```
#!/usr/bin/env python

"""demo.py - Demo script for py2pdf 0.5.

The main idea is: take one Python file and make a whole
bunch of PDFs out of it for test purposes.

Dinu Gherman
"""


import string, re, os, os.path, sys, shutil
from py2pdf import *


### Custom layouter class used with test3().

class ImgPDFLayouter (PythonPDFLayouter):
    "A custom layouter displaying an image on each page."

    def setMainFrame(self, frame=None):
        "Make a frame in the right half of the page."

        width, height = self.options.realPaperFormat.size
        self.frame = height - 2*cm, 2*cm, 250, width-1*cm
        self.makeForm()


    def makeForm(self):
        "Use the experimental ReportLab form support."

        width, height = self.options.realPaperFormat.size
        tm, bm, lm, rm = self.frame
        c = self.canvas

        # Define a PDF form containing an image frame
        # that will be included on every page, but
        # stored only once in the resulting file.
        c.beginForm0("imageFrame")
        c.saveState()
        x, y = 219.0, 655.0 # Known size of the picture.
        c.scale((lm - 1*cm)/x, height/y)
        path = 'vertpython.jpg'
        c.drawInlineImage(path, 0, 0)
        c.restoreState()
        c.endForm0()


    def putPageDecoration(self):
        "Draw the left border image and page number."

        width, height = self.options.realPaperFormat.size
        tm, bm, lm, rm = self.frame
        c = self.canvas

        # Footer.
        x, y = lm + 0.5 * (rm - lm), 0.5 * bm
        c.setFillColor(Color(0, 0, 0))
        c.setFont('Times-Italic', 12)
        label = "Page %d" % self.pageNum
        c.drawCentredString(x, y, label)

        # Call the previously stored form.
        c.doForm0("imageFrame")


### Helpers.

def modifyPath(path, new, ext='.py'):
    "Modifying the base name of a file."

    rest, ext = os.path.splitext(path)
    path, base = os.path.split(rest)
    format = "%s-%s%s" % (base, new, ext)
    return os.path.join(path, format)


def getAllTestFunctions():
    "Return a list of all test functions available."

    globs = globals().keys()
    tests = filter(lambda g: re.match('test[\d]+', g), globs)
    tests.sort()
    return map(lambda t: globals()[t], tests)


### Test functions.
###
### In order to be automatically found and applied to
### a Python file all test functions must follow the
### following naming pattern: 'test[0-9]+' and contain
### a doc string.

def test0(path):
    "Creating a PDF assuming an ASCII file."

    p = PDFPrinter()
    p.process(path)
```

```
def test1(path):
    "Creating a PDF using only default options."

    p = PythonPDFPrinter()
    p.process(path)


def test2(path):
    "Creating a PDF with some modified options."

    p = PythonPDFPrinter()
    p.options.updateOption('landscape', 1)
    p.options.updateOption('fontName', 'Helvetica')
    p.options.updateOption('fontSize', '14')
    p.options.display()
    p.process(path)


def test3(path):
    "Creating several PDFs as 'magazine listings'."

    p = PythonPDFPrinter()
    p.Layouter = EmptyPythonPDFLayouter
    p.options.updateOption('paperSize', '(250,400)')
    p.options.updateOption('multiPage', 1)
    p.options.updateOption('lineNum', 1)
    p.process(path)


def test4(path):
    "Creating a PDF in monochrome mode."

    p = PythonPDFPrinter()
    p.options.updateOption('mode', 'mono')
    p.process(path)


def test5(path):
    "Creating a PDF with options from a config file."

    p = PythonPDFPrinter()
    i = string.find(path, 'test5')
    newPath = modifyPath(path[:i-1], 'config') + '.txt'

    try:
        p.options.updateWithContentsOfFile(newPath)
        p.options.display()
        p.process(path)
    except IOError:
        print "Skipping test5() due to IOError."


def test6(path):
    "Creating a PDF with modified layout."

    p = PythonPDFPrinter()
    p.Layouter = ImgPDFLayouter
    p.options.updateOption('fontName', 'Helvetica')
    p.options.updateOption('fontSize', '12')
    p.options.display()
    p.process(path)


### Main.

def main(inPath, *tests):
    "Apply various tests to one Python source file."

    for t in tests:
        newPath = modifyPath(inPath, t.__name__)
        shutil.copyfile(inPath, newPath)
        print t.__doc__
        t(newPath)
        os.remove(newPath)
        print


if __name__=='__main__':
    # Usage: "python demo.py <file> <test1> [<test2> ...]"
    try:
        try:
            tests = map(lambda a: globals()[a], sys.argv[2:])
        except IndexError:
            tests = getAllTestFunctions()

        fileName = sys.argv[1]
        apply(main, [fileName]+tests)

    # Usage: "python demo.py" (implicitly does this:
    # "python demo.py demo.py" <allTestsAvailable>)
    except IndexError:
        print "Performing self-test..."
        fileName = sys.argv[0]
        tests = getAllTestFunctions()
        apply(main, [fileName]+tests)
```