Capítulo 1.3
Generación de contenido

1. PDF

1.1 Presentación

1.1.1 Formato PDF

PDF son las siglas de *Portable Document Format*, un formato de documento que utiliza un lenguaje de descripción de página que es el PDL (siglas de *Page Document Language*) y un protocolo de impresión independiente del fabricante, que es una evolución de Postscript. Se ha convertido, progresivamente, en el estándar de impresión de documentos y en una norma ISO.

A día de hoy existen muchos formatos de datos (texto, dibujos, imágenes...) que incluyen funcionalidades para su exportación a PDF.

1.1.2 Ventajas

Sus principales ventajas son:

– concordancia entre la representación en pantalla y lo que realmente se imprimirá;
– independencia respecto al sistema operativo;
– independencia respecto al hardware;
– la existencia de aplicaciones y librerías libres para este formato;
– la gran difusión de lectores de documentos PDF, la mayoría de ellos libres y gratuitos.

1.1.3 Inconvenientes

Los principales inconvenientes son de varios tipos:

– Los permisos vinculados a cada documento dependen, a la vez, de los propios del formato, además de aquellos ligados a todo lo que está contenido en un documento, es decir, permisos de los autores sobre los textos, las imágenes incrustadas, o incluso los tipos de letras utilizados.

– La evolución del formato está vinculada, principalmente, a la política de un único fabricante.

1.1.4 Presentación de la librería libre

ReportLab es una librería externa escrita en Python que ofrece herramientas sencillas y con buen rendimiento para generar documentos PDF. Esta librería es mantenida por un fabricante que proporciona dos ramas, una de ellas destinada a la comunidad, que es la que nos interesa utilizar.

Está migrada a Python 3, en particular a partir de Python 3.3, y el código puede encontrarse en: https://github.com/nakagami/reportlab

1.2 Bajo nivel

1.2.1 Librería de datos

Lo primero que hay que tener en cuenta para crear un documento es disponer de los datos prefabricados. En efecto, es conveniente disponer de los formatos listos correspondientes a un A4, A3 o cualquier otro formato habitual, con colores predefinidos listos para ser empleados, y un sistema que permita convertir las medidas que manejamos en aquellas utilizadas por ReportLab, a saber, un **`point`**, la unidad de medida de las pantallas y de las impresoras.

Esto es lo que podemos encontrar si profundizamos un poco en esta librería de datos:

```
>>> from reportlab import lib
>>> dir(lib)
['RL_DEBUG', '__builtins__', '__doc__', '__file__', '__name__',
'__package__', '__path__', '__version__', 'boxstuff', 'colors',
'logger', 'os', 'pagesizes', 'rltempfile', 'units', 'utils']
```

He aquí cómo utilizar el módulo dedicado a los formatos estándar más habituales en las impresoras:

```
>>> from reportlab.lib import pagesizes
>>> dir(pagesizes)
['A0', 'A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'B0', 'B1', 'B2', 'B3',
'B4', 'B5', 'B6', 'ELEVENSEVENTEEN', 'LEGAL', 'LETTER', '_BH',
'_BW', '_H', '_W', '__builtins__', '__doc__', '__file__',
'__name__', '__package__', '__version__', 'cm', 'elevenSeventeen',
'inch', 'landscape', 'legal', 'letter', 'portrait']
```

Aquí vemos algunas de las unidades:

```
>>> units.inch
72.0
>>> units.cm
28.346456692913385
```

Existen, por tanto, 72 puntos por pulgada y 28 por centímetro. Encontramos fácilmente que una pulgada mide 2,54 centímetros. Será preferible hablar de una unidad que sea homogénea en todo el sistema internacional, aunque la propia librería utiliza unidades inglesas:

```
>>> units.inch / units.cm
2.54
```

Es posible, de este modo, convertir las medidas de las páginas a centímetros:

```
>>> [s / cm for s in pagesizes.A4]
[21.0, 29.7]
```

Existe un último módulo, particularmente útil, que permite no complicarse la vida para crear los colores, y que dispone de aquellos de uso más común:

```
>>> from reportlab.lib import colors
>>> dir(colors)
```

El resultado de este comando es relativamente largo, dado el gran número de colores disponibles. Conviene verificar que EL color que se desea utilizar se encuentra en esta lista.

He aquí cómo se presentan los colores:

```
>>> colors.yellow
Color(1,1,0,1)
>>> colors.yellowgreen
Color(.603922,.803922,.196078,1)
>>> colors.turquoise
Color(.25098,.878431,.815686,1)
>>> colors.olive
Color(.501961,.501961,0,1)
```

Se trata, en realidad, de una especie de 4-tupla que presenta un color en formato CMYK (Cian, Magenta, Amarillo, Negro o, en inglés, *Cyan*, *Magenta*, *Yellow*, *Black* o *Key*), a diferencia de lo que podríamos pensar en un primer momento, dado que se trata del formato que utilizan las impresoras y no un formato RGB.

Es posible obtener información relativa a un color mediante varios métodos diferentes:

```
>>> c = colors.turquoise
>>> c.red, c.green, c.blue, c.alpha
(0.25098039215686274, 0.8784313725490196, 0.8156862745098039, 1)
>>> c.hexval(), c.hexvala()
('0x40e0d0', '0x40e0d0ff')
>>> c.rgb(), c.rgba()
((0.25098039215686274, 0.8784313725490196, 0.8156862745098039),
(0.25098039215686274, 0.8784313725490196, 0.8156862745098039, 1))
>>> c.bitmap_rgb(), c.bitmap_rgba()
((64, 224, 208), (64, 224, 208, 255))
```

Y, si no se encuentra el color adecuado, siempre es posible crearlo. He aquí, por ejemplo, un «Azul claro», cuyos valores provienen de Wikipedia (https://es.wikipedia.org/wiki/Anexo:Colores):

```
>>> c = colors
>>> c = colors.Color(116./255, 208./255, 241./255, 1)
>>> c
Color(.454902,.815686,.945098,1)
```

Del mismo modo, es posible fabricar un formato de impresión, pues no se trata sino de una 2-tupla, aunque esto es más extraño. ReportLab tiene todo lo que puede hacer falta a este nivel.

Un último aspecto, muy importante, es la gestión de los tipos de letra. De forma predeterminada, se utilizan 14 de ellas, que están estandarizadas y se encuentran disponibles siempre, sin ningún esfuerzo particular; basta con invocarlas utilizando su nombre.

He aquí estas tipografías:

```
>>> from reportlab.pdfbase import pdfmetrics
>>> pdfmetrics.standardFonts
('Courier', 'Courier-Bold', 'Courier-Oblique', 'Courier-
BoldOblique', 'Helvetica', 'Helvetica-Bold', 'Helvetica-Oblique',
'Helvetica-BoldOblique', 'Times-Roman', 'Times-Bold', 'Times-
Italic', 'Times-BoldItalic', 'Symbol', 'ZapfDingbats')
```

Su uso es libre. Aun así, es posible utilizar otros tipos de letra, que pueden estar potencialmente sometidos a algún tipo de licencia, y que es preciso cargar e incluir en el documento.

1.2.2 Canvas (Lienzo)

He aquí un pequeño script, disponible para los lectores de este libro, que permite utilizar el lienzo (*canvas*).

Su objetivo es mostrar lo que es posible hacer a bajo nivel. En primer lugar, se realizan las importaciones necesarias:

```
>>> from reportlab.pdfgen.canvas import Canvas
>>> from reportlab.lib.units import cm
```

He aquí cómo crear un canvas y agregarle metadatos que serán visibles por los sistemas operativos y los lectores de PDF:

```
>>> canvas = Canvas("hello.pdf")
>>> canvas.setTitle("Primer documento")
>>> canvas.setSubject("Creación de un documento PDF con ReportLab")
>>> canvas.setAuthor('SCH')
>>> canvas.setKeywords(['PDF', 'ReportLab', 'Python'])
>>> canvas.setCreator('sch')
```

Aquí vemos cómo situar un texto precisando su fuente y su tamaño:

```
>>> canvas.setFont("Helvetica", 36)
>>> canvas.drawCentredString(12.0 * cm, 18.0 * cm, "Hello world")
```

Este es otro ejemplo interesante:

```
>>> canvas.setFont("Times-Roman", 12)
>>> canvas.drawString(1.0 * cm, 1.0 * cm, "O")
>>> canvas.drawString(2.0 * cm, 1.0 * cm, "X")
>>> canvas.drawString(1.0 * cm, 2.0 * cm, "Y")
```

Se ha puesto una marca 0, X, Y, que permite ver en qué sentido se tienen en cuenta las medidas. Como puede verse en el documento generado, el punto 0 está situado abajo a la izquierda, el eje X a la derecha y el eje Y en la parte superior.

Mientras no se cambie de tipo de letra, se sigue utilizando la definida. No es preciso indicarla con cada escritura.

A continuación, se muestra la noción de alineación:

```
>>> canvas.drawString(8.5 * cm, 16.0 * cm, "I")
>>> canvas.drawRightString(8.5 * cm, 15.0 * cm, "D")
>>> canvas.drawCentredString(8.5 * cm, 14.0 * cm, "C")
>>> canvas.drawString(12.5 * cm, 16.0 * cm, "Alinear a la izquierda")
>>> canvas.drawRightString(12.5 * cm, 15.0 * cm, "Alinear a la
derecha")
>>> canvas.drawCentredString(12.5 * cm, 14.0 * cm, "Alinear al centro")
```

Por último, se pone de relieve el hecho de que el cambio de línea no puede realizarse de manera sencilla y que el texto no salta de línea de forma automática:

```
>>> canvas.drawCentredString(10.5 * cm, 10.0 * cm,
'\n'.join(["Alineado al centro"] * 5))
>>> canvas.drawString(18.5 * cm, 12.0 * cm, "Mal Alineado" * 5)
>>> canvas.drawRightString(2.5 * cm, 12.0 * cm, "Mal Alineado" * 5)
>>> canvas.save()
```

Por último, se guarda y se crea el archivo.

1.3 Alto nivel

1.3.1 Estilos

Un elemento clave en la generación de un documento con herramientas de alto nivel es la manipulación de los estilos.

He aquí cómo recuperar una hoja de estilo estándar:

```
>>> from reportlab.lib.styles import getSampleStyleSheet
>>> styles = getSampleStyleSheet()
>>> styles.list ()
```

Vemos, enseguida, el comando que debemos utilizar para cada estilo:

```
BodyText None
   name = BodyText
   parent = <ParagraphStyle 'Normal'>
   alignment = 0
   allowOrphans = 0
```

```
    allowWidows = 1
    backColor = None
    borderColor = None
    borderPadding = 0
    borderRadius = None
    borderWidth = 0
    bulletFontName = Helvetica
    bulletFontSize = 10
    bulletIndent = 0
    firstLineIndent = 0
    fontName = Helvetica
    fontSize = 10
    leading = 12
    leftIndent = 0
    rightIndent = 0
    spaceAfter = 0
    spaceBefore = 6
    textColor = Color(0,0,0,1)
    textTransform = None
    wordWrap = None
```

La siguiente lista muestra los demás estilos estándar:

– Bullet	– Heading1	– Heading4	– Italic
– Code	– Heading2	– Heading5	– Normal
– Definition	– Heading3	– Heading6	– Title

A continuación, se muestra cómo crear un nuevo estilo de párrafo:

```
>>> from reportlab.lib.styles import ParagraphStyle
>>> from reportlab.lib.enums import TA_JUSTIFY
>>> mi_estilo = ParagraphStyle(name='mi_estilo',
alignment=TA_JUSTIFY, fontName = "Helvetica", fontSize = 14)
```

Y cómo agregarlo a la lista de estilos:

```
>>> styles.add(mi_estilo)
```

Esto forma parte del modelo que hemos visto en la introducción.

El uso de parámetros nombrados es la regla, cuando se tiene muchos, que permite realizar una lectura de código más clara, dado que no se conocen de memoria las firmas de los métodos:

```
>>> help(ParagraphStyle)
```

Existen varios medios de crear estilos específicos bajo demanda antes de comenzar a crear el contenido. Todos se aplican a los textos.

No obstante, existe un caso particular en el que resulta algo especial aplicar los estilos; es el caso de las tablas:

```
>>> estilo_tabla = [
...     ('ALIGN',         (0,0),  (-1,-1), "LEFT"),
...     ('VALIGN',        (0,0),  (-1,-1), "TOP"),
...     ('LEFTPADDING',   (0,0),  (-1,-1), 0*cm),
...     ('RIGHTPADDING',  (0,0),  (-1,-1), 0*cm),
...     ('TOPPADDING',    (0,0),  (-1,-1), 0*cm),
...     ('BOTTOMPADDING', (0,0),  (-1,-1), 0*cm),
... ]
```

Esta hoja de estilos es una lista de 4-tuplas que definen, cada una, respectivamente el nombre del estilo, la celda superior izquierda y la celda inferior derecha que engloban las celdas de la tabla afectadas por el valor asociado al estilo.

Es posible crear otro estilo a partir de este para evitar una doble escritura:

```
>>> estilo_tabla1 = estilo_tabla[:]
>>> estilo_tabla1.append(('LINEABOVE', (0,0), (-1, 0), 1,
colors.turquoise))
>>> estilo_tabla1.append(('LINEABOVE', (0,1), (-1,-1), 0.5,
colors.darkturquoise))
```

Es muy importante no olvidar los **[:]** para realizar una copia, pues en caso contrario estaríamos creando un puntero hacia la propia lista.

1.3.2 Flujo de datos

Para crear un documento de texto formado por una sucesión de párrafos, o incluso de objetos como imágenes o tablas, se escribe un flujo de datos.

He aquí los elementos que hemos visto antes y que son necesarios:

```
>>> from reportlab.lib import colors
>>> from reportlab.lib.styles import getSampleStyleSheet
>>> from reportlab.lib.styles import ParagraphStyle
>>> from reportlab.lib.enums import TA_JUSTIFY
>>> from reportlab.lib.pagesizes import A4
>>> from reportlab.lib.units import cm
```

El modelo que se utiliza es platypus, cuyo elemento central es:

```
>>> from reportlab.platypus import SimpleDocTemplate
```

Es preciso crear una lista de otros objetos de este mismo módulo:

```
>>> flowables = []
```

Una vez definidos los estilos, tal y como hemos visto en la subsección Estilos de este capítulo, es posible agregar un párrafo de la siguiente manera:

```
>>> from reportlab.platypus import Paragraph
>>> flowables.append(Paragraph("Archivo PDF
Generado", styles["Heading1"]))
>>> flowables.append(Paragraph("Sébastien
CHAZALLET", styles["Normal"]))
>>> flowables.append(Paragraph("http://
www.inspyration.com", styles["Code"]))
```

En cada párrafo se precisa el estilo. Es posible insertar contenido en varias líneas, aunque no formarán más que una única línea en el resultado final:

```
>>> content = """Este documento lo genera el script 02_flux.py.
... Este script se entrega con este libro.
... Puede modificarlo para construir sus propios proyectos"""
>>> flowables.append(Paragraph(content, styles["Normal"]))
```

Existe, también, un objeto particular para insertar un salto de línea:

```
>>> from reportlab.platypus import  Spacer
>>> flowables.append(Spacer (0, 0.2*cm))
```

Otro para gestionar el salto de página:

```
>>> from reportlab.platypus import   PageBreak
>>> flowables.append(PageBreak())
```

He aquí cómo escribir una tabla. Se trata de una lista de listas de párrafos, cada una con su estilo (además del propio estilo de la tabla):

```
>>> data, line = [], []
>>> line.append( Paragraph ("Tecnología", styles["Normal"]) )
>>> line.append( Paragraph ("Aplicaciones", styles["Normal"]) )
>>> line.append( Paragraph ("Alternativas", styles["Normal"]) )
>>> data.append(line)
>>> line = []
>>> line.append( Paragraph ("OS", styles["Normal"]) )
>>> line.append( Paragraph ("Debian", styles["Normal"]) )
>>> line.append( Paragraph ("Ubuntu, Fedora", styles["Normal"]) )
>>> data.append(line)
```