

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

chickenize

Arno Trautmann
arno.trautmann@gmx.de

August 12, 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	How You Can Use It	3
2.1	TeX Commands – Document Wide	3
2.2	How to Deactivate It	4
2.3	\text-Versions	4
2.4	Lua functions	5
3	How to Adjust It	5
II	Implementation	7
4	TeX file	7
5	LT_εX package	9
5.1	Definition of User-Level Macros	9
6	Lua Module	10
6.1	chickenize	10
6.2	leet	12
6.3	randomfonts	13
6.4	randomucl	13
6.5	randomchars	13
6.6	randomcolor	14
6.7	uppercasecolor	15
6.8	colorstretch	16
7	Known Bugs	20
8	To Dos	20

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 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 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 ...

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 T_EX 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")}} % yes, I /am/ funny
11 \def\unchickenize{
12   \directlua{luatexbase.remove_from_callback("pre_linebreak_filter","chickenize")
13     luatexbase.remove_from_callback("start_page_number","cstartpage")
14     luatexbase.remove_from_callback("stop_page_number","cstoppage")}}
15
16 \def\colorstretch{
17   \directlua{luatexbase.add_to_callback("post_linebreak_filter",colorstretch,"stretch_expansion")}
18 \def\uncolorstretch{
19   \directlua{luatexbase.remove_from_callback("post_linebreak_filter","colorstretch")}}
20
21 \def\leetspeak{
22   \directlua{luatexbase.add_to_callback("post_linebreak_filter",leet,"1337")}}
23 \def\unleetspeak{
24   \directlua{luatexbase.remove_from_callback("post_linebreak_filter","1337")}}
25
26 \def\rainbowcolor{
27   \directlua{luatexbase.add_to_callback("post_linebreak_filter",randomcolor,"rainbowcolor")
28     rainbowcolor = true}}
29 \def\unrainbowcolor{
30   \directlua{luatexbase.remove_from_callback("post_linebreak_filter","rainbowcolor")
31     rainbowcolor = false}}
32 \let\nyanize\rainbowcolor
33 \let\unnyanize\unrainbowcolor
34
```

```

35 \def\pancakenize{
36   \directlua{}}
37 \def\unpancakenize{
38   \directlua{}}
39
40 \def\coffeestainize{
41   \directlua{}}
42 \def\uncoffeestainize{
43   \directlua{}}
44
45 \def\randomcolor{
46   \directlua{luatexbase.add_to_callback("post_linebreak_filter",randomcolor,"randomcolor")}}
47 \def\unrandomcolor{
48   \directlua{luatexbase.remove_from_callback("post_linebreak_filter","randomcolor")}}
49
50 \def\randomfonts{
51   \directlua{luatexbase.add_to_callback("post_linebreak_filter",randomfonts,"randomfonts")}}
52 \def\unrandomfonts{
53   \directlua{luatexbase.remove_from_callback("post_linebreak_filter","randomfonts")}}
54
55 \def\randomuclc{
56   \directlua{luatexbase.add_to_callback("pre_linebreak_filter",randomuclc,"randomuclc")}}
57 \def\unrandomuclc{
58   \directlua{luatexbase.remove_from_callback("pre_linebreak_filter","randomuclc")}}
59
60 \def\uppercasecolor{
61   \directlua{luatexbase.add_to_callback("post_linebreak_filter",uppercasecolor,"uppercasecolor")}}
62 \def\unuppercasecolor{
63   \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.

```

64 \newluaTeXattribute\leetattr
65 \newluaTeXattribute\randcolorattr
66 \newluaTeXattribute\randfontsattrib
67 \newluaTeXattribute\randuclcattrib
68
69 \long\def\textleetspeak#1%
70   {\setluaTeXattribute\leetattr{42}#1\unsetluaTeXattribute\leetattr}
71 \long\def\textrandomcolor#1%
72   {\setluaTeXattribute\randcolorattr{42}#1\unsetluaTeXattribute\randcolorattr}
73 \long\def\textrandomfonts#1%
74   {\setluaTeXattribute\randfontsattrib{42}#1\unsetluaTeXattribute\randfontsattrib}
75 \long\def\textrandomfontsattrib#1%
76   {\setluaTeXattribute\randfontsattrib{42}#1\unsetluaTeXattribute\randfontsattrib}
77 \long\def\textrandomuclc#1%

```



```
78 {\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.

```
79 \def\chickenizesetup#1{\directlua{#1}}
```

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.

```
80 \input{chickenize}
```

5.1 Definition of User-Level Macros

```
81 %% We want to "chickenize" figures, too. So ...
82 \iffalse
83 \DeclareDocumentCommand\includegraphics{0}{m}{
84     \fbox{Chicken} %% actually, I'd love to draw a mp graph showing a chicken ...
85 }
86 %%%% specials: the balmerpeak. A tribute to http://xkcd.com/323/.
87 %% So far, you have to load pgfplots yourself.
88 %% As it is a mighty package, I don't want the user to force loading it.
89 \NewDocumentCommand\balmerpeak{G}{0{-4cm}}{
90     \begin{tikzpicture}
91     \hspace*{#2} %% anyhow necessary to fix centering ... strange :(
92     \begin{axis}
93     [width=10cm,height=7cm,
94     xmin=-0.005,xmax=0.28,ymin=-0.05,ymax=1,
95     xtick={0,0.02,...,0.27},ytick=\empty,
96     /pgf/number format/precision=3,/pgf/number format/fixed,
97     tick label style={font=\small},
98     label style = {font=\Large},
99     xlabel = \fontspec{Punk Nova} BLOOD ALCOHOL CONCENTRATION (\%),
100    ylabel = \fontspec{Punk Nova} \rotatebox{-90}{\parbox{3cm}{\center programming\\ skills}}]
101    \addplot
102    [domain=-0.01:0.27,color=red,samples=250]
103    {0.8*exp(-0.5*((x-0.1335)^2)/.00002)+
104    0.5*exp(-0.5*((x+0.015)^2)/0.01)
105    };
106    \end{axis}
107    \end{tikzpicture}
108 }
```

109 \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.

```
110 Hhead = node.id("hhead")
111 RULE = node.id("rule")
112 GLUE = node.id("glue")
113 WHAT = node.id("whatsit")
114 COL = node.subtype("pdf_colorstack")
115 GLYPH = node.id("glyph")
```

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

```
116 color_push = node.new(WHAT, COL)
117 color_pop = node.new(WHAT, COL)
118 color_push.stack = 0
119 color_pop.stack = 0
120 color_push.cmd = 1
121 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.

```
122 chickenstring = {}
123 chickenstring[1] = "Chicken" -- chickenstring is a table, please remeber this!
124
125 chickenizefraction = 0.5
126 -- set this to a small value to fool somebody, or to see if your text has been read carefully. Th
127
128 local tbl = font.getfont(font.current())
129 local space = tbl.parameters.space
130 local shrink = tbl.parameters.space_shrink
131 local stretch = tbl.parameters.space_stretch
132 local match = unicode.utf8.match
133 chickenize_ignore_word = false
134
135 chickenize_real_stuff = function(i, head)
136     while ((i.next.id == 37) or (i.next.id == 11) or (i.next.id == 7) or (i.next.id == 0)) do --
137         i.next = i.next.next
138     end
```

```

139
140     chicken = {} -- constructing the node list.
141
142 -- Should this be done only once? No, then we loose the freedom to change the string in-document.
143 --but it could be done only once each paragraph as in-paragraph changes are not possible!
144
145     chickenstring_tmp = chickenstring[math.random(1,#chickenstring)]
146     chicken[0] = node.new(37,1) -- only a dummy for the loop
147     for i = 1,string.len(chickenstring_tmp) do
148         chicken[i] = node.new(37,1)
149         chicken[i].font = font.current()
150         chicken[i-1].next = chicken[i]
151     end
152
153     j = 1
154     for s in string.utfvalues(chickenstring_tmp) do
155         local char = unicode.utf8.char(s)
156         chicken[j].char = s
157         if match(char,"%s") then
158             chicken[j] = node.new(10)
159             chicken[j].spec = node.new(47)
160             chicken[j].spec.width = space
161             chicken[j].spec.shrink = shrink
162             chicken[j].spec.stretch = stretch
163         end
164         j = j+1
165     end
166
167     node.slide(chicken[1])
168     lang.hyphenate(chicken[1])
169     chicken[1] = node.kerning(chicken[1]) -- FIXME: does not work
170     chicken[1] = node.ligaturing(chicken[1]) -- dito
171
172     node.insert_before(head,i,chicken[1])
173     chicken[1].next = chicken[2] -- seems to be necessary ... to be fixed
174     chicken[string.len(chickenstring_tmp)].next = i.next
175     return head
176 end
177
178 chickenize = function(head)
179     for i in node.traverse_id(37,head) do --find start of a word
180         if (chickenize_ignore_word == false) then -- normal case: at the beginning of a word, we jump
181             head = chickenize_real_stuff(i,head)
182         end
183     end
184 -- At the end of the word, the ignoring is reset. New chance for everyone.

```

```

185     if not((i.next.id == 37) or (i.next.id == 7) or (i.next.id == 22) or (i.next.id == 11)) then
186         chickenize_ignore_word = false
187     end
188
189 -- and the random determination of the chickenization of the next word:
190     if math.random() > chickenizefraction then
191         chickenize_ignore_word = true
192     end
193 end
194 return head
195 end

```

6.2 leet

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.

```

196 leet_onlytext = false
197 leettable = {
198     [101] = 51, -- E
199     [105] = 49, -- I
200     [108] = 49, -- L
201     [111] = 48, -- O
202     [115] = 53, -- S
203     [116] = 55, -- T
204
205     [101-32] = 51, -- e
206     [105-32] = 49, -- i
207     [108-32] = 49, -- l
208     [111-32] = 48, -- o
209     [115-32] = 53, -- s
210     [116-32] = 55, -- t
211 }

```

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

```

212 leet = function(head)
213     for line in node.traverse_id(Hhead,head) do
214         for i in node.traverse_id(GLYPH,line.head) do
215             if not(leetspeak_onlytext) or
216                 node.has_attribute(i,luatexbase.attributes.leetattr)
217             then
218                 if leettable[i.char] then
219                     i.char = leettable[i.char]
220                 end
221             end
222         end
223     end

```

```

224 return head
225 end

```

6.3 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 explicitly in terms of \bf etc.

```

226 randomfontslower = 1
227 randomfontsupper = 0
228 %
229 randomfonts = function(head)
230   if (randomfontsupper > 0) then -- fixme: this should be done only once, no? Or at every paragr
231     rfub = randomfontsupper -- user-specified value
232   else
233     rfub = font.max() -- or just take all fonts
234   end
235   for line in node.traverse_id(Hhead,head) do
236     for i in node.traverse_id(GLYPH,line.head) do
237       if not(randomfonts_onlytext) or node.has_attribute(i,luatexbase.attributes.randfontsattrib) then
238         i.font = math.random(randomfontslower,rfub)
239       end
240     end
241   end
242   return head
243 end

```

6.4 randomuclc

Traverses the input list and changes lowercase/uppercase codes.

```

244 uclcratio = 0.5 -- ratio between uppercase and lower case
245 randomuclc = function(head)
246   for i in node.traverse_id(37,head) do
247     if not(randomuclc_onlytext) or node.has_attribute(i,luatexbase.attributes.randuclcattrib) then
248       if math.random() < uclcratio then
249         i.char = tex.uccode[i.char]
250       else
251         i.char = tex.lccode[i.char]
252       end
253     end
254   end
255   return head
256 end

```

6.5 randomchars

```

257 randomchars = function(head)
258   for line in node.traverse_id(Hhead,head) do
259     for i in node.traverse_id(GLYPH,line.head) do
260       i.char = math.floor(math.random()*512)
261     end
262   end
263   return head
264 end

```

6.6 randomcolor

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.

```

265 randomcolor_grey = false
266 randomcolor_onlytext = false --switch between local and global colorization
267 rainbowcolor = false
268
269 grey_lower = 0
270 grey_upper = 900
271
272 Rgb_lower = 1
273 rGb_lower = 1
274 rgB_lower = 1
275 Rgb_upper = 254
276 rGb_upper = 254
277 rgB_upper = 254

```

Variables for the rainbow. 1/rainbow_step*5 is the number of letters used for one cycle, the color changes from red to yellow to green to blue to purple.

```

278 rainbow_step = 0.005
279 rainbow_Rgb = 1-rainbow_step -- we start in the red phase
280 rainbow_rGb = rainbow_step -- values x must always be 0 < x < 1
281 rainbow_rgB = rainbow_step
282 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.

```

283 randomcolorstring = function()
284   if randomcolor_grey then
285     return (0.001*math.random(grey_lower,grey_upper)).." g"
286   elseif rainbowcolor then
287     if rainind == 1 then -- red
288       rainbow_rGb = rainbow_rGb + rainbow_step
289       if rainbow_rGb >= 1-rainbow_step then rainind = 2 end
290     elseif rainind == 2 then -- yellow
291       rainbow_Rgb = rainbow_Rgb - rainbow_step

```

```

292     if rainbow_Rgb <= rainbow_step then rainind = 3 end
293 elseif rainind == 3 then -- green
294     rainbow_rgB = rainbow_rgB + rainbow_step
295     rainbow_rGb = rainbow_rGb - rainbow_step
296     if rainbow_rGb <= rainbow_step then rainind = 4 end
297 elseif rainind == 4 then -- blue
298     rainbow_Rgb = rainbow_Rgb + rainbow_step
299     if rainbow_Rgb >= 1-rainbow_step then rainind = 5 end
300 else -- purple
301     rainbow_rgB = rainbow_rgB - rainbow_step
302     if rainbow_rGb <= rainbow_step then rainind = 1 end
303 end
304 return rainbow_Rgb.." "..rainbow_rGb.." "..rainbow_rgB.." rg"
305 else
306   Rgb = math.random(Rgb_lower,Rgb_upper)/255
307   rGb = math.random(rGb_lower,rGb_upper)/255
308   rgB = math.random(rgB_lower,rgB_upper)/255
309   return Rgb.." "..rGb.." "..rgB.." .." rg"
310 end
311 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. Otherwise, all glyphs are taken.

```

312 randomcolor = function(head)
313   for line in node.traverse_id(0,head) do
314     for i in node.traverse_id(37,line.head) do
315       if not(randomcolor_onlytext) or
316         (node.has_attribute(i,luatexbase.attributes.randcolorattr))
317       then
318         color_push.data = randomcolorstring() -- color or grey string
319         line.head = node.insert_before(line.head,i,node.copy(color_push))
320         node.insert_after(line.head,i,node.copy(color_pop))
321       end
322     end
323   end
324   return head
325 end

```

6.7 uppercasecolor

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

```

326 uppercasecolor = function (head)
327   for line in node.traverse_id(Hhead,head) do
328     for upper in node.traverse_id(GLYPH,line.head) do

```

```

329     if (((upper.char > 64) and (upper.char < 91)) or
330         ((upper.char > 57424) and (upper.char < 57451))) then -- for small caps! nice
331         color_push.data = randomcolorstring() -- color or grey string
332         line.head = node.insert_before(line.head,upper,node.copy(color_push))
333         node.insert_after(line.head,upper,node.copy(color_pop))
334     end
335 end
336 end
337 return head
338 end

```

6.8 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 gray, 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 `colorexpan`, are used to control the behaviour of the function.

```

339 keeptext = true
340 colorexpansion = true
341 drawstretchnumbers = true
342 drawstretchthreshold = 0.1
343 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 (`colorexpan == 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.

```

344 colorstretch = function (head)
345
346   local f = font.getfont(font.current()).characters
347   for line in node.traverse_id(Hhead,head) do
348     local rule_bad = node.new(RULE)
349

```



```

350 if colorexansion then -- if also the font expansion should be shown
351     local g = line.head
352     while not(g.id == 37) do
353         g = g.next
354     end
355     exp_factor = g.width / f[g.char].width
356     exp_color = .5 + (1-exp_factor)*10 .. " g"
357     rule_bad.width = 0.5*line.width -- we need two rules on each line!
358 else
359     rule_bad.width = line.width -- only the space expansion should be shown, only one rule
360 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.

```

361 rule_bad.height = tex.baselineskip.width*4/5 -- this should give a better output
362 rule_bad.depth = tex.baselineskip.width*1/5
363
364 local glue_ratio = 0
365 if line.glue_order == 0 then
366     if line.glue_sign == 1 then
367         glue_ratio = .5 * math.min(line.glue_set,1)
368     else
369         glue_ratio = -.5 * math.min(line.glue_set,1)
370     end
371 end
372 color_push.data = .5 + glue_ratio .. " g"
373

```

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

```

374 -- set up output
375 local p = line.head
376
377 -- a rule to immitate kerning all the way back
378 local kern_back = node.new(RULE)
379 kern_back.width = -line.width
380
381 -- if the text should still be displayed, the color and box nodes are inserted additionally
382 -- and the head is set to the color node
383 if keeptext then
384     line.head = node.insert_before(line.head,line.head,node.copy(color_push))
385 else
386     node.flush_list(p)
387     line.head = node.copy(color_push)
388 end
389 node.insert_after(line.head,line.head,rule_bad) -- then the rule
390 node.insert_after(line.head,line.head.next,node.copy(color_pop)) -- and then pop!

```

```

391     tmpnode = node.insert_after(line.head,line.head.next.next,kern_back)
392
393     -- then a rule with the expansion color
394     if colorexansion then -- if also the stretch/shrink of letters should be shown
395         color_push.data = exp_color
396         node.insert_after(line.head,tmpnode,node.copy(color_push))
397         node.insert_after(line.head,tmpnode.next,node.copy(rule_bad))
398         node.insert_after(line.head,tmpnode.next.next,node.copy(color_pop))
399     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.

```

400     if drawstretchnumbers then
401         j = 1
402         glue_ratio_output = {}
403         for s in string.utfvalues(math.abs(glue_ratio)) do -- using math.abs here gets us rid of the
404             local char = unicode.utf8.char(s)
405             glue_ratio_output[j] = node.new(37,1)
406             glue_ratio_output[j].font = font.current()
407             glue_ratio_output[j].char = s
408             j = j+1
409         end
410         if math.abs(glue_ratio) > drawstretchthreshold then
411             if glue_ratio < 0 then color_push.data = "0.99 0 0 rg"
412             else color_push.data = "0 0.99 0 rg" end
413         else color_push.data = "0 0 0 rg"
414         end
415
416         node.insert_after(line.head,node.tail(line.head),node.copy(color_push))
417         for i = 1,math.min(j-1,7) do
418             node.insert_after(line.head,node.tail(line.head),glue_ratio_output[i])
419         end
420         node.insert_after(line.head,node.tail(line.head),node.copy(color_pop))
421     end -- end of stretch number insertion
422 end
423 return head
424 end

```

And that's it!



Well, not totally ... I've decided to print a small message at the very end of everything.
It's nice to hear some friendly words from your computer, isn't it?

```
425 nicetext = function()
426   texio.write_nl(" ")
427   texio.write_nl("-----")
428   texio.write_nl(" ")
429   texio.write_nl("Heyho dear user,")
430   texio.write_nl("good job you did there, fine document.")
431   texio.write_nl("I had fun compiling this, really!")
432   texio.write_nl(" ")
433   texio.write_nl("Now, leave the computer, go outside, have fun")
434   texio.write_nl("and enjoy your life. There are so many great things out there!")
435   texio.write_nl("Have a nice day!!!11")
436   texio.write_nl(" ")
437   texio.write_nl("And don't forget to feed your chicken!")
438   texio.write_nl(" ")
439   texio.write_nl("-----")
440   texio.write_nl(" ")
441   texio.write_nl(" ")
442 end
443
444 luatexbase.add_to_callback("stop_run",nicetext,"a nice text")
```

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.