     local rule_bad = node.new(RULE)
416
417     if colorexpansion then -- if also the font expansion should be shown
418       local g = line.head
419       while not(g.id == 37) do
420         g = g.next
```

```

421         end
422         exp_factor = g.width / f[g.char].width
423         exp_color = colorstretch_coloroffset + (1-exp_factor)*10 .. " g"
424         rule_bad.width = 0.5*line.width  -- we need two rules on each line!
425     else
426         rule_bad.width = line.width  -- only the space expansion should be shown, only one rule
427     end

```

Height and depth of the rules are adapted to print a closed grey pattern, so no white interspace is left.

The glue order and sign can be obtained directly and are translated into a grey scale.

```

428     rule_bad.height = tex.baselineskip.width*chickenize_rule_bad_height -- this should give a bett
429     rule_bad.depth = tex.baselineskip.width*chickenize_rule_bad_depth
430
431     local glue_ratio = 0
432     if line.glue_order == 0 then
433         if line.glue_sign == 1 then
434             glue_ratio = colorstretch_colorange * math.min(line.glue_set,1)
435         else
436             glue_ratio = -colorstretch_colorange * math.min(line.glue_set,1)
437         end
438     end
439     color_push.data = colorstretch_coloroffset + glue_ratio .. " g"
440

```

Now, we throw everything together in a way that works. Somehow ...

```

441 -- set up output
442     local p = line.head
443
444 -- a rule to immitate kerning all the way back
445     local kern_back = node.new(RULE)
446     kern_back.width = -line.width
447
448 -- if the text should still be displayed, the color and box nodes are inserted additionally
449 -- and the head is set to the color node
450     if kepttext then
451         line.head = node.insert_before(line.head,line.head,node.copy(color_push))
452     else
453         node.flush_list(p)
454         line.head = node.copy(color_push)
455     end
456     node.insert_after(line.head,line.head,rule_bad)  -- then the rule
457     node.insert_after(line.head,line.head.next,node.copy(color_pop)) -- and then pop!
458     tmpnode = node.insert_after(line.head,line.head.next.next,kern_back)
459
460 -- then a rule with the expansion color
461     if colorexansion then  -- if also the stretch/shrink of letters should be shown

```

```

462     color_push.data = exp_color
463     node.insert_after(line.head,tmpnode,node.copy(color_push))
464     node.insert_after(line.head,tmpnode.next,node.copy(rule_bad))
465     node.insert_after(line.head,tmpnode.next.next,node.copy(color_pop))
466     end

```

Now we are ready with the boxes and stuff and everything. However, a very useful information might be the amount of stretching, not encoded as color, but the real value. In concreto, I mean: narrow boxes get one color, loose boxes get another one, but only if the badness is above a certain amount. This information is printed into the right-hand margin. The threshold is user-adjustable.

```

467     if colorstretchnumbers then
468         j = 1
469         glue_ratio_output = {}
470         for s in string.utfvalues(math.abs(glue_ratio)) do -- using math.abs here gets us rid of the
471             local char = unicode.utf8.char(s)
472             glue_ratio_output[j] = node.new(37,1)
473             glue_ratio_output[j].font = font.current()
474             glue_ratio_output[j].char = s
475             j = j+1
476         end
477         if math.abs(glue_ratio) > drawstretchthreshold then
478             if glue_ratio < 0 then color_push.data = "0.99 0 0 rg"
479             else color_push.data = "0 0.99 0 rg" end
480         else color_push.data = "0 0 0 rg"
481         end
482
483         node.insert_after(line.head,node.tail(line.head),node.copy(color_push))
484         for i = 1,math.min(j-1,7) do
485             node.insert_after(line.head,node.tail(line.head),glue_ratio_output[i])
486         end
487         node.insert_after(line.head,node.tail(line.head),node.copy(color_pop))
488     end -- end of stretch number insertion
489 end
490 return head
491 end

```

And that's it!



6.11 draw a chicken

A *very* first, experimental implementation of a drawing of a chicken. The parameters should be consistent, easy to change and that monster should look more like a cute chicken. However, it is chicken, it is Lua, so it belongs into this package. So far, all numbers and positions are hard coded, this will of course change!

```
492 --
493 function pdf_print (...)
494   for _, str in ipairs({...}) do
495     pdf.print(str .. " ")
496   end
497   pdf.print("\string\n")
498 end
499
500 function move (p)
501   pdf_print(p[1],p[2],"m")
502 end
503
504 function line (p)
505   pdf_print(p[1],p[2],"l")
506 end
507
508 function curve(p1,p2,p3)
509   pdf_print(p1[1], p1[2],
510             p2[1], p2[2],
511             p3[1], p3[2], "c")
512 end
513
514 function close ()
515   pdf_print("h")
516 end
517
518 function linewidth (w)
519   pdf_print(w,"w")
520 end
521
522 function stroke ()
523   pdf_print("S")
524 end
525 --
526
527 function strictcircle(center,radius)
528   local left = {center[1] - radius, center[2]}
529   local lefttop = {left[1], left[2] + 1.45*radius}
530   local leftbot = {left[1], left[2] - 1.45*radius}
531   local right = {center[1] + radius, center[2]}
```

```

532 local righttop = {right[1], right[2] + 1.45*radius}
533 local rightbot = {right[1], right[2] - 1.45*radius}
534
535 move (left)
536 curve (lefttop, righttop, right)
537 curve (rightbot, leftbot, left)
538 stroke()
539 end
540
541 function disturb_point(point)
542   return {point[1] + math.random()*5 - 2.5,
543           point[2] + math.random()*5 - 2.5}
544 end
545
546 function sloppycircle(center,radius)
547   local left = disturb_point({center[1] - radius, center[2]})
548   local lefttop = disturb_point({left[1], left[2] + 1.45*radius})
549   local leftbot = {lefttop[1], lefttop[2] - 2.9*radius}
550   local right = disturb_point({center[1] + radius, center[2]})
551   local righttop = disturb_point({right[1], right[2] + 1.45*radius})
552   local rightbot = disturb_point({right[1], right[2] - 1.45*radius})
553
554   local right_end = disturb_point(right)
555
556   move (right)
557   curve (rightbot, leftbot, left)
558   curve (lefttop, righttop, right_end)
559   linewidth(math.random()+0.5)
560   stroke()
561 end
562
563 function sloppyline(start,stop)
564   local start_line = disturb_point(start)
565   local stop_line = disturb_point(stop)
566   start = disturb_point(start)
567   stop = disturb_point(stop)
568   move(start) curve(start_line,stop_line,stop)
569   linewidth(math.random()+0.5)
570   stroke()
571 end

```

7 Known Bugs

The behaviour of the `\chickenize` macro is under construction and everything it does so far is considered a feature.

babel Using `chickenize` with `babel` leads to a problem with the `"` character, as it is made active: When using `\chickenizesetup` *after* `\begin{document}`, you can *not* use `"` for strings, but you have to use `'`. No problem really, but take care of this.

8 To Dos

Some things that should be implemented but aren't so far or are very poor at the moment:

rainbowcolor should be more flexible – the angle of the rainbow should be easily adjustable.

pancakenize should do something funny.

chickenize should differ between character and punctuation.

swing swing dancing apes!

chickenmath chickenization of math mode