```
#!/usr/bin/env python
# -*- coding:latin-1 -*-

# TODO:
# - allow selection of encoding
# - allow selection of lexer
# - list encodings
# - list lexers


import time
import os
import getpass

username = getpass.getuser()
pagesize = (595.28, 841.89) # A4


class Font:
    def __init__(self, subtype, basefont, widths):
        self.subtype = subtype
        self.basefont = basefont
        self.widths = widths

    def __repr__(self):
        return '<%s>' % self.basefont

    def measure(self, s, size):
        charwidths = self.widths
        w = 0
        for c in s:
            try:
                w += charwidths[c]
            except:
                w += charwidths['m'] # XXX HACK
        return w*size/1000.0

    def split(self, s, fontsize, width0, width1):
        # Splits s such that the size of the resulting pieces is less
        # than *width*.
        #
        # NOTE: if width is smaller than the first character things
        # are complicated: if we are at the first line, we split
        # before the character, otherwise behind it.
        maxw = width0
        charwidths = self.widths
        r = [(0, '')]
        w = 0
        for c in s:
            try:
                _w = charwidths[c]
            except:
                _w = charwidths['m'] # XXX HACK
            _w *= fontsize/1000.0
            if w+_w > maxw:
                r.append((0, ''))
                w = 0
                maxw = width1
            a = r[-1]
            r[-1] = a[0]+_w, a[1]+c
            w += _w
```



```
        return r

_widths = {unichr(i) : 600 for i in range(256)}
COURIER = Font("Type1", "Courier", _widths)
COURIER_BOLD = Font("Type1", "Courier-Bold", _widths)
COURIER_OBLIQUE = Font("Type1", "Courier-Oblique", _widths)

l = 278,278,278,278,278,278,278,278,278,278,278,278,278,278,278,278,278,278,\
    278,278,278,278,278,278,278,278,278,278,278,278,278,278,355,556,\
    556,889,667,191,333,333,389,584,278,333,278,278,556,556,556,556,556,556,\
    556,556,556,556,278,278,584,584,584,556,1015,667,667,722,722,667,611,778,\
    722,278,500,667,556,833,722,778,667,778,722,667,611,722,667,944,667,667,\
    611,278,278,278,469,556,333,556,556,500,556,556,278,556,556,222,222,500,\
    222,833,556,556,556,556,333,500,278,556,500,722,500,500,500,334,260,334,\
    584,350,556,350,222,556,333,1000,556,556,333,1000,667,333,1000,350,611,\
    350,350,222,222,333,333,350,556,1000,333,1000,500,333,944,350,500,667,278,\
    333,556,556,556,556,260,556,333,737,370,556,584,333,737,333,400,584,333,\
    333,333,556,537,278,333,333,365,556,834,834,834,611,667,667,667,667,667,\
    667,1000,722,667,667,667,667,278,278,278,278,722,722,778,778,778,778,778,\
    584,778,722,722,722,722,667,667,611,556,556,556,556,556,556,889,500,556,\
    556,556,556,278,278,278,278,556,556,556,556,556,556,556,556,584,611,556,556,\
    556,556,500,556,500
_widths = {unichr(i) : l[i] for i in range(256)}
HELVETICA = Font("Type1", "Helvetica", _widths)

l = 278,278,278,278,278,278,278,278,278,278,278,278,278,278,278,278,278,278,\
    278,278,278,278,278,278,278,278,278,278,278,278,278,278,333,474,556,\
    556,889,722,238,333,333,389,584,278,333,278,278,556,556,556,556,556,556,\
    556,556,556,556,333,333,584,584,584,611,975,722,722,722,722,667,611,778,\
    722,278,556,722,611,833,722,778,667,778,722,667,611,722,667,944,667,667,\
    611,333,278,333,584,556,333,556,611,556,611,556,333,611,611,278,278,556,\
    278,889,611,611,611,611,389,556,333,611,556,778,556,556,500,389,280,389,\
    584,350,556,350,278,556,500,1000,556,556,333,1000,667,333,1000,350,611,\
    350,350,278,278,500,500,350,556,1000,333,1000,556,333,944,350,500,667,\
    278,333,556,556,556,556,280,556,333,737,370,556,584,333,737,333,400,584,\
    333,333,333,611,556,278,333,333,365,556,834,834,834,611,722,722,722,722,\
    722,722,1000,722,667,667,667,667,278,278,278,278,722,722,778,778,778,778,\
    778,584,778,722,722,722,722,667,667,611,556,556,556,556,556,556,889,556,\
    556,556,556,278,278,278,278,611,611,611,611,611,611,611,584,611,611,\
    611,611,611,556,611,556

_widths = {unichr(i) : l[i] for i in range(256)}
HELVETICA_BOLD = Font("Type1", "Helvetica-Bold", _widths)

l = 250,250,250,250,250,250,250,250,250,250,250,250,250,250,250,250,250,250,\
    250,250,250,250,250,250,250,250,250,250,250,250,250,250,333,408,500,\
    500,833,778,180,333,333,500,564,250,333,250,278,500,500,500,500,500,500,\
    500,500,500,500,278,278,564,564,564,444,921,722,667,667,722,611,556,722,\
    722,333,389,722,611,889,722,722,556,722,667,556,611,722,722,944,722,722,\
    611,333,278,333,469,500,333,444,500,444,500,444,333,500,500,278,278,500,\
    278,778,500,500,500,500,333,389,278,500,500,722,500,500,444,480,200,480,\
    541,350,500,350,333,500,444,1000,500,500,333,1000,556,333,889,350,611,350,\
    350,333,333,444,444,350,500,1000,333,980,389,333,722,350,444,722,250,333,\
    500,500,500,500,200,500,333,760,276,500,564,333,760,333,400,564,300,300,\
    333,500,453,250,333,300,310,500,750,750,750,444,722,722,722,722,722,722,\
    889,667,611,611,611,611,333,333,333,333,722,722,722,722,722,722,564,\
    722,722,722,722,722,722,556,500,444,444,444,444,444,444,667,444,444,444,\
    444,444,278,278,278,278,500,500,500,500,500,500,500,564,500,500,500,500,\
    500,500,500,500
_widths = {unichr(i) : l[i] for i in range(256)}
TIMES = Font("Type1", "Times", _widths)
```

```
def build_lines(pieces, max_width):
    # A piece is a tuple (string, font)
    line = []
    width = 0
    for text, (font, fontsize) in pieces:
        add_nl = False
        for s in text.split('\n'):
            if add_nl:
                yield line
                width = 0
                line = []
            else:
                add_nl = True

            first = True
            for w, ss in font.split(s, fontsize, max_width-width, max_width):
                if first:
                    first = False
                else:
                    assert width <= max_width
                    yield line
                    width = 0
                    line = []
                if ss:
                    line.append((ss, (font, fontsize)))
                    width += w


def ngroup(l, n):
    # Split l into tuples of size n
    r = []
    for x in l:
        r.append(x)
        if len(r) >= n:
            yield r
            r = []
    if r:
        yield r


def compute_pieces(filename, fontsize):
    from pygments.formatter import Formatter
    from pygments.lexers import guess_lexer_for_filename, get_lexer_by_name
    from pygments import token as Token
    from pygments import highlight

    style = {
        Token.Keyword : (COURIER_BOLD, fontsize),
        Token.Name.Builtin.Pseudo : (COURIER_BOLD, fontsize),
        Token.Comment.Single : (COURIER_OBLIQUE, fontsize),
        Token.Comment : (COURIER_OBLIQUE, fontsize),
        Token.Literal.String : (COURIER_OBLIQUE, fontsize),
        Token.Literal.String.Single : (COURIER_OBLIQUE, fontsize),
        Token.Literal.String.Double : (COURIER_OBLIQUE, fontsize),
        Token.Literal.String.Escape : (COURIER_OBLIQUE, fontsize),
        Token.Literal.String.Interpol : (COURIER_OBLIQUE, fontsize),
        None : (COURIER, fontsize), # default
    }
```



```
    class FontFormatter(Formatter):
        def format(self, tokensource, outfile, style=style):
            last = None
            l = []
            for token, text in tokensource:
                try:
                    fontinfo = style[token]
                except KeyError:
                    fontinfo = style[None]
                if fontinfo == last:
                    l[-1][0] += text
                else:
                    l.append([text, fontinfo])
                    last = fontinfo
            self.pieces = l

    formatter = FontFormatter()
    src = open(filename, 'r').read()
    lexer = guess_lexer_for_filename(filename, src)
    highlight(src, lexer, formatter)
    return formatter.pieces


def make_ref(refs):
    r = len(refs)+1, 0
    refs.add(r)
    return r


def quote(s):
    return s.replace('\\','\\\\').replace(')','\\)').replace('(','\\(').\
        replace('\r','\\r')


def make_s(obj):
    # Create string representation for obj
    if type(obj) == str:
        return obj
    if type(obj) == unicode:
        return obj.encode('latin-1') # XXX add ignore-flag?
    if type(obj) == tuple:
        # we always interpret tuples as references!
        return '%i %i R' % obj
    if type(obj) == list:
        # we always interpret lists as lists!
        l = [make_s(child) for child in obj]
        return '['+(' '.join(l))+']'
    if type(obj) == dict:
        # we always interpret dicts as dicts!
        l = ['/'+make_s(k)+' '+make_s(v) for (k, v) in obj.items()]
        return '<<\n'+(' '.join(l))+' >>\n'
    if type(obj) in (int, float):
        return str(obj)
    raise Exception("Wrong type: %s" % repr(obj))


def shrink(rect, d):
    # Shrink rectangle rect by amount d
    x, y, w, h = rect
    return (x+d, y+d, w-2*d, h-2*d)
```

```
def grow(rect, d):
    # Grow rectangle rect by amount d
    return shrink(rect, -d)


def create_pdf(infilename, outfilename, twoup, fontsize):
    pieces = compute_pieces(infilename, fontsize)
    f = open(outfilename, 'wb')
    t = time.localtime(os.path.getmtime(infilename))
    file_date = time.strftime("%b %d, %y %H:%M", t)
    date = time.strftime("%b %d, %y %H:%M")

    def out(obj, f=f):
        s = make_s(obj)
        f.write(s)

    def centered(x, y, text, font, size):
        w = font.measure(text, size)
        out("BT /%s %i Tf %i %i Td (%s)Tj ET\n" %
            (fontnames[font], size, x-0.5*w, y, text))

    def right_aligned(x, y, text, font, size):
        w = font.measure(text, size)
        out("BT /%s %i Tf %i %i Td (%s)Tj ET\n" %
            (fontnames[font], size, x-w, y, text))


    fontnames = {
        COURIER : 'F1',
        COURIER_BOLD : 'F2',
        TIMES : 'F3',
        COURIER_OBLIQUE : 'F4',
        HELVETICA : 'F5',
        HELVETICA_BOLD : 'F6',
    }
    xref = {} #  of tuples (number, id, position)
    pages = [] # list of content ids
    refs = set() # used references

    # geometry
    if twoup:
        ph, pw = pagesize
        bl = 25
        br = 5
        bt = 60
        bb = 40
        pwh = 0.5*pw
        w = pwh-bl-br
        left_frame = bl, bb, w, ph-bt-bb
        right_frame = pwh+br, bb, w, ph-bt-bb
        outer_frame = bl, bb, pwh+br+w-bl, ph-bt-bb
        lines = build_lines(pieces, left_frame[2])
        lines_per_frame = int(left_frame[3] / fontsize)
    else:
        pw, ph = pagesize
        bl = 40
        br = 40
        bt = 80
        bb = 40
        frame = bb, bl, pw-bl-br, ph-bt-bb
```



```
        outer_frame = frame
        lines_per_frame = int(frame[3] / fontsize)
        lines = build_lines(pieces, frame[2])

    # header
    out("%PDF-1.3\n%\xC7\xEC\x8F\xA2\n")

    # object 1: catalog
    refs.add((1, 0))
    xref[(1, 0)] = f.tell()
    out("1 0 obj\n  << /Type /Catalog\n /Pages 2 0 R >> \nendobj\n\n")

    # object 2: root pages - we write it later!
    refs.add((2, 0))

    # writing the pages
    pagerefs = []
    frame_groups = tuple(ngroup(lines, lines_per_frame))
    if twoup:
        frames_per_page = 2
    else:
        frames_per_page = 1

    page_groups = tuple(ngroup(frame_groups, frames_per_page))
    nframes = len(frame_groups)
    npages = len(page_groups)
    ngroups = nframes

    iframe = 0
    for i, page_group in enumerate(page_groups):
        cref = make_ref(refs)
        xref[cref] = f.tell()

        out("%i %i obj\n" % cref)
        lref = make_ref(refs)

        out("<< /Length %i % i R>>\n" % lref)
        out('stream\n')
        pos = f.tell()

        out("q\n") # save state
        if twoup:
            out("0 1 -1 0 %s 0 cm\n" % pagesize[0])
        d = 9
        x, y = outer_frame[:2]
        out("BT /F5 %i Tf %i %i Td (%s)Tj ET\n" % \
            (d, x, y-d-2, 'Printed by '+username))
        x += outer_frame[2]
        right_aligned(x, y-d-2, date, HELVETICA, d)


        for j, frame_group in enumerate(page_group):
            iframe += 1
            if twoup:
                if j == 0:
                    frame = left_frame
                else:
                    frame = right_frame

            if 0:
                print frame
```

```
            out("%i %i %i %i re\nS\n" % frame)
            out("%i %i %i %i re\nS\n" % (frame[:2]+(10, 10)))

        r = grow(frame, 2)
        out("q\n") # save state
        out("%i %i %i %i re\nS\n" % r)
        x, y, w, h = r
        d = 9 # font size header
        out("0.95 0.95 0.95 rg\n")
        out("%i %i %i %i re\nf\n" % (x, y+h, w, 1.9*d))
        out("Q\n") # restore state
        out("%i %i %i %i re\nS\n" % (x, y+h, w, 1.9*d))
        out("BT /F5 %i Tf %i %i Td (%s)Tj ET\n" % \
            (d, x+d, y+h+0.5*d, file_date))
        centered(x+0.5*w, y+h+0.5*d, infilename, HELVETICA_BOLD, d+3)

        s = "Page %i/%i" % (iframe, ngroups)
        right_aligned(x+w-d, y+h+0.5*d, s, HELVETICA, d)

        out("BT\n")
        x0, y0 = frame[:2]
        y0 += frame[3]-fontsize
        out("%s %s Td\n" % (x0, y0))
        out("%i TL\n" % fontsize)

        lastfontinfo = None
        for l in frame_group:
            #print l
            for s, fontinfo in l:
                if fontinfo != lastfontinfo:
                    font, size = fontinfo
                    fontname = fontnames[font]
                    out("/%s %s Tf\n" % (fontname, size))
                    lastfontinfo = fontinfo
                out("(%s)Tj\n" % quote(s).encode('latin-1'))
            out("()'\n")
        out("ET\n")
    out("Q\n") # restore state

    length = f.tell()-pos
    out("\nendstream\n")
    out("endobj\n")

    xref[lref] = f.tell()
    out("%i %i obj\n" % lref)
    out("%i\n" % length)
    out("endobj\n")

    # directly after the content: write the page
    pref = make_ref(refs)
    pagerefs.append(pref)

    xref[pref] = f.tell()
    out("%i %i obj\n" % pref)
    if twoup:
        rotate = "/Rotate 90\n"
    else:
        rotate = ""

    out("<< /Parent 2 0 R /Resources << /Font << ")
    for font, key in sorted(fontnames.items(), key=lambda x:x[1]) :
```



```
            out("""       /%s << /Encoding /WinAnsiEncoding /Type /Font
               /BaseFont /%s
               /Subtype /Type1 >>""" % (key, font.basefont))
        out("""        >>
      >>
      /MediaBox [ 0 0 595.28 841.89 ]
      %s
      /Type /Page /ProcSet [/PDF /Text /ImageB /ImageC /ImageI]
      /Contents %i %i R
    >>
endobj\n""" % ((rotate,)+cref))

    # finally writing the root pages (object 2)
    kids = ["%i %i R" % ref for ref in pagerefs]
    s = " ".join(kids)

    xref[(2, 0)] = f.tell()
    out("2 0 obj\n")
    w, h = pagesize
    out(dict(
        Type='/Pages', Kids=kids, Count=len(kids),
        MediaBox=[0, 0, pagesize[0], pagesize[1]]))
    out("endobj\n\n")

    # write the xref table
    startxref = f.tell()
    out('xref\n')
    out('0 %i\n' % (len(xref)+1))
    out('%010i' % 0); out(' 65535'); out(' f\n')
    for ref in sorted(xref.keys()):
        pos = xref[ref]
        version = ref[1]
        out('%010i' % pos); out(' %05i' % version); out(' n\n')

    # write trailer
    out('trailer\n')
    out(dict(Root='1 0 R', Size=len(xref)+1))
    out('startxref\n')
    out('%i\n' % startxref)
    out('%%EOF\n')


def startfile(filename):
    import sys, subprocess
    if sys.platform == "win32":
        os.startfile(filename)
    else:
        opener ="open" if sys.platform == "darwin" else "xdg-open"
        subprocess.call([opener, filename])

from entrypoint2 import entrypoint
@entrypoint
def main(srcfile, outfile=None, dont_view=False, single_page=False,
         fontsize=None):
    twoup = not single_page

    if fontsize is None:
        if twoup:
            fontsize = 8
        else:
            fontsize = 10
```

```
    if outfile is None:
        base, ext = os.path.splitext(srcfile)
        outfile = base+'.pdf'

    create_pdf(srcfile, outfile, twoup, fontsize)
    if not dont_view:
        startfile(outfile)
```