```python
__author__ = 'David Owen'

import os
import os.path
import shutil
import sys
import subprocess

import pygments.formatters
import pygments.lexers

from processing_lexer import ProcessingLexer
from idle_style import IdleStyle

INLINE_STYLES = True
HTML_FOR_CANVAS = True
ADD_CANVAS_CSS = True
MATHJAX = False
DEFAULT_OUTPUT = 'html'
DEFAULT_STYLE = 'trac'
PYTHON_STYLE = 'idle'
C_STYLE = DEFAULT_STYLE
JAVA_STYLE = DEFAULT_STYLE
PROCESSING_STYLE = 'default'
PYTHON3 = True
USE_PDF_TEMPLATE = True
DEBUG_PDFLATEX = False
XELATEX = False
DELETE_HLINES = True

HTML_COMMAND = 'pandoc -s -S -t html5 --email-obfuscation=none'
SLIDES_COMMAND = 'pandoc -s -S -t dzslides --no-highlight \
    --email-obfuscation=none -c dzslides.css'

if HTML_FOR_CANVAS:
    INLINE_STYLES = True

if ADD_CANVAS_CSS:
    HTML_COMMAND += ' -c canvas.css'

if MATHJAX:
    HTML_COMMAND += ' --mathjax'
    SLIDES_COMMAND += ' --mathjax'

PDF_COMMAND = 'pandoc -V fontsize=10pt -V margin=0.8in'


def pygmentize(filename, out_format):
    """
    Use pygments to highlight source code blocks in pandoc
    input file.  (If file is source code, put it in a fenced
    code block to make a pandoc input file, then use pygments
    to add highlighting.)

    :param filename: pandoc input file.
    :param out_format: html, slides (html for dzslides) or pdf.
```

```
:return: pandoc input with pygments' highlighting (as html
         tags) in fenced code blocks.
"""

pd = open(filename).read().split('\n')
ext = filename[filename.find('.') + 1:]

if ext != 'txt':
    pd = ['~~~' + ext] + pd + ['~~~']

style_defs_for_pdf = ''
pd_new = []
i = 0

while i < len(pd):
    s = pd[i].strip()

    if not s.startswith('~~~') or s == '~~~':
        pd_new.append(pd[i])
        i += 1

    # Block to highlight.
    else:
        indent = pd[i].find('~~~')

        # Set language, style.
        language = pd[i][indent + 3:].rstrip()

        if 'py' in language:
            style = PYTHON_STYLE

            if PYTHON3 and language == 'python':
                language = 'python3'

        elif language == 'c':
            style = C_STYLE
        elif language == 'java':
            style = JAVA_STYLE
        elif language == 'processing':
            style = PROCESSING_STYLE
        else:
            style = DEFAULT_STYLE

        # Set lexer.
        if PYTHON3 and language == 'pycon':
            lexer = pygments.lexers.PythonConsoleLexer(
                python3=True)
        elif language == 'processing':
            lexer = ProcessingLexer()
        else:
            lexer = pygments.lexers.get_lexer_by_name(
                language)

        # Set formatter for html / slides.  If not
        # INLINE_STYLES write css file and update
```

```
            # HTML_COMMAND / SLIDES_COMMAND.
            if out_format == 'html' or out_format == 'slides':
                if style == 'idle':
                    formatter = pygments.formatters.HtmlFormatter(
                        style=IdleStyle, noclasses=INLINE_STYLES)
                else:
                    formatter = pygments.formatters.HtmlFormatter(
                        style=style, noclasses=INLINE_STYLES)

                if not INLINE_STYLES:
                    css_filename = style + '.css'
                    css_file = open(css_filename, 'w')
                    print(formatter.get_style_defs(), file=css_file)
                    css_file.close()

                    if out_format == 'html':
                        global HTML_COMMAND

                        if not ('css' in HTML_COMMAND):
                            HTML_COMMAND += ' -c ' + css_filename
                    else:  # slides
                        global SLIDES_COMMAND

                        if not ('css' in SLIDES_COMMAND):
                            SLIDES_COMMAND += ' -c ' + css_filename

            # Set formatter for pdf.
            else:
                if style == 'idle':
                    formatter = pygments.formatters.LatexFormatter(
                        style=IdleStyle)
                else:
                    formatter = pygments.formatters.LatexFormatter(
                        style=style)

                style_defs_for_pdf = formatter.get_style_defs()

            # Highlight code block.
            cb = []
            i += 1

            while pd[i].strip() != '~~~':
                cb.append(pd[i][indent:])
                i += 1

            i += 1

            hb = pygments.highlight('\n'.join(cb), lexer, formatter)

            for line in hb.split('\n'):
                pd_new.append(' ' * indent + line)

    return '\n'.join(pd_new), style_defs_for_pdf


def run_pandoc(in_filename, out_filename, out_format,
```

```python
                canvas_fixes, style_defs_for_pdf):

    if out_format == 'slides':
        subprocess.call(SLIDES_COMMAND.split() +
                        [in_filename, '-o', out_filename])
        os.remove(in_filename)

    elif out_format == 'html':
        subprocess.call(HTML_COMMAND.split() +
                        [in_filename, '-o', out_filename])
        os.remove(in_filename)
        html = open(out_filename).read()

        if canvas_fixes:
            # Get rid of code tags.
            html = html.replace(
                '<code>', '<span style="font-family:monospace;">')
            html = html.replace('</code>', '</span>')

            # Align table cell contents vertically to match
            # latex tables.
            html = html.replace(
                '<td style="text-align: left;">',
                '<td style="text-align: left; vertical-align: top">')
            html = html.replace(
                '<td style="text-align: center;">',
                '<td style="text-align: center; vertical-align: top">')
            html = html.replace(
                '<td style="text-align: right;">',
                '<td style="text-align: right; vertical-align: top">')

            # More little fixes.
            html = html.replace('h3', 'h4')
            html = html.replace(
                '<code>', '<span style="font-family:monospace;">')
            html = html.replace('<code class="url">', '<span>')
            html = html.replace('</code>', '</span>')
            html = html.replace(
                '<pre style="line-height: 125%">',
                '<pre style="font-family:monospace;">')
            html = html.replace('</table>', '</table>&nbsp;')

        temp_file = open('temp', 'w')
        print(html, file=temp_file)
        temp_file.close()
        shutil.copy('temp', out_filename)
        os.remove('temp')

    else:  # pdf
        subprocess.call(PDF_COMMAND.split() +
                        [in_filename, '-o', 'temp.tex'])
        os.remove(in_filename)
        # tex = open('temp.tex', encoding='utf-8').read()
        tex = open('temp.tex').read()
        os.remove('temp.tex')
        if DELETE_HLINES:
            # Get rid of horizontal lines at the beginning and
            # at the end of the table (to make latex version
            # look like html version).

            # For older pandoc version (Debian)...
            tex = tex.replace('\FL', '')
            tex = tex.replace('\LL', '')

            # For newer pandoc version (OS X)...
            tex = tex.replace('\\hline', '')

            # For still newer pandoc version (Arch)...
            tex = tex.replace('{longtable}[c]', '{longtable}[l]')
            tex = tex.replace('\\toprule', '')
            tex = tex.replace('\\bottomrule', '')

        if style_defs_for_pdf != '':
            i = tex.find('\\begin{document}')
            tex = (tex[:i] + '\\usepackage{fancyvrb}' +
                   '\\usepackage{color}' + style_defs_for_pdf
                   + tex[i:])

        # tex_file = open('temp.tex', 'w', encoding='utf-8')
        tex_file = open('temp.tex', 'w')
        print(tex, file=tex_file)
        tex_file.close()

        if XELATEX:
            c = 'xelatex'
        else:
            c = 'pdflatex'

        if DEBUG_PDFLATEX:
            subprocess.call([c, 'temp.tex'])
        else:
            so_file = open('temp', 'w')
            subprocess.call([c, 'temp.tex'], stdout=so_file)
            so_file.close()
            os.remove('temp')
            os.remove('temp.tex')

        shutil.move('temp.pdf', out_filename)
        os.remove('temp.aux')
        os.remove('temp.log')
        os.remove('temp.out')


def main():
    out_format = DEFAULT_OUTPUT
    processed_args = []
    filename = ''

    if USE_PDF_TEMPLATE:
        global PDF_COMMAND
        PDF_COMMAND += " --template=" + \
            os.path.abspath(os.path.dirname(sys.argv[0])) + \
            "/pdpm.latex"

    for a in sys.argv[1:]:
        if a.startswith('--'):
            out_format = a[2:]
            processed_args.append(a)
        else:
            filename = a

    temp_file = open('temp', 'w')
    pg, style_defs_for_pdf = pygmentize(filename, out_format)
    print(pg, file=temp_file)
    temp_file.close()

    if out_format == 'slides':
        ext = 'html'
    else:
        ext = out_format

    out_filename = filename[:filename.rfind('.') + 1] + ext
    run_pandoc('temp', out_filename, out_format,
               HTML_FOR_CANVAS, style_defs_for_pdf)

if __name__ == '__main__':
    main()
```