PDFgen graphics API test script

Hello World
This tests a low-level API to the PDF file formac. It is intended to sit
underneath PIDDLE, and to closely mirror the PDF / Postscript imaging
model. There is an almost one to one correspondence between commands
and PDF operators. However, where PDF provides several ways to do a job,
we have generally only picked one.

The test script attempts to use all of the methods exposed by pdfgen.PDFEngine.

First, let's look at test output. Here are the basic commands:

canvas.enterTextMode() must be called before text operations
canvas.exitTextMode() must be called after text operations
You can do graphics in between calls, but no coordinate transforms
or clipping.

canvas.setTextOrigin(x, y) sets the text origin
canvas.getCursor() returns the current text cursor

canvas.textOut(text) writes text, and moves the cursor to the righc.
canvas.textLine(text) writes text, and moves the cursor down 'leading'.
This means textLine() is faster - no need to do a stringWidth!

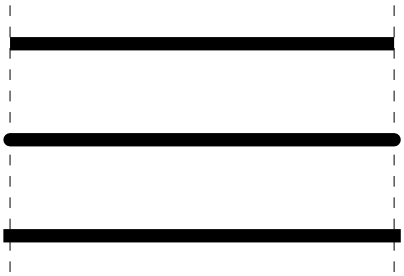
canvas.textLines(stuff) accepts a multi-line string or a list/tuple
of strings, and moves the cursor down the page.

The green crosshairs test whether the text cursor is tracking correctly.
textOut moves across:textOut moves across:textOut moves across:
textLine moves down
textLine moves down
textLine moves down

This is a multi-line
string with embedded newlines
drawn with textLines().
This is a list of strings
drawn with textLines().

Small text.Bigger fixed width text.Small text again.

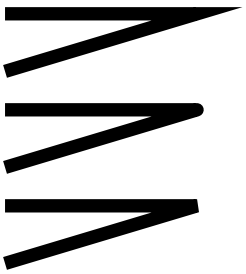
Line Drawing Styles



the default - butt caps project half a width

round caps

square caps



Default - mitered join

round join

bevel join

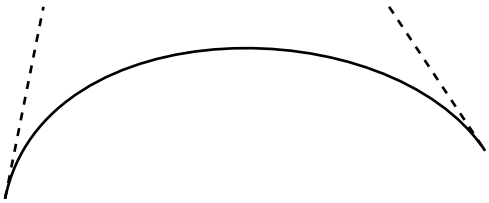
dash pattern 6 points on, 3 off- setDash(6,3)

dash pattern lengths growing - setDash([1,2,3,4,5,6],0)

Shape Drawing Routines

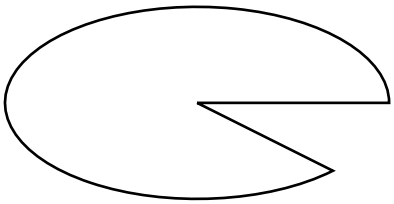
Rather than making your own paths, you have access to a range of shape routines. These are built in pdfgen out of lines and bezier curves, but use the most compact set of operators possible. We can add any new ones that are of general use at no cost to performance.

canvas.line(x1, y1, x2, y2)

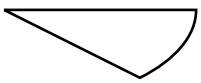


canvas.bezier(x1, y1, x2, y2, x3, y3, x4, y4)

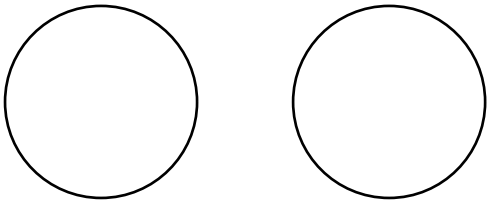
canvas.rect(x, y, width, height) - x,y is lower left



canvas.wedge(x1, y1, x2, y2, startDeg, extentDeg)
Note that this is an elliptical arc, not just circular!



Use a negative extent to go clockwise



canvas.circle(x, y, radius)

Font Control

Listing available fonts...

This should be Courier
This should be Courier-Bold
This should be Courier-BoldOblique
This should be Courier-Oblique
This should be Helvetica
This should be Helvetica-Bold
This should be Helvetica-BoldOblique
This should be Helvetica-Oblique
Τηισ σηουλδ βε Σψμβολ
This should be Times-Bold
This should be Times-BoldItalic
This should be Times-Italic
This should be Times-Roman
✶❋❉▲ ▲❋❑◆●❄ ✲❄ ❋❁❒❄✿❉■✳✲❁▼▲

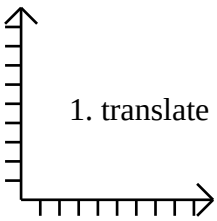
Now we'll look at the color functions and how they interact with the text. In theory, a word is just a shape; so setFillColorRGB() determines most of what you see. If you specify other text rendering modes, an outline color could be defined by setStrokeColorRGB() too

Green fill, no stroke

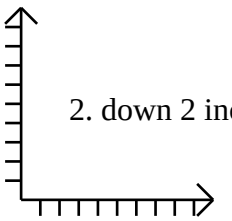
Green fill, red stroke - yuk!

Coordinate Transforms

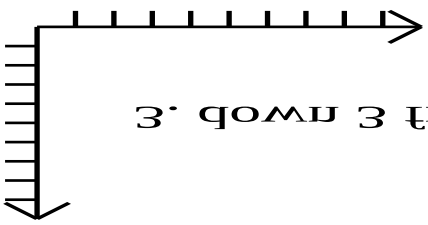
This shows coordinate transformations. We draw a set of axes, moving down the page and transforming space before each one. You can use saveState() and restoreState() to unroll transformations. Note that functions which track the text cursor give the cursor position in the current coordinate system; so if you set up a 6 inch high frame 2 inches down the page to draw text in, and move the origin to its top left, you should stop writing text after six inches and not eight.



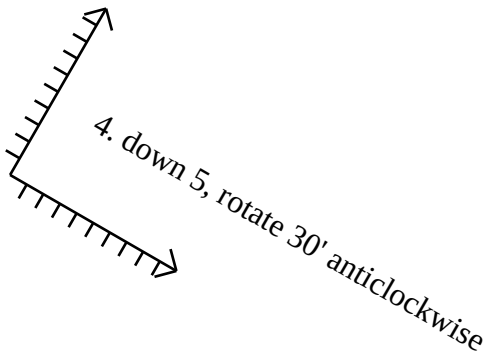
1. translate near top of page



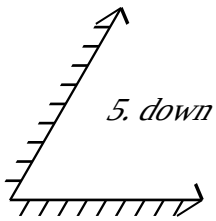
2. down 2 inches, across 1



3. down 3 from top, scale (3, -1)



4. down 5, rotate 30' anticlockwise

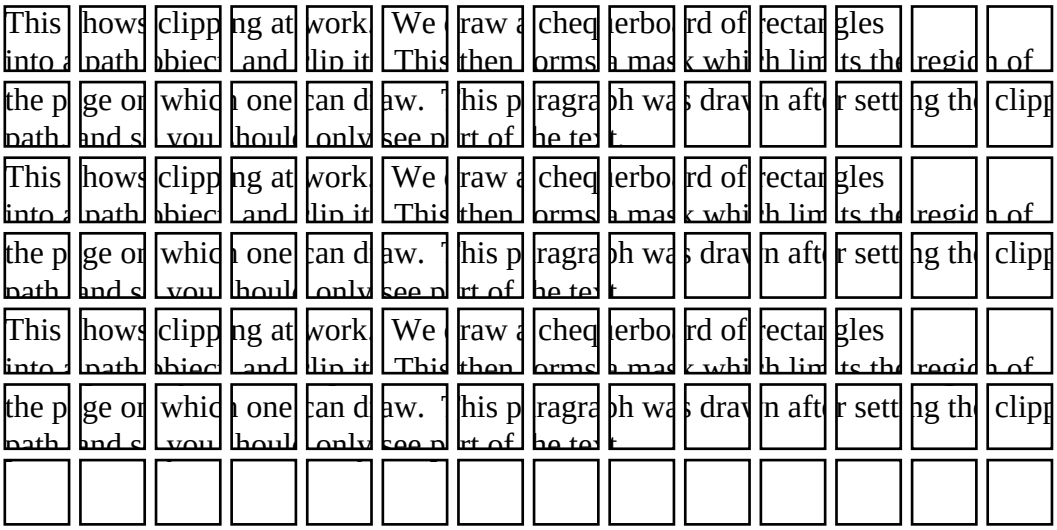


5. down 5, 3 across, skew beta 30

0. at origin

Clipping

This shows clipping at work. We draw a chequerboard of rectangles into a path object, and clip it. This then forms a mask which limits the region of the page on which one can draw. This paragraph was drawn after setting the clipping path, and so you should only see part of the text.



You can also use text as an outline for clipping with the text render mode. The API is not particularly clean on this and one has to follow the right sequence; this can be optimized shortly.

Images

This shows image capabilities. If I've done things
right, the bitmap should have its bottom left corner aligned
with the crosshairs.
PDFgen uses the Python Imaging Library to process
a very wide variety of image formats. Although some processing
is required, cached versions of the image are prepared and
stored in the project directory, so that subsequent builds of
an image-rich document are very fast indeed.

image drawn at natural size

image distorted to fit box