# Typesetting Code Listings and Emulating Screenshots with LaTeX Beautifully

https://github.com/xziyue/latex-beautiful-listings-screenshot

Ziyue “Alan” Xiang

May 9, 2020

## Contents

# 1. Quick Guide

Download `customlisting.sty` from source and place it in your project folder. Load the style with `\usepackage{customlisting}`.

# A. Source code of `customlisting.sty`

*Code*

```
\RequirePackage{listings}
\RequirePackage[breakable, skins]{tcolorbox}
\RequirePackage{minted}

\tcbuselibrary{minted, listings}

\newmintinline[cinline]{c}{frame=none, fontsize=\fontsize{10}{10}}
\newmintinline[rawinline]{text}{frame=none, fontsize=\fontsize{10}{10}}
\newmintinline[pyinline]{python}{frame=none, fontsize=\fontsize{10}{10}}


\lstset{
    basicstyle=\ttfamily\selectfont;
}

\definecolor{mygreen}{rgb}{0,0.6,0}
\definecolor{mygray}{rgb}{0.5,0.5,0.5}
\definecolor{mymauve}{rgb}{0.58,0,0.82}
\definecolor{clrconsoleframe}{HTML}{1aa3ff}
\definecolor{clrconsoleback}{HTML}{e6f5ff}

\definecolor{terminalbgcolor}{HTML}{330033}
\definecolor{terminalrulecolor}{HTML}{000099}

\definecolor{terminalbggray}{rgb}{0.97,0.97,0.97}

\newtcblisting{tcbconsole}{
    listing only,
    enhanced jigsaw, breakable, colback=clrconsoleback, boxsep=0pt, colframe=clrconsoleframe,
    ↪ top=0pt, bottom=0pt, left=2mm, right=2mm, boxrule=2pt, title={\itshape Terminal},
    ↪ fontupper=\fontfamily{GoMono-TLF}\fontsize{9}{11}\selectfont,
    listing options={
        backgroundcolor=\color{clrconsoleback},   % choose the background color; you must add
        ↪ \usepackage{color} or \usepackage{xcolor}; should come as last argument
            basicstyle=\linespread{0.8}\fontfamily{DejaVuSansMono-TLF}\fontsize{8}{8}\selectfont,
            ↪ % the size of the fonts that are used for the code
            breakatwhitespace=false,         % sets if automatic breaks should only happen at
            ↪ whitespace
            breaklines=true,                 % sets automatic line breaking
            captionpos=b,                    % sets the caption-position to bottom
            commentstyle=\color{mygreen},    % comment style
            deletekeywords={...},            % if you want to delete keywords from the given
            ↪ language
            escapeinside={\%*}{*)},          % if you want to add LaTeX within your code
            extendedchars=true,              % lets you use non-ASCII characters; for 8-bits
            ↪ encodings only, does not work with UTF-8
            frame=none,                     % adds a frame around the code
            %rulesepcolor=\color{black},
            keepspaces=true,                 % keeps spaces in text, useful for keeping
            ↪ indentation of code (possibly needs columns=flexible)
            keywordstyle=\color{blue},       % keyword style
            %language=none,                 % the language of the code
            morekeywords={*,...},            % if you want to add more keywords to the set
            numbers=none,                    % where to put the line-numbers; possible values are
            ↪ (none, left, right)
            numbersep=5pt,                   % how far the line-numbers are from the code
            %framerule=3pt,
```

```
            numberstyle=\color{mygray}\fontsize{7}{7}\selectfont, % the style that is used for the line-numbers
            rulecolor=\color{terminalrulecolor},         % if not set, the frame-color may be changed on line-breaks within not-black text (e.g. comments (green here))
            showspaces=false,                % show spaces everywhere adding particular underscores; it overrides 'showstringspaces'
            showstringspaces=false,          % underline spaces within strings only
            showtabs=false,                  % show tabs within strings adding particular underscores
            stepnumber=2,                    % the step between two line-numbers. If it's 1, each line will be numbered
            stringstyle=\color{mymauve},     % string literal style
            tabsize=2,                      % sets default tabsize to 2 spaces
            %title=\lstname                   % show the filename of files included with \lstinputlisting; also try caption instead of title
    }
}


\definecolor{clrcodeframe}{HTML}{00b33c}
\definecolor{clrcodeback}{HTML}{e6ffee}



\newtcblisting{tcbcode}[1]{
    listing only, listing engine=minted,
    enhanced jigsaw, breakable, colback=clrcodeback, boxsep=0pt, colframe=clrcodeframe, top=5pt, bottom=5pt, left=8mm, right=2mm, boxrule=2pt, title={\hspace*{-6mm}\itshape Code},
    minted options={linenos,autogobble,breaklines, numbersep=3mm, obeytabs, tabsize=4,fontsize=\fontsize{8}{8}},
    minted language=#1
}

\newtcbinputlisting{\tcbinputcode}[2]{
    listing only, listing engine=minted,
    enhanced jigsaw, breakable, colback=clrcodeback, boxsep=0pt, colframe=clrcodeframe, top=5pt, bottom=5pt, left=8mm, right=2mm, boxrule=2pt, title={\hspace*{-6mm}\itshape Code},
    minted options={linenos,autogobble,breaklines, numbersep=3mm, obeytabs, tabsize=4,fontsize=\fontsize{8}{8}},
    minted language=#1, listing file=#2
}

\definecolor{clrverbframe}{HTML}{ff4d94}
\definecolor{clrverbback}{HTML}{ffe6f0}

\newtcblisting{tcbverbatim}{
    listing only,
    enhanced jigsaw, breakable, colback=clrverbback, boxsep=0pt, colframe=clrverbframe, top=0pt, bottom=0pt, left=2mm, right=2mm, boxrule=2pt, title={\itshape Verbatim},
    listing options={
        backgroundcolor=\color{clrverbback},   % choose the background color; you must add \usepackage{color} or \usepackage{xcolor}; should come as last argument

            basicstyle=\linespread{0.9}\fontsize{5}{5}\fontfamily{lmtt}\fontseries{lc}\selectfont, % the size of the fonts that are used for the code
            breakatwhitespace=false,         % sets if automatic breaks should only happen at whitespace
            breaklines=true,                 % sets automatic line breaking
            captionpos=b,                    % sets the caption-position to bottom
            commentstyle=\color{mygreen},    % comment style
            deletekeywords={...},            % if you want to delete keywords from the given language
            escapeinside={\%*}{*)},          % if you want to add LaTeX within your code
            extendedchars=true,              % lets you use non-ASCII characters; for 8-bits encodings only, does not work with UTF-8
            frame=none,                     % adds a frame around the code
            keepspaces=true,                 % keeps spaces in text, useful for keeping indentation of code (possibly needs columns=flexible)
            keywordstyle=\color{blue},       % keyword style
            %language=none,                 % the language of the code
            morekeywords={*,...},            % if you want to add more keywords to the set
```

```
            numbers=none,                    % where to put the line-numbers; possible values are
            ↪ (none, left, right)
            numbersep=10pt,                   % how far the line-numbers are from the code
            numberstyle=\color{mygray}\fontsize{7}{7}\ttfamily\selectfont, % the style that is
            ↪ used for the line-numbers
            rulecolor=\color{black},         % if not set, the frame-color may be changed on
            ↪ line-breaks within not-black text (e.g. comments (green here))
            showspaces=false,                % show spaces everywhere adding particular
            ↪ underscores; it overrides 'showstringspaces'
            showstringspaces=false,          % underline spaces within strings only
            showtabs=false,                  % show tabs within strings adding particular
            ↪ underscores
            %framerule=1pt,
            stepnumber=2,                    % the step between two line-numbers. If it's 1, each
            ↪ line will be numbered
            stringstyle=\color{mymauve},     % string literal style
            tabsize=2,                      % sets default tabsize to 2 spaces
            %title=\lstname                   % show the filename of files included with
            ↪ \lstinputlisting; also try caption instead of title
    }
}
```

## B. Source code of `html2tex_gui.py`

*Code*

```python
from html.parser import HTMLParser
from colour import Color
from pylatex.utils import escape_latex, NoEscape
import re
import wx


def get_default_entity():
    return {
        'tag': None,
        'data': [],
        'attrs': None,
        'last_pointer': None
    }


class AhaHTMLParser(HTMLParser):
    def __init__(self):
        super().__init__()

        self.root = get_default_entity()
        self.root['tag'] = '@root'
        self.treeStorage = [self.root]

        self.curPointer = self.root

    def handle_starttag(self, tag, attrs):
        # create new structure in the tree
        entity = get_default_entity()
        entity['last_pointer'] = self.curPointer
        entity['tag'] = tag
        entity['attrs'] = attrs
        self.treeStorage.append(entity)
        self.curPointer = entity

    def handle_endtag(self, tag):
        # append this entity to last pointer
        self.curPointer['last_pointer']['data'].append(self.curPointer)
        self.curPointer = self.curPointer['last_pointer']

```

```
41      def handle_data(self, data):
42          # append data to current pointer
43          dataEntity = get_default_entity()
44          dataEntity['data'].append(data)
45          self.curPointer['data'].append(dataEntity)
46
47
48  def get_html_tree_f(filename):
49      with open(filename, 'r') as infile:
50          htmlContent = infile.read()
51      parser = AhaHTMLParser()
52      parser.feed(htmlContent)
53      return (parser.root, parser.treeStorage)
54
55  def get_html_tree(text):
56      parser = AhaHTMLParser()
57      parser.feed(text)
58      return (parser.root, parser.treeStorage)
59
60  def find_pre_in_tree(node):
61      if node['tag'] == 'pre':
62          return node
63
64      for data in node['data']:
65          if isinstance(data, dict):
66              result = find_pre_in_tree(data)
67              if result is not None:
68                  return result
69
70      return None
71
72
73  def parse_css_style(styleStr):
74      styles = styleStr.split(';')
75      result = dict()
76      for style in styles:
77          if len(style) == 0:
78              continue
79
80          key, val = style.split(':')
81          key = key.strip()
82          val = val.strip()
83          assert len(key) > 0
84          assert len(val) > 0
85          result[key] = val
86
87      return result
88
89
90  class HTMLTree2Latex:
91
92      def __init__(self):
93          self.colorConv = dict()
94          self.result = []
95
96          self.colorNameConvDict = dict()
97          for i in range(ord('A'), ord('F') + 1):
98              self.colorNameConvDict[chr(i)] = chr(i)
99          for i in range(ord('0'), ord('9') + 1):
100             self.colorNameConvDict[chr(i)] = chr(ord('F') + 1 + i - ord('0'))
101
102
103     def to_latex(self, node):
104         self.result.clear()
105         self.colorConv.clear()
106
107         self._to_latex(node)
108         # generate color definition
109         colorDefFormat = r'\definecolor{%s}{HTML}{%s}'
110
111         colorDefs = []
```

```
        for key, val in self.colorConv.items():
            colorDef = colorDefFormat % (val['latex_name'], val['value'])
            colorDefs.append(NoEscape(colorDef))

        colorDefStr = '\n'.join(colorDefs)

        return colorDefStr, self.result

    def _get_color_item(self, colorStr):
        if colorStr[0] == '#':
            assert len(colorStr) == 7
            capStr = colorStr[1:].upper()
            capStr = ''.join([self.colorNameConvDict[x] for x in capStr])

            if capStr not in self.colorConv:
                colorItem = dict()
                colorItem['latex_name'] = self._get_color_latex_name(capStr)
                colorItem['value'] = colorStr[1:]
                self.colorConv[capStr] = colorItem

            colorStr = capStr

        if colorStr not in self.colorConv:
            # create new color item
            colorItem = dict()
            colorItem['latex_name'] = self._get_color_latex_name(colorStr)
            colorItem['value'] = Color(colorStr).get_hex_l()[1:]
            self.colorConv[colorStr] = colorItem

        colorItem = self.colorConv[colorStr]
        return colorItem

    def _get_color_latex_name(self, color):
        return 'xxxhtmlcolor{}'.format(color)

    def _escape_utf8(self, data):
        reconData = []
        for s in data:
            if ord(s) < 128:
                reconData.append(s)
            else:
                sInd = ord(s)
                if sInd == 0xfffd:
                    reconData.append('%*\\ucr*)')
                elif sInd == 0x2588:
                    reconData.append(r'%*$\blacksquare$*)')
                else:
                    hexCode = '{:x}'.format(ord(s))
                    escapedS = '%*\\unichar{{\"{}}}*)'.format(hexCode)
                    reconData.append(escapedS)
        return NoEscape(''.join(reconData))

    # allow consecutive white spaces in latex
    def _escape_whitespace(self, data):
        reconData = []
        for s in data:
            if s == ' ':
                reconData.append(NoEscape(r'\space '))
            else:
                reconData.append(s)
        return reconData

    def _to_latex(self, node, inLatex = False):
        result = self.result

        # process style
        hasLatex = False
        endCap = []

        startResultSize = len(result)

```

```
183         # dealing with specific tags
184         if node['tag'] == 'b':
185             hasLatex = True
186             result.append(NoEscape(r'{\bfseries '))
187             endCap.insert(0, NoEscape('}'))
188         elif node['tag'] == 'font':
189             # dealing with font color
190             for key, val in node['attrs']:
191                 if key == 'color':
192                     hasLatex = True
193                     colorStr = val
194                     colorItem = self._get_color_item(colorStr)
195                     result.append(NoEscape(r'{\color{%s}' % colorItem['latex_name']))
196                     endCap.insert(0, NoEscape('}'))
197         else:
198             # dealing with generic tags with css styles
199             if node['attrs'] is not None:
200                 for key, val in node['attrs']:
201                     if key == 'style':
202                         # if there is style, then the entity has to be escaped
203                         hasLatex = True
204                         cssStyle = parse_css_style(val)
205 
206                         if 'font-weight' in cssStyle:
207                             if cssStyle['font-weight'] == 'bold':
208                                 result.append(NoEscape(r'{\bfseries '))
209                                 endCap.insert(0, NoEscape('}'))
210 
211                         if 'color' in cssStyle:
212                             colorStr = cssStyle['color']
213                             colorItem = self._get_color_item(colorStr)
214                             result.append(NoEscape(r'{\color{%s}' % colorItem['latex_name']))
215                             endCap.insert(0, NoEscape('}'))
216 
217                         if 'background-color' in cssStyle:
218                             self.addColorBoxDef = True
219                             colorStr = cssStyle['background-color']
220                             colorItem = self._get_color_item(colorStr)
221 
                                ↪  result.append(NoEscape(r'\smash{\colorbox{%s}{'%colorItem['latex_name']))
222                             endCap.insert(0, NoEscape('}}'))
223 
224         if hasLatex:
225             if not inLatex:
226                 result.insert(startResultSize, '%*')
227                 endCap.append('*)')
228                 inLatex = True
229 
230         startResultSize = len(result)
231 
232         for data in node['data']:
233             if isinstance(data, str):
234                 if inLatex:
235                     # if in escape mode, just put in UTF-8 characters
236                     result.extend(self._escape_whitespace(data))
237                 else:
238                     result.append(self._escape_utf8(data))
239 
240             elif isinstance(data, dict):
241                 self._to_latex(data, inLatex)
242 
243         if hasLatex:
244             for i in range(startResultSize, len(result)):
245                 if not isinstance(result[i], NoEscape):
246                     # preserve NoEscape
247                     result[i] = escape_latex(result[i])
248             result.extend(endCap)
249 
250 
251 def html_to_console_style_f(filename):
252     root, tree = get_html_tree_f(filename)
```

```
    preEntity = find_pre_in_tree(root)
    assert preEntity is not None
    html2altex = HTMLTree2Latex()
    colorDef, content = html2altex.to_latex(preEntity)

    outputFmt = r'''{\lstconsolestylenf
%s
\begin{consolebox}
\begin{lstlisting}
%s
\end{lstlisting}
\end{consolebox}}'''

    return outputFmt%(colorDef, ''.join(content))

def html_to_console_style(text):
    root, tree = get_html_tree(text)
    preEntity = find_pre_in_tree(root)
    assert preEntity is not None
    html2altex = HTMLTree2Latex()
    colorDef, content = html2altex.to_latex(preEntity)

    outputFmt = r'''{
%s
\setlength{\fboxsep}{1pt}
\begin{tcbconsole}
%s
\end{tcbconsole}
}'''

    result = outputFmt % (colorDef, ''.join(content).strip())
    return result.encode('utf8')

if __name__ == '__main__':
    # run the GUI

    class MyFrame(wx.Frame):

        def __init__(self, *args, **kwargs):
            super().__init__(*args, **kwargs)

            sizer = wx.BoxSizer(wx.VERTICAL)
            self.panel = wx.Panel(self)

            self.SetSize(wx.Size(800, 600))
            self.SetTitle('HTML2LaTeX')

            self.textIn = wx.TextCtrl(self.panel, style=wx.TE_MULTILINE)
            self.textOut = wx.TextCtrl(self.panel, style=wx.TE_MULTILINE | wx.TE_READONLY)
            sizer.Add(self.textIn, 1, wx.ALL | wx.EXPAND, 10)
            sizer.Add(self.textOut, 1, wx.ALL | wx.EXPAND, 10)

            self.btnConv = wx.Button(self.panel, label='Convert')
            self.btnConv.Bind(wx.EVT_BUTTON, self.evtBtn)
            sizer.Add(self.btnConv, 0, wx.ALL | wx.ALIGN_CENTER, 5)

            self.panel.SetSizerAndFit(sizer)

            self.Show()

        def evtBtn(self, evt):
            inText = self.textIn.GetValue()
            result = None
            try:
                result = html_to_console_style(inText)
            except Exception as e:
                wx.MessageBox('An exception occured during conversion: {}'.format(repr(e)),
                ↪  'Exception', wx.OK | wx.ICON_ERROR)

            if result is not None:
                self.textOut.SetValue(result)
```

```
323
324
325         app = wx.App()
326         MyFrame(None)
327         app.MainLoop()
```