PDFgen graphics API test script

The ReportLab library permits you to create PDF documents directly from your Python code. The "pdfgen" subpackage is the lowest level exposed to the user and lets you directly position test and graphics on the page, with access to almost the full range of PDF features.
The API is intended 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 the Canvas class, defined in reportlab/pdfgen/canvas.py
First, let's look at text output. There are some basic commands to draw strings:

- canvas.setFont(fontname, fontsize [, leading])
- canvas.drawString(x, y, text)
- canvas.drawRightString(x, y, text)
- canvas.drawCentredString(x, y, text)

The coordinates are in points starting at the bottom left corner of the page. When setting a font, the leading (i.e. inter-line spacing) defaults to 1.2 * fontsize if the fontsize is not provided.

For more sophisticated operations, you can create a Text Object, defined in reportlab/pdfgen/testobject.py. Text objects produce tighter PDF, run faster and have many methods for precise control of spacing and position.
Basic usage goes as follows:

- tx = canvas.beginText(x, y)
- tx.textOut('Hello') # this moves the cursor to the right
- tx.textLine('Hello again') # prints a line and moves down
- y = tx.getY() # getX, getY and getCursor track position
- canvas.drawText(tx) # all gets drawn at the end

The green crosshairs below test whether the text cursor is working properly. They should appear at the bottom left of each relevant substring.

Testing decimal alignment
- aim for silver line

1,234,567.89
3,456.789
123
(7,192,302.30)

textOut moves across:textOut moves across:textOut moves across:
textLine moves down
textLine moves down
textLine moves down

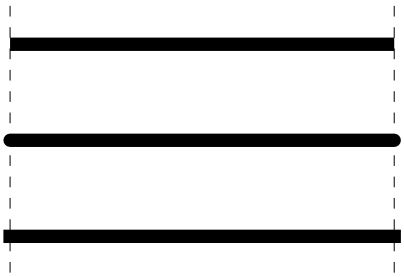
This is a multi-line
string with embedded newlines
drawn with textLines().
This is a list of strings
drawn with textLines().

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

copyright© trademark™ registered® scissors✂: ReportLab in unicode!

copyright© trademark™ registered® scissors✂: ReportLab in utf8!

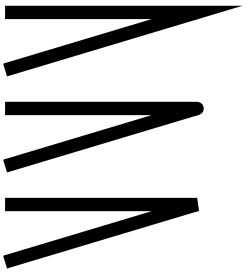
Line Drawing Styles



the default - butt caps project half a width

round caps

square caps



Default - mitered join

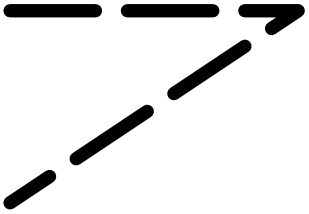
round join

bevel join

dash 6 points on, 6 off- setDash(6,6) setLineCap(0)

dash 6 points on, 6 off- setDash(6,6) setLineCap(1)

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

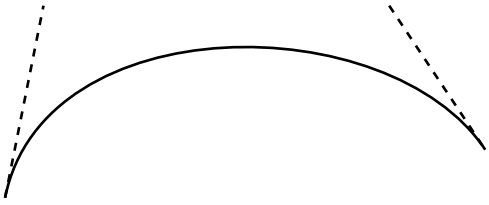


dash pattern, join and cap style interacting - round join & miter results in sausages

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)

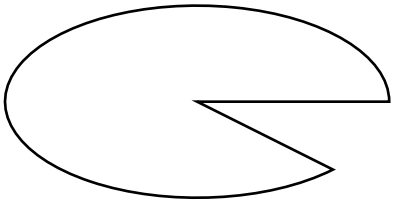


canvas.bezier(x1, y1, x2, y2, x3, y3, x4, y4)

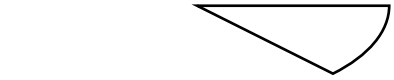
canvas.rect(x, y, width, height) - x,y is lower left

canvas.roundRect(x,y,width,height,radius)

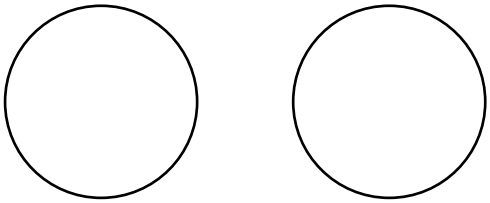
canvas.arc(x1, y1, x2, y2, startDeg, extentDeg)
Note that this is an elliptical arc, not just circular!



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

■■■■ ■■■■■■■ ■■ ■■■■■■■

This should be Times-Bold

This should be Times-BoldItalic

This should be Times-Italic

This should be Times-Roman

■■■■ ■■■■■■■ ■■ ■■■■■■■■■■■■■■■

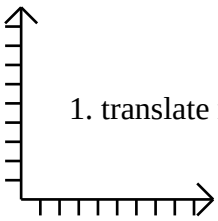
Now we'll look at the color functions and how they interact with the text. In theory, a word is just a shape; so setFillColorRGB() determines most of what you see. If you specify other text rendering modes, an outline color could be defined by setStrokeColorRGB() too

Green fill, no stroke

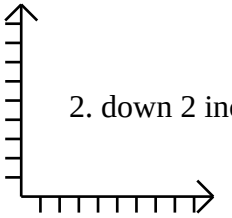
Green fill, red stroke - yuk!

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.

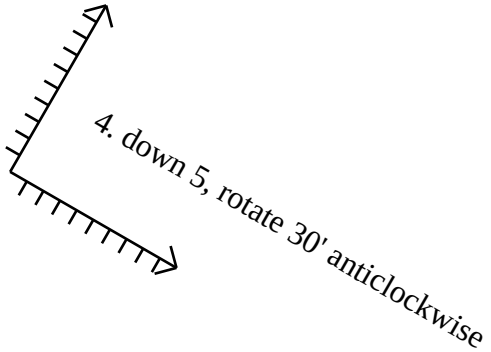


1. translate near top of page

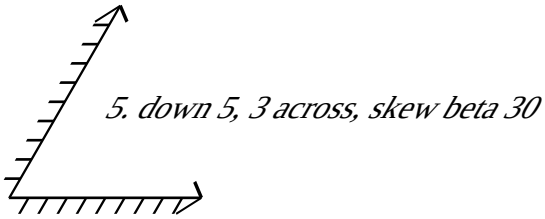


2. down 2 inches, across 1

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



4. down 5, rotate 30' anticlockwise

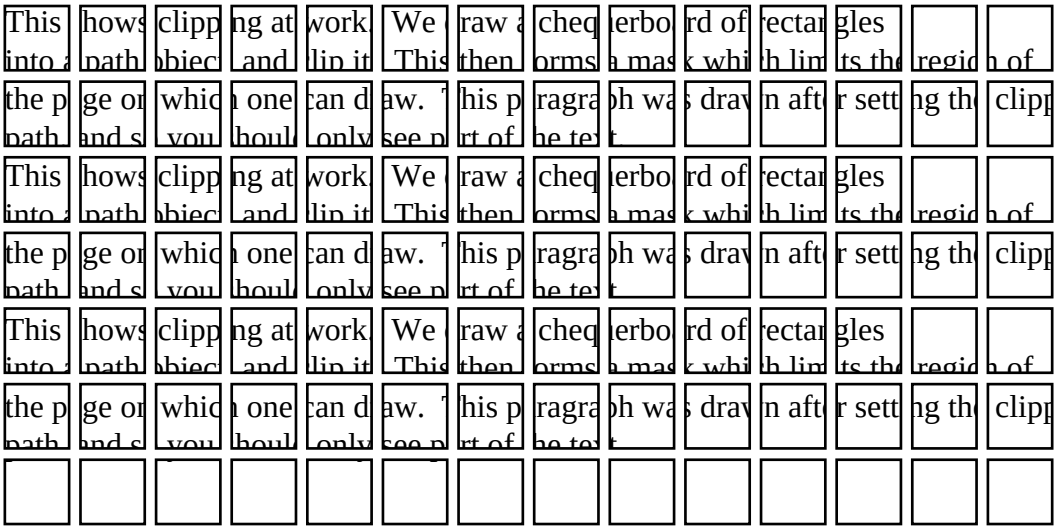


5. down 5, 3 across, skew beta 30

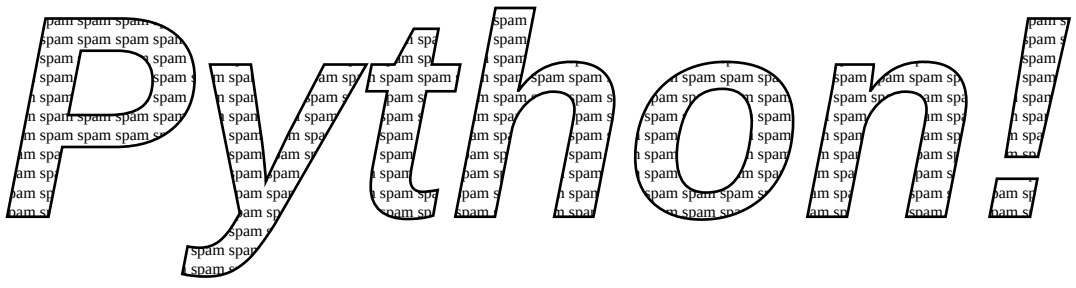
0. at origin

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

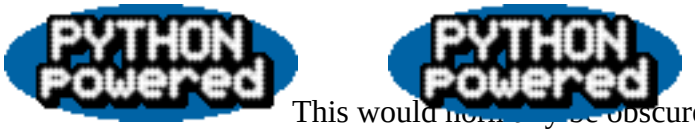
PDFgen uses the Python Imaging Library (or, under Jython, java.awt.image and javax.imageio) to process a very wide variety of image formats.
This page shows image capabilities. If I've done things right, the bitmap should have its bottom left corner aligned with the crosshairs.
There are two methods for drawing images. The recommended use is to call drawImage. This produces the smallest PDFs and the fastest generation times as each image's binary data is only embedded once in the file. Also you can use advanced features like transparency masks.
You can also use drawInlineImage, which puts images in the page stream directly. This is slightly faster for Acrobat to render or for very small images, but wastes space if you use images more than once.

inline image drawn at natural size

inline image distorted to fit box

Image XObjects can be defined once in the file and drawn many times.
This results in faster generation and much smaller files.

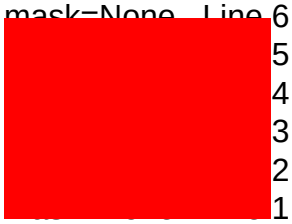
The optional 'mask' parameter lets you define transparent colors. We used a color picker to determine that the yellow in the image above is RGB=(225,223,0). We then define a mask spanning these RGB values: [254, 255, 222, 223, 0, 1]. The background vanishes!!



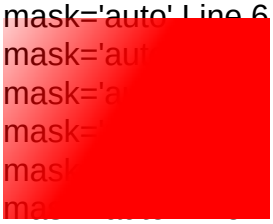
This would normally be obscured

For rgba type images we can use the alpha channel if we set mask='auto'.
The first image is solid red with variable alpha.
The second image is white alpha=0% to purple=100%

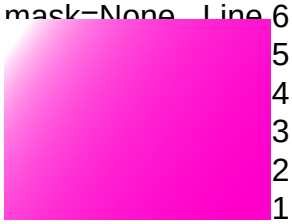
mask=None Line 7
6
5
4
3
2
1
mask=None Line 0



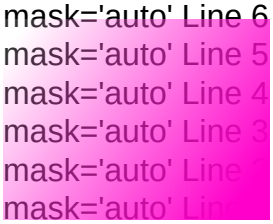
mask='auto' Line 7
mask='auto' Line 6
mask='auto' Line 5
mask='auto' Line 4
mask='auto' Line 3
mask='auto' Line 2
mask='auto' Line 1
mask='auto' Line 0



mask=None Line 7
6
5
4
3
2
1
mask=None Line 0



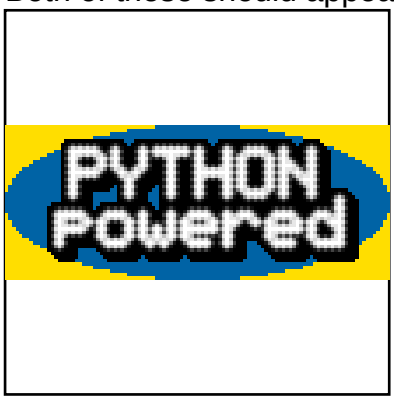
mask='auto' Line 7
mask='auto' Line 6
mask='auto' Line 5
mask='auto' Line 4
mask='auto' Line 3
mask='auto' Line 2
mask='auto' Line 1
mask='auto' Line 0

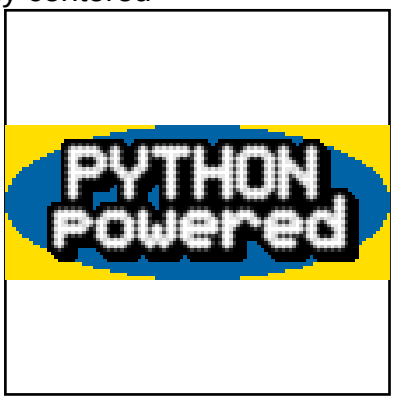


This jpeg is actually a gif

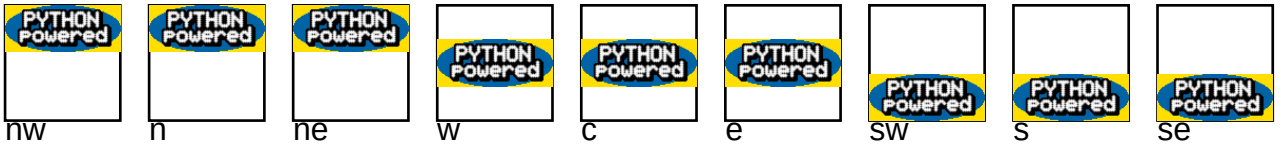
This gif is actually a jpeg

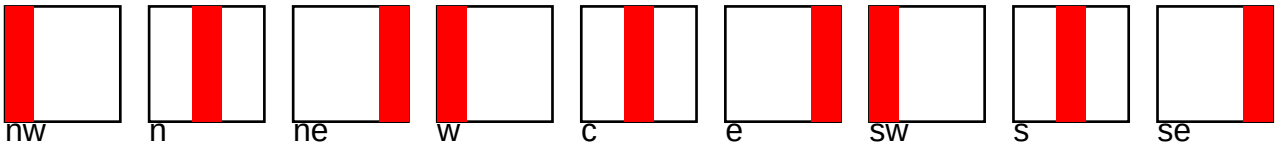
Image positioning tests with preserveAspectRatio
Both of these should appear within the boxes, vertically centered





anchored by respective corners - use both a wide and a tall one as tests





Forms and Links

Forms are sequences of text or graphics operations
which are stored only once in a PDF file and used as many times
as desired. The blue logo bar to the left is an example of a form
in this document. See the function framePageForm in this demo script
for an example of how to use canvas.beginForm(name, ...) ... canvas.endForm().

Documents can also contain cross references where (for example) a rectangle
on a page may be bound to a position on another page. If the user clicks
on the rectangle the PDF viewer moves to the bound position on the other
page. There are many other types of annotations and links supported by PDF.

For example, there is a bookmark to each page in this document and below
is a browsable index that jumps to those pages. In addition we show two
URL hyperlinks; for these, you specify a rectangle but must draw the contents
or any surrounding rectangle yourself.

Hyperlink to www.reportlab.com, with green border

mailto: hyperlink, without border

Hyperlink with custom border style

Hard link to C:/Users/ian/Documents/Development-Ross/reportlab-daily-win32/reportlab-20140105201709/

Gradients code contributed by Peter Johnson <johnson.peter@gmail.com>