



Biblioteca PDF de ReportLab

Guía del usuario

ReportLab Versión 3.5.13

Documento generado el 07/02/2019 20:17:44

Tabla de contenido

Tabla de contenido	2
Capítulo 1 Introducción	6
1.1 Acerca de este documento.	6
1.2 ¿Qué es la biblioteca PDF de ReportLab?	6
1.3 Software comercial de ReportLab.	7
1.4 ¿Qué es Python?	7
1.5 Agradecimientos.	7
1.6 Instalación y configuración.	8
1.7 Involucrarse.	8
1.8 Configuración del sitio.	8
1.9 Aprender más sobre Python.	9
1.10 Objetivos para la serie de versiones 3.x.	9
Capítulo 2 Gráficos y texto con pdfgen	10
2.1 Conceptos básicos.	10
2.2 Más sobre el Lienzo.	10
2.3 Operaciones de dibujo.	11
2.4 Las herramientas: las operaciones de "dibujo".	12
2.5 La caja de herramientas: las operaciones de "cambio de estado".	14
2.6 Otros métodos de lienzo.	dieciséis
2.7 Coordenadas (espacio de usuario predeterminado).	dieciséis
2.8 colores.	21
2.9 Comprobación del espacio de color.	24
2.10 Sobreimpresión en color.	24
2.11 Fuentes estándar y objetos de texto.	26
2.12 Métodos de objetos de texto.	28
2.13 Trayectorias y Líneas.	34
2.14 Rectángulos, círculos, elipses.	38
2.15 curvas de Bézier.	39
2.16 Métodos de objeto de ruta.	41
2.17 Lecturas adicionales: Biblioteca de gráficos ReportLab.	46
Capítulo 3 Fuentes y codificaciones	47
3.1 Unicode y UTF8 son las codificaciones de entrada predeterminadas.	47
3.2 Sustitución automática de fuentes de salida.	47

3.3 Uso de fuentes Tipo 1 no estándar.	48
3.4 Codificaciones de fuentes estándar de un solo byte.	49
3.5 Compatibilidad con fuentes TrueType.	51
3.6 Compatibilidad con fuentes asiáticas.	52
3.7 Pruebas de RenderPM.	53
Capítulo 4 Exponer las capacidades especiales de PDF	54
4.1 Formularios.	54
4.2 Enlaces y Destinos.	54
4.3 Árboles de contorno.	56
4.4 Efectos de transición de página.	56
4.5 Anotaciones de archivos internos.	56
4.6 Cifrado.	57
4.7 Formularios interactivos.	59
Capítulo 5 PLATYPUS - Diseño de página y tipografía usando scripts	64
5.1 Objetivos de diseño.	64
5.2 Primeros pasos.	<small>Sesenta y cinco</small>
5.3 Fluidos.	66
5.4 Directrices para un posicionamiento fluido.	66
5.5 Marcos.	67
5.6 Documentos y Plantillas.	68
Capítulo 6 Párrafos	71
6.1 Uso de estilos de párrafo.	72
6.2 Etiquetas de marcado XML de párrafo.	77
6.3 Marcado dentro del párrafo.	78
6.4 Numeración de viñetas y párrafos.	81
Capítulo 7 Tablas y estilos de tabla	83
7.1 Métodos de usuario de tablas.	83
7.2 Estilo de tabla.	84
7.3 Métodos de usuario de TableStyle.	84
7.4 Comandos de estilo de tabla.	84
Capítulo 8 Programación de flujos	90
8.1 DocAssign(self, var, expr, life='forever')	90
8.2 DocExec(self, stmt, lifelife='forever')	90

8.3 DocPara(self, expr, format=Ninguno, estilo=Ninguno, klass=Ninguno, escape=True)	90
8.4 DocAssert(self, cond, format=Ninguno)	90
8.5 DocIf(self, cond, thenBlock, elseBlock=[])	90
8.6 DocWhile(self, cond, whileBlock)	90
Capítulo 9 Otros fluidos útiles	91
9.1 Preformatoado(texto, estilo, viñetaTexto=Ninguno, sangría=0, maxLineLength=Ninguno, splitChars=Ninguno, newLineChars=Ninguno)	91
9.2 XReformatoado(texto, estilo, viñetaTexto=Ninguno, sangría=0, fragmentos=Ninguno)	91
9.3 Imagen (nombre de archivo, ancho = Ninguno, alto = Ninguno)	92
9.4 Espaciador (ancho, alto)	93
9.5 Salto de página()	93
9.6 CondPageBreak(altura)	93
9.7 KeepTogether(fluidos)	93
9.8 Tabla de contenidos()	93
9.9 ÍndiceSimple()	94
9.10 ListFlowable(), ListItem()	95
9.11 Columnas Equilibradas()	96
Capítulo 10 Escribiendo tus propios objetos fluidos	97
10.1 Un Flowable muy simple	97
10.2 Modificación de un Flowable incorporado	98
Capítulo 11 Gráficos	100
11.1 Introducción	100
11.2 Conceptos generales	100
11.3 Gráficos	103
11.4 Etiquetas	105
11.5 Ejes	106
11.6 Gráficos de barras	110
11.7 Gráficos de líneas	113
11.8 Gráficos lineales	114
11.9 Gráficos circulares	116
11.10 Leyendas	120
11.11 Formas	121
11.12 Aparatos	126
Apéndice A Demostraciones de ReportLab	132
A.1 Odyssey	132

A.2 Fuentes y colores estándar.	132
A.3 Py2pdf.	132
A.4 Papel tábano .	133
A.5 Punto de Python.	133

Capítulo 1 Introducción

1.1 Acerca de este documento

Este documento es una introducción a la biblioteca PDF de ReportLab. Alguna experiencia previa en programación es Se presume y se recomienda estar familiarizado con el lenguaje de programación Python. Si eres nuevo en Python, Te contamos en el siguiente apartado dónde acudir para orientarte.

Este manual no cubre el 100% de las funciones, pero debe explicar todos los conceptos principales y ayudarle a obtener comenzó y le indicará otros recursos de aprendizaje. Después de superar esto, deberías estar listo. comenzar a escribir programas para producir informes sofisticados.

En este capítulo, cubriremos el trabajo preliminar:

- ¿De qué se trata ReportLab y por qué debería usarlo?
- ¿Qué es Python?
- ¿Cómo puedo configurar y ejecutar todo?

Necesitamos su ayuda para asegurarnos de que este manual sea completo y útil. Por favor envíe cualquier comentario a nuestro usuario. lista de correo, que está señalizada desde www.reportlab.com.

1.2 ¿Qué es la biblioteca PDF de ReportLab?

Esta es una biblioteca de software que le permite crear documentos directamente en el formato de documento portátil (PDF) de Adobe. utilizando el lenguaje de programación Python. También crea cuadros y gráficos de datos en varios mapas de bits y vectores. formatos además de PDF.

PDF es el estándar mundial para documentos electrónicos. Admite impresión de alta calidad pero es totalmente portátil en todas las plataformas, gracias al Acrobat Reader, disponible gratuitamente. Cualquier aplicación que haya generado previamente Los informes impresos o el manejo de una impresora pueden beneficiarse al crear documentos PDF; estos se pueden archivar, enviado por correo electrónico, colocado en la web o impreso a la antigua usanza. Sin embargo, el formato de archivo PDF es complejo. formato binario indexado que es imposible escribir directamente. La especificación del formato PDF es más de 600 páginas de largo y los archivos PDF deben proporcionar desplazamientos de bytes precisos: un único carácter adicional colocado en cualquier lugar de un Un documento PDF válido puede invalidarlo. Esto hace que sea más difícil de generar que HTML.

La mayoría de los documentos PDF del mundo han sido producidos por las herramientas Acrobat de Adobe o por rivales como JAWS. PDF Creator, que actúan como 'controladores de impresión'. Cualquiera que desee automatizar la producción de PDF normalmente utilizará un Producto como Quark, Word o Framemaker ejecutándose en bucle con macros o complementos, conectados a Acrobat. Los canales de varios idiomas y productos pueden ser lentos y algo difíciles de manejar.

La biblioteca ReportLab crea directamente PDF basado en sus comandos gráficos. No hay intervención pasos. Sus aplicaciones pueden generar informes extremadamente rápido, a veces mucho más rápido que las herramientas tradicionales de redacción de informes. Este enfoque es compartido por varias otras bibliotecas: PDFlib para C, iText para Java, iTextSharp para .NET y otros. Sin embargo, la biblioteca ReportLab se diferencia en que puede funcionar a niveles mucho más altos. niveles, con un motor con todas las funciones para diseñar documentos completos con tablas y gráficos.

Además, como se escribe un programa en un potente lenguaje de propósito general, no hay ninguna restricción sobre de dónde se obtienen los datos, cómo se transforman y el tipo de salida que se puede crear. Y puede reutilizar código en familias enteras de informes.

Se espera que la biblioteca ReportLab sea útil al menos en los siguientes contextos:

- Generación dinámica de PDF en la web
- Informes corporativos de gran volumen y publicación de bases de datos
- Un motor de impresión integrable para otras aplicaciones, incluido un 'lenguaje de informes' para que los usuarios puede personalizar sus propios informes. Esto es particularmente relevante para las aplicaciones multiplataforma que no pueden depender de una API de impresión o vista previa consistente en cada sistema operativo.
- Un 'sistema de compilación' para documentos complejos con gráficos, tablas y texto, como cuentas de gestión, informes estadísticos y artículos científicos.
- Pasar de XML a PDF en un solo paso

1.3 Software comercial de ReportLab

La biblioteca ReportLab forma la base de nuestra solución comercial para la generación de PDF, Report Markup Language (RML). Esto está disponible para su evaluación en nuestro sitio web con la documentación completa. Creemos que RML es la forma más rápida y sencilla de desarrollar flujos de trabajo PDF enriquecidos. Trabaja en un lenguaje de marcado a un nivel similar al HTML, utilizando su sistema de plantillas favorito para completar un documento RML; luego llame a nuestra función API rml2pdf para generar un PDF. Es lo que utiliza el personal de ReportLab para crear todas las soluciones que puede ver en reportlab.com. Diferencias clave:

- Totalmente documentado con dos manuales, una especificación formal (la DTD) y extensas pruebas autodocumentadas. (Por el contrario, intentamos asegurarnos de que la documentación de código abierto no sea incorrecta, pero no siempre estamos al día con el código)
- Trabaja en marcado de alto nivel en lugar de construir gráficos de objetos Python. No requiere experiencia en Python: ¡sus colegas pueden agradecerle cuando se haya ido!
- Soporte para gráficos vectoriales e inclusión de otros documentos PDF.
- Muchas más funciones útiles expresadas con una sola etiqueta, que requerirían mucha codificación en el paquete de código
- abierto. Se incluye soporte comercial.

Pedimos a los desarrolladores de código abierto que consideren probar RML cuando sea apropiado. Puede registrarse en nuestro sitio y probar una copia antes de comprar. Los costos son razonables y están vinculados al volumen del proyecto, y los ingresos nos ayudan a dedicar más tiempo a desarrollar este software.

1.4 ¿Qué es Python?

Python es un lenguaje de programación interpretado, interactivo y orientado a objetos . A menudo se lo compara con Tcl, Perl, Scheme o Java.

Python combina un poder notable con una sintaxis muy clara. Tiene módulos, clases, excepciones, tipos de datos dinámicos de muy alto nivel y escritura dinámica. Hay interfaces para muchas bibliotecas y llamadas al sistema, así como para varios sistemas de ventanas (X11, Motif, Tk, Mac, MFC). Los nuevos módulos integrados se escriben fácilmente en C o C++. Python también se puede utilizar como lenguaje de extensión para aplicaciones que necesitan una interfaz programable.

Python es tan antiguo como Java y su popularidad ha ido creciendo constantemente durante años; Desde que salió nuestra biblioteca por primera vez, se ha convertido en algo común. Muchos usuarios de la biblioteca ReportLab ya son devotos de Python, pero si no lo eres, creemos que el lenguaje es una excelente opción para aplicaciones de generación de documentos debido a su expresividad y capacidad de obtener datos desde cualquier lugar.

Python tiene derechos de autor, pero se puede utilizar y distribuir libremente, incluso para uso comercial.

1.5 Agradecimientos

Mucha gente ha contribuido a ReportLab. Nos gustaría agradecer en particular (en orden alfabético): Albertas Agejevas, Alex Buck, Andre Reitz, Andrew Cutler, Andrew Mercer, Ben Echols, Benjamin Dumke, Benn B, Chad Miller, Chris Buergi, Chris Lee, Christian Jacobs, Dinu Gherman, Edward Greve, Eric Johnson, Felix Labrecque, Fubu @ bitbucket, Gary Poster, Germán M. Bravo, Guillaume Francois, Hans Brand, Henning Vonbargen, Hosam Aly, Ian Stevens, James Martin-Collar, Jeff Bauer, Jerome Alet, Jerry Casiano, Jorge Godoy, Keven D Smith, Kyle MacFarlane, Magnus Lie Hetland, Marcel Tromp, Marius Gedminas, Mark de Wit, Matthew Duggan, Matthias Kirst, Matthias Klose, Max M, Michael Egorov, Michael Spector, Mike Folwell, Mirko Dziadzka, Moshe Wagner, Nate Silva, Paul McNett, Peter Johnson, PJACock, Publio da Costa Melo, Randolph Bentson, Robert Alsina, Robert Hözl, Robert Kern, Ron Peleg, Ruby Yocom, Simon King, Stephan Richter, Steve Halasz, Stoneleaf @ bitbucket, T Blatter, Tim Roberts, Tomasz Swiderski, Ty Sarna, Volker Haas, Yoann Roman y muchos más.

Un agradecimiento especial a Just van Rossum por su valiosa ayuda con los tecnicismos tipográficos.

Moshe Wagner y Hosam Aly merecen un enorme agradecimiento por contribuir al parche RTL, que aún no está en el baúl.

Marius Gedminas merece un gran reconocimiento por contribuir con el trabajo sobre las fuentes TrueType y nos complace incluirlas en el conjunto de herramientas. Finalmente agradecemos a Michal Kosmulski por la fuente DarkGarden y a Bitstream Inc. por

las fuentes Vera.

1.6 Instalación y configuración

Para evitar duplicaciones, las instrucciones de instalación se guardan en el archivo README de nuestra distribución, que puede Puede verse en línea en <http://bitbucket.org/rptlab/reportlab/>

Esta versión (3.0) de ReportLab requiere las versiones 2.7, 3.3 o superiores de Python. Si necesita utilizar Python 2.5 o 2.6, utilice el último paquete ReportLab 2.x.

1.7 Involucrarse

ReportLab es un proyecto de código abierto. Aunque somos una empresa comercial, proporcionamos las fuentes principales de generación de PDF de forma gratuita, incluso con fines comerciales, y no obtenemos ingresos directamente de estos módulos. También agradecemos la ayuda de la comunidad tanto como cualquier otro proyecto de código abierto. Hay muchas maneras en el que puedes ayudar:

- Comentarios generales sobre la API principal. ¿Funciona para ti? ¿Hay algunas asperezas? ¿Hay algo que te parezca torpe e incómodo?
- Nuevos objetos para poner en informes, o utilidades útiles para la biblioteca. Tenemos un estándar abierto para objetos de informes, por lo que si ha escrito un buen gráfico o clase de tabla, ¿por qué no contribuir con él?
- Fragmentos y estudios de casos: si ha producido algún resultado interesante, regístrate en línea en <http://www.reportlab.com> y envíe un fragmento de su resultado (con o sin guiones). Si ReportLab le resolvió un problema en el trabajo, escriba un pequeño "estudio de caso" y envíelo. Y si su sitio web utiliza nuestras herramientas para realizar informes, permítanos vincularlo. Estaremos encantados de mostrar su trabajo (y acredítelo con su nombre y empresa) en nuestro sitio!
- Trabajando en el código central: tenemos una larga lista de cosas por perfeccionar o implementar. Si usted es Te faltan algunas funciones o simplemente quiere ayudar, ¡háznos saber!

El primer paso para cualquiera que quiera aprender más o participar es unirse a la lista de correo. Para suscribirse visita <http://two.pairlist.net/mailman/listinfo/reportlab-users>. Desde allí también puedes explore los archivos y contribuciones del grupo. La lista de correo es el lugar para informar errores y obtener apoyo.

El código ahora reside en BitBucket (<http://bitbucket.org/rptlab/reportlab/>) en un Mercurial repositorio, junto con un rastreador de problemas y wiki. Todos deberían sentirse libres de contribuir, pero si está trabajando activamente en algunas mejoras o desea llamar la atención sobre un problema, utilice la lista de correo para informarnos. saber.

1.8 Configuración del sitio

Hay una serie de opciones que probablemente deban configurarse globalmente para un sitio. La secuencia de comandos de Python El módulo reportlab/rl_config.py se puede editar para cambiar los valores de varios valores importantes en todo el sitio. propiedades.

- detallado: establecido en valores enteros para controlar la salida de diagnóstico.
- ShapeChecking: configúrelo en cero para desactivar muchas comprobaciones de errores en los módulos gráficos.
- defaultEncoding: configúrelo en WinAnsiEncoding o MacRomanEncoding.
- defaultPageSize: configúrelo en uno de los valores definidos en reportlab/lib/pagesizes.py; Como entregado está configurado en tamaños de página.A4; otros valores son tamaños de página, letras, etc.
- defaultImageCaching: establecido en cero para inhibir la creación de archivos .a85 en su disco duro. El valor predeterminado es crear estos archivos de imagen compatibles con PDF preprocesados para una carga más rápida.
- T1SearchPath: esta es una lista de Python de cadenas que representan directorios que pueden consultarse para obtener información sobre fuentes Tipo 1.
- TTFSearchPath: esta es una lista de Python de cadenas que representan directorios que pueden consultarse información sobre fuentes TrueType
- CMapSearchPath: esta es una lista de Python de cadenas que representan directorios que pueden consultarse información sobre mapas de códigos de fuentes.
- showBoundary: establecido en un valor distinto de cero para dibujar líneas de límite.

- `ZLIB_WARNINGS`: establecido en distinto de cero para recibir advertencias si la extensión de compresión de Python no es encontrado.
- `pageCompression`: establezca un valor distinto de cero para intentar obtener un PDF comprimido.
- `allowTableBoundsErrors`: establecido en 0 para forzar un error en elementos de tabla Platypus muy grandes
- `vaciaTableAction`: controla el comportamiento de las tablas vacías, puede ser 'error' (predeterminado), 'indicar' o 'ignorar'.

1.9 Aprender más sobre Python

Si es un principiante total en Python, debe consultar uno o más del creciente número de recursos sobre programación en Python. Los siguientes están disponibles gratuitamente en la web:

- Documentación de Python. Una lista de documentación en el sitio web Python.org.
<http://www.python.org/doc/>
- Tutorial de Python. El tutorial oficial de Python, escrito originalmente por el propio Guido van Rossum. <http://docs.python.org/tutorial/>
- Aprendiendo a programar. Un tutorial sobre programación de Alan Gauld. Tiene un fuerte énfasis en Python, pero también utiliza otros lenguajes.
<http://www.freenetpages.co.uk/hp/alan.gauld/>
- Pitón instantáneo. Un curso intensivo mínimo de 6 páginas realizado por Magnus Lie Hetland.
<http://www.hetland.org/python/instant-python.php>
- Sumérgete en Python. Un tutorial gratuito de Python para programadores experimentados.
<http://www.diveintopython.net/>

1.10 Objetivos para la serie de versiones 3.x

ReportLab 3.0 se produjo para ayudar en la migración a Python 3.x. Python 3.x será estándar en el futuro

Ubuntu se lanza y está ganando popularidad, y una buena proporción de los principales paquetes de Python ahora se ejecutan en Python 3.

- Compatibilidad con Python 3.x. Se debe ejecutar una sola línea de código en 2.7 y 3.3.
- `__init__.py` se restringe a 2.7 o ≥ 3.3
- `__init__.py` permite la importación de `reportlab.local_rl_mods` opcionales para permitir el parcheo de monos etc.
- `rl_config` ahora importa `rl_settings` y opcionalmente `local_rl_settings`
- Las extensiones de ReportLab C ahora se encuentran dentro de `reportlab`; `_rl_accel` ya no es necesario. Todo `_rl_accel` las importaciones ahora pasan por `reportlab.lib.rl_accel`
- `xmllib` desapareció, junto con el material del paraparser que causó problemas a favor de `HTMLParser`.
- algunas extensiones C obsoletas (`sgmlop` y `pyHnj`) han desaparecido
- Soporte mejorado para sistemas multiproceso para el módulo de extensión `_rl_accel` C.
- Se eliminó `reportlab/lib/para.py` y `pycanvas.py`. Es mejor que estos pertenezcan a paquetes de terceros, que pueden hacer uso de la función de parcheo de monos mencionada anteriormente.
- Agregue la capacidad de generar imágenes en escala de grises y PIL de 1 bit sin conversión a RGB. (contribuido por Matthew Duggan)
- anotación destacada (aportada por Ben Echols)
- Cumplimiento total de pip, easy_install, ruedas, etc.

Las notas de versión detalladas están disponibles en

<http://www.reportlab.com/software/documentation/relnotes/30/>

Capítulo 2 Gráficos y texto con pdfgen

2.1 Conceptos básicos

El paquete pdfgen es la interfaz de nivel más bajo para generar documentos PDF. Un programa pdfgen es esencialmente una secuencia de instrucciones para "pintar" un documento en una secuencia de páginas. El objeto de interfaz que proporciona las operaciones de pintura es el lienzo pdfgen.

El lienzo debe considerarse como una hoja de papel blanco con puntos en la hoja identificados mediante coordenadas cartesianas (X,Y) que, de forma predeterminada, tienen el punto de origen (0,0) en la esquina inferior izquierda de la página. Además, la primera coordenada x va hacia la derecha y la segunda coordenada y sube, por defecto.

A continuación se muestra un programa de ejemplo sencillo que utiliza un lienzo.

```
desde reportlab.pdfgen importar lienzo def hola(c):
    c.drawString(100,100,"Hola mundo") c =
    canvas.Canvas("hola.pdf") hola(c) c.showPage()
    c.save()
```

El código anterior crea un objeto de lienzo que generará un archivo PDF llamado hello.pdf en el directorio de trabajo actual. Luego llama a la función hola pasando el lienzo como argumento. Finalmente, el método showPage guarda la página actual del lienzo y el método save almacena el archivo y cierra el lienzo.

El método showPage hace que el lienzo deje de dibujar en la página actual y cualquier operación adicional se dibujará en una página posterior (si hay más operaciones, si no, no se crea ninguna página nueva). Se debe llamar al método save una vez completada la construcción del documento: genera el documento PDF, que es el propósito principal del objeto canvas.

2.2 Más sobre el lienzo

Antes de describir las operaciones de dibujo, haremos una digresión para cubrir algunas de las cosas que se pueden hacer para configurar un lienzo. Hay muchas configuraciones diferentes disponibles. Si es nuevo en Python o no puede esperar para producir algún resultado, puede saltar adelante, ¡pero regrese más tarde y lea esto!

En primer lugar, veremos los argumentos del constructor del lienzo:

```
def __init__(self,nombre de archivo,
            tamaño de página=(595.27,841.89),
            abajo hacia arriba =
            1, compresión de página=0,
            codificación=rl_config.defaultEncoding, verbosidad=0
            cifrar=Ninguno):
```

El argumento del nombre de archivo controla el nombre del archivo PDF final. También puede pasar cualquier objeto de archivo abierto (como sys.stdout, la salida estándar del proceso Python) y el documento PDF se escribirá en él.

Dado que PDF es un formato binario, debes tener cuidado al escribir otras cosas antes o después; ¡No puedes entregar documentos PDF en línea en medio de una página HTML!

El argumento del tamaño de página es una tupla de dos números en puntos (1/72 de pulgada). El lienzo tiene como valor predeterminado A4 (un tamaño de página estándar internacional que difiere del tamaño de página estándar estadounidense de carta), pero es mejor especificarlo explícitamente. Los tamaños de página más comunes se encuentran en el módulo de biblioteca reportlab.lib.pagesizes, por lo que puede usar expresiones como

```
de reportlab.lib.pagesizes importar letra, A4 myCanvas = Canvas('myfile.pdf',
pagesize=letter) ancho, alto = letra #guardar para más adelante
```



Si tiene problemas para imprimir su documento, asegúrese de utilizar el tamaño de página correcto (generalmente A4 o carta). Algunas impresoras no funcionan bien con páginas demasiado grandes o demasiado pequeñas.

Muy a menudo, querrás calcular cosas en función del tamaño de la página. En el ejemplo anterior extrajimos el ancho y el alto. Más adelante en el programa podemos usar la variable ancho para definir un margen derecho como ancho - pulgadas en lugar de usar una constante. Al utilizar variables, el margen seguirá teniendo sentido incluso si cambia el tamaño de la página.

El argumento ascendente cambia los sistemas de coordenadas. Algunos sistemas gráficos (como PDF y PostScript) colocan (0,0) en la parte inferior izquierda de la página, otros (como muchas interfaces gráficas de usuario [GUI]) colocan el origen en la parte superior izquierda. El argumento ascendente está en desuso y es posible que se elimine en el futuro. Es necesario ver si realmente funciona para todas las tareas y, si no, deshacerse de él.

La opción pageCompression determina si el flujo de operaciones de PDF para cada página está comprimido. De forma predeterminada, los flujos de páginas no se comprimen porque la compresión ralentiza el proceso de generación de archivos. Si el tamaño de salida es importante, establezca pageCompression=1, pero recuerde que los documentos comprimidos serán más pequeños, pero más lentos de generar. Tenga en cuenta que las imágenes siempre están comprimidas y esta opción solo ahorrará espacio si tiene una gran cantidad de texto y gráficos vectoriales en cada página.

El argumento de codificación está en gran medida obsoleto en la versión 2.0 y probablemente el 99% de los usuarios pueda omitirlo. Su valor predeterminado está bien a menos que necesites usar específicamente uno de los aproximadamente 25 caracteres que están presentes en MacRoman y no en Winansi. Una referencia útil a estos se encuentra aquí: <http://www.alanwood.net/demos/charsetdiffs.html>. El parámetro determina qué codificación de fuente se utiliza para las fuentes estándar Tipo 1; esto debería corresponder a la codificación de su sistema. Tenga en cuenta que esta es la codificación utilizada internamente por la fuente; El texto que pase al kit de herramientas de ReportLab para su procesamiento siempre debe ser un objeto de cadena Unicode de Python o una cadena de bytes codificada en UTF-8 (consulte el siguiente capítulo). La codificación de fuente tiene dos valores actualmente: 'WinAnsiEncoding' o 'MacRomanEncoding'. La variable rl_config.defaultEncoding anterior apunta a la primera, que es estándar en Windows, Mac OS X y muchos Unices (incluido Linux). Si es usuario de Mac y no tiene OS X, es posible que desee realizar un cambio global: modifique la línea en la parte superior de reportlab/pdfbase/pdfdoc.py para cambiarla. De lo contrario, probablemente puedas ignorar este argumento por completo y no pasarlo nunca. Para todas las fuentes TTF y CID de uso común, la codificación que ingresa aquí se ignora, ya que la biblioteca reportlab conoce las codificaciones correctas en esos casos.

El script de demostración reportlab/demos/stdfonts.py imprimirá dos documentos de prueba que muestran todos los puntos de código en todas las fuentes, para que pueda buscar caracteres. Se pueden insertar caracteres especiales en comandos de cadena con las secuencias de escape habituales de Python; por ejemplo \101 = 'A'.

El argumento de detalle determina cuánta información de registro se imprime. De forma predeterminada, es cero para ayudar a las aplicaciones que desean capturar PDF desde una salida estándar. Con un valor de 1, recibirá un mensaje de confirmación cada vez que se genere un documento. Cifras más altas pueden dar lugar a una mayor producción en el futuro.

El argumento de cifrado determina si el documento está cifrado y cómo. De forma predeterminada, el documento no está cifrado. Si cifrar es un objeto de cadena, se utiliza como contraseña de usuario para el pdf. Si cifrar es una instancia de reportlab.lib.pdfencrypt.StandardEncryption, este objeto se utiliza para cifrar el pdf. Esto permite un control más detallado sobre la configuración de cifrado. El cifrado se trata con más detalle en el Capítulo 4. Qué hacer: todas las funciones de información y otras cosas que no son de dibujo. Cubrir todos los argumentos del constructor, setAuthor, etc.

2.3 Operaciones de dibujo

Supongamos que la función de saludo mencionada anteriormente se implementa de la siguiente manera (aún no explicaremos cada una de las operaciones en detalle).

```
def hola(c): desde
    reportlab.lib.units import inch # mueve el origen hacia arriba
    y hacia la izquierda c.translate(inch,inch) # define una
    fuente grande c.setFont("Helvetica",
    14) # elige algunos colores
    c.setStrokeColorRGB(0.2,0.5,0.3)
    c.setFillColorRGB(1,0,1)
```

```
# dibuja algunas líneas
c.line(0,0,0,1.7*inch)
c.line(0,0,1*inch,0) # dibuja un
rectángulo
c.rect(0.2*inch,0.2*inch,1* inch,1.5*inch, fill=1) # hacer que el texto vaya hacia
arriba c.rotate(90) # cambiar color

c.setFillColorRGB(0,0,0.77)
# saludar (nota después de rotar, ¡la coord
y debe ser negativa! ) c.drawString(0,3*pulgada, -pulgada, "Hola mundo")
```

Al examinar este código, observe que existen esencialmente dos tipos de operaciones realizadas utilizando un lienzo. El primer tipo dibuja algo en la página, como una cadena de texto, un rectángulo o una línea. El segundo tipo cambia el estado del lienzo, como cambiar el color de relleno o trazo actual o cambiar el tipo y tamaño de fuente actual.

Si imaginamos el programa como un pintor trabajando en el lienzo, las operaciones de "dibujo" aplican pintura al lienzo usando el conjunto actual de herramientas (colores, estilos de línea, fuentes, etc.) y las operaciones de "cambio de estado" cambian uno de los actuales. herramientas (cambiar el color de relleno de lo que fuera a azul, o cambiar la fuente actual a Times-Roman en 15 puntos, por ejemplo).

El documento generado por el programa "hola mundo" mencionado anteriormente contendría los siguientes gráficos.

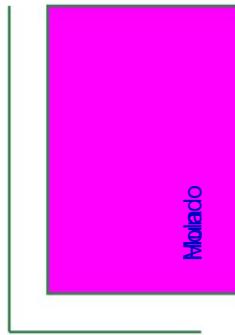


Figura 2-1: "Hola mundo" en pdfgen

Acerca de las demostraciones en este documento

Este documento contiene demostraciones del código analizado como el que se muestra en el rectángulo de arriba.

Estas demostraciones están dibujadas en una "pequeña página" integrada dentro de las páginas reales de la guía. Las pequeñas páginas miden 5,5 pulgadas de ancho y 3 pulgadas de alto. Las pantallas de demostración muestran la salida real del código de demostración. Por conveniencia, el tamaño de la salida se ha reducido ligeramente.

2.4 Las herramientas: las operaciones de "dibujo"

Esta sección enumera brevemente las herramientas disponibles en el programa para pintar información en una página usando la interfaz canvas. Estos se discutirán en detalle en secciones posteriores. Se enumeran aquí para facilitar su consulta y con fines resumidos.

Métodos de línea

```
lienzo.line(x1,y1,x2,y2)
```

```
lienzo.lines(lista de líneas)
```

Los métodos de línea dibujan segmentos de línea recta en el lienzo.

Métodos de forma

```
lienzo.grid (listax, listay)
```

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

```
lienzo.arc(x1,y1,x2,y2)
```

```
lienzo.rect(x, y, ancho, alto, trazo=1, relleno=0)
```

```
lienzo.elipse(x1,y1, x2,y2, trazo=1, relleno=0)
```

```
canvas.wedge(x1,y1, x2,y2, startAng, extensión, trazo=1, relleno=0)
```

```
lienzo.circle(x_cen, y_cen, r, trazo=1, relleno=0)
```

```
canvas.roundRect(x, y, ancho, alto, radio, trazo=1, relleno=0)
```

Los métodos de formas dibujan formas complejas comunes en el lienzo.

Métodos de dibujo de cuerdas

```
lienzo.drawString(x, y, texto):
```

```
lienzo.drawRightString(x, y, texto)
```

```
lienzo.drawCentredString(x, y, texto)
```

Los métodos de dibujo de cadena dibujan líneas individuales de texto en el lienzo.

Los métodos del objeto de texto.

```
objeto de texto = lienzo.beginText(x, y)
```

```
lienzo.drawText(objeto de texto)
```

Los objetos de texto se utilizan para dar formato al texto de formas que no son compatibles directamente con la interfaz del lienzo. Un programa crea un objeto de texto desde el lienzo usando beginText y luego da formato al texto invocando métodos de objetos de texto. Finalmente, el objeto de texto se dibuja en el lienzo usando drawText.

Los métodos del objeto de ruta.

```
ruta = lienzo.beginPath()
```

```
canvas.drawPath(ruta, trazo=1, relleno=0, modo de relleno=Ninguno)
```

```
canvas.clipPath(ruta, trazo=1, relleno=0, modo de relleno=Ninguno)
```

Los objetos de ruta son similares a los objetos de texto: proporcionan control dedicado para realizar tareas gráficas complejas. dibujo no proporcionado directamente por la interfaz del lienzo. Un programa crea un objeto de ruta usando beginPath llena la ruta con gráficos usando los métodos del objeto de ruta y luego dibuja la ruta en el lienzo

usando drawPath.

También es posible utilizar una ruta como "región de recorte" usando el método clipPath (por ejemplo, una ruta circular).

La ruta se puede utilizar para recortar las partes exteriores de una imagen rectangular dejando solo una parte circular de la imagen visible en la página.

Si se especifica fill=1, entonces el argumento fillMode se puede usar para establecer 0=par-impar o 1=modo de llenado distinto de cero. lo que alterará la forma en que se llenan los caminos complejos. Si los valores predeterminados Ninguno se usa, entonces se usa el valor del atributo canvas._fillMode (normalmente 0, es decir, par-impar).

Métodos de imagen



Necesita la biblioteca de imágenes de Python (PIL) para usar imágenes con el paquete ReportLab. Ejemplos de la Las técnicas siguientes se pueden encontrar ejecutando el script test_pdfgen_general.py en nuestro subdirectorio de pruebas y consultando la página 7 del resultado.

Hay dos formas que suenan similares de dibujar imágenes. El preferido es el método drawImage. Esto implementa un sistema de almacenamiento en caché para que pueda definir una imagen una vez y dibujarla muchas veces; solo se almacenará una vez en el archivo PDF. drawImage también expone un parámetro avanzado, una máscara de transparencia, y expondrá más en el futuro. La técnica más antigua, drawInlinelImage, almacena mapas de bits dentro del flujo de página y se por lo tanto, es muy ineficaz si utiliza la misma imagen más de una vez en un documento; pero puede generar archivos PDF que renderice más rápido si las imágenes son muy pequeñas y no se repiten. Primero discutiremos el más antiguo:

```
canvas.drawInlinelImage(self, imagen, x,y, ancho=Ninguno,alto=Ninguno)
```

El método drawInlinelImage coloca una imagen en el lienzo. El parámetro de imagen puede ser un PIL Objeto de imagen o nombre de archivo de imagen. Se aceptan muchos formatos de archivo comunes, incluidos GIF y JPEG. Devuelve el tamaño de la imagen real en píxeles como una tupla (ancho, alto).

```
canvas.drawImage(self, imagen, x,y, ancho=Ninguno,alto=Ninguno,máscara=Ninguno)
```

Los argumentos y el valor de retorno funcionan como para drawInlinelImage. Sin embargo, utilizamos un sistema de almacenamiento en caché; una imagen determinada solo se almacenará la primera vez que se use y solo se hará referencia a ella en usos posteriores. Si proporciona un nombre de archivo, se supone que el mismo nombre de archivo significa la misma imagen. Si proporciona una imagen PIL, prueba si la El contenido realmente ha cambiado antes de volver a incrustarlo.

El parámetro de máscara le permite crear imágenes transparentes. Toma 6 números y define el rango de valores RGB que serán enmascarados o tratados como transparentes. Por ejemplo con [0,2,40,42,136,139], enmascarará elimine cualquier píxel con un valor Rojo de 0 o 1, Verde de 40 o 41 y Azul de 136, 137 o 138 (en una escala de 0-255). Actualmente es tu trabajo saber qué color es el "transparente" o el de fondo.

PDF permite muchas funciones de imagen y expondremos más con el tiempo, probablemente con palabras clave adicionales. argumentos para dibujarImagen.

Finalizar una página

```
lienzo.showPage()
```

El método showPage finaliza la página actual. Todo dibujo adicional se realizará en otra página.



¡Advertencia! Todos los cambios de estado (cambios de fuente, configuraciones de color, transformaciones de geometría, etc.) están OLVIDADOS cuando avanza a una nueva página en pdfgen. Cualquier configuración de estado que desee conservar debe configurarse nuevamente ¡Antes de que el programa continúe con el dibujo!

2.5 La caja de herramientas: las operaciones de "cambio de estado"

Esta sección enumera brevemente las formas de cambiar las herramientas utilizadas por el programa para pintar información en un página usando la interfaz de lienzo. Estos también se discutirán en detalle en secciones posteriores.

Cambiando colores

```
lienzo.setFillColorCMYK(c, m, y, k)
```

```
lienzo.setStrokeColorCMYK(c, m, y, k)
```

```
lienzo.setFillColorRGB(r, g, b)
```

```
lienzo.setStrokeColorRGB(r, g, b)
```

```
canvas.setFillColor(color)
```

```
canvas.setStrokeColor(color)
```

```
lienzo.setFillGray(gris)
```

```
lienzo.setStrokeGray(gris)
```

PDF admite tres modelos de color diferentes: nivel de grises, aditivo (rojo/verde/azul o RGB) y sustractivo con parámetro de oscuridad (cian/magenta/amarillo/oscuridad o CMYK). Los paquetes ReportLab también proporcionan nombres colores como el verde césped. Hay dos parámetros de color básicos en el estado de los gráficos: el color de relleno para el interior de figuras gráficas y el color de Trazo para el límite de figuras gráficas. Los métodos anteriores admite la configuración del color de relleno o trazo utilizando cualquiera de las cuatro especificaciones de color.

Cambiar fuentes

```
canvas.setFont(psfontname, tamaño, interlineado = Ninguno)
```

El método `setFont` cambia la fuente del texto actual a un tipo y tamaño determinados. El parámetro principal especifica la distancia hacia abajo que se debe mover al avanzar de una línea de texto a la siguiente.

Cambiar estilos de línea gráfica

```
lienzo.setLineWidth(ancho)
```

```
lienzo.setLineCap(modo)
```

```
lienzo.setLineJoin(modo)
```

```
lienzo.setMiterLimit(límite)
```

```
lienzo.setDash(self, matriz=[], fase=0)
```

Las líneas dibujadas en PDF se pueden presentar en varios estilos gráficos. Las líneas pueden tener diferentes anchos, pueden terminar en diferentes estilos de tapa, pueden unirse en diferentes estilos de unión y pueden ser continuos o pueden ser punteado o rayado. Los métodos anteriores ajustan estos diversos parámetros.

Cambiando de geometría

```
lienzo.setPageSize(par)
```

```
lienzo.transform(a,b,c,d,e,f):
```

```
lienzo.translate(dx, dy)
```

```
lienzo.escala(x, y)
```

```
lienzo.rotar(theta)
```

```
lienzo.skew(alfa, beta)
```

Todos los dibujos en PDF caben en un tamaño de página específico. Los elementos dibujados fuera del tamaño de página especificado no son visibles. Además, todos los elementos dibujados pasan por una transformación afín que puede ajustar su ubicación y/o distorsionar su apariencia. El método setPageSize ajusta el tamaño de la página actual. Los métodos de transformar, trasladar, escalar, rotar y sesgar añaden transformaciones adicionales a la transformación actual. Es importante recordar que estas transformaciones son incrementales: una nueva transformación modifica la transformación actual (pero no la reemplaza).

Control del Estado

```
lienzo.saveState()
```

```
lienzo.restoreState()
```

Muy a menudo es importante guardar la fuente actual, la transformación de los gráficos, los estilos de línea y otros estados de los gráficos para poder restaurarlos más tarde. El método saveState marca el estado actual de los gráficos para su posterior restauración mediante un recoveryState coincidente. Tenga en cuenta que la invocación del método de guardar y restaurar debe coincidir: una llamada de restauración restaura el estado al estado guardado más recientemente que aún no se ha restaurado. Sin embargo, no puede guardar el estado en una página y restaurarlo en la siguiente; no se conserva ningún estado entre páginas.

2.6 Otros métodos de lienzo .

No todos los métodos del objeto lienzo encajan en las categorías "herramienta" o "caja de herramientas". A continuación se muestran algunos de los desajustes, incluidos aquí para que estén completos.

```
canvas.setAuthor()
lienzo.addOutlineEntry(título, clave, nivel=0, cerrado=Ninguno) lienzo.setTitle(título)
lienzo.setSubject(subj)
lienzo.pageHasData()
lienzo.showOutline()
lienzo.bookmarkPage(nombre)
lienzo .bookmarkHorizontalAbsolute(nombre,
yhorizontal) canvas.doForm() canvas.beginForm(nombre, lowerx=0, lowery=0,
Upperx=Ninguno,
Uppery=Ninguno) canvas.endForm() lienzo.linkAbsolute(contenido, nombre de destino, Rect=Ninguno ,
addtopage=1,
nombre=Ninguno, **kw) lienzo.linkRect(contenido, nombre de destino, Rect=Ninguno, addtopage=1, relativo=1, nombre=Ninguno, **kw)
lienzo.getPageNumber() lienzo.addLiteral( ) canvas.getAvailableFonts() canvas.stringWidth(self, text, fontName, fontSize, encoding=Ninguno)
canvas.setPageCompression(onoff=1)
canvas setPageTransition(self,
effectname=Ninguno, duración=1,
dirección=0,dimensión= 'H',movimiento='l')
```

2.7 Coordenadas (espacio de usuario predeterminado)

De forma predeterminada, las ubicaciones de una página se identifican mediante un par de números. Por ejemplo, el par (4,5*pulgada, 1*pulgada) identifica la ubicación que se encuentra en la página comenzando en la esquina inferior izquierda y moviéndose hacia la derecha 4,5 pulgadas y una pulgada hacia arriba.

Por ejemplo, la siguiente función dibuja varios elementos en un lienzo.

```
def coords(lienzo): desde
    reportlab.lib.units importar pulgadas desde reportlab.lib.colors
    importar rosa, negro, rojo, azul, verde
    c = lienzo
    c.setStrokeColor(rosa)
    c.grid([pulgadas, 2*pulgadas, 3*pulgadas, 4*pulgadas], [0,5*pulgadas, pulgadas, 1,5*pulgadas, 2*pulgadas, 2,5*pulgadas])
    c.setStrokeColor( negro)
    c.setFont("Times-Roman", 20) c.drawString(0,0,
"(0,0) el origen") c.drawString(2,5*pulgada, pulgada, "(2,5,1")
en pulgadas ") c.drawString(4*pulgadas, 2,5*pulgadas, "(4, 2,5)") c.setFillColor(rojo)
```

```
c.rect(0,0.2*pulgada,0.2*pulgada,0.3*pulgada, relleno=1)
c.setFillColor(verde)
c.circle(4.5*pulgada, 0.4*pulgada, 0.2*pulgada, relleno=1)
```

En el espacio de usuario predeterminado, el punto "origen" (0,0) está en la esquina inferior izquierda. Ejecutando la función coords en el espacio de usuario predeterminado (para la "minipágina de demostración") obtenemos lo siguiente.



Figura 2-2: El sistema de coordenadas

Mover el origen: el método de traducción A menudo

es útil "mover el origen" a un nuevo punto en la esquina inferior izquierda. El método canvas.translate(x,y) mueve el origen de la página actual al punto actualmente identificado por (x,y).

Por ejemplo, la siguiente función de traducción primero mueve el origen antes de dibujar los mismos objetos como se muestra arriba.

```
def traducir(lienzo): desde
    reportlab.lib.units importar cm lienzo.translate(2.3*cm,
    0.3*cm) coords(lienzo)
```

Esto produce lo siguiente.



Figura 2-3: Mover el origen: el método de traducción



Nota: Como se ilustra en el ejemplo, es perfectamente posible dibujar objetos o partes de objetos "fuera de la página". En particular, un error confuso común es una operación de traducción que traduce todo el dibujo fuera del área visible de la página. Si un programa produce una página en blanco, es posible que todos los objetos dibujados estén fuera de la página.

Reducir y crecer: la operación de escala

Otra operación importante es el escalado. La operación de escala `canvas.scale(dx,dy)` estira o reduce las dimensiones x e y según los factores dx, dy respectivamente. A menudo, dx y dy son iguales; por ejemplo, para reducir un dibujo a la mitad en todas las dimensiones, utilice `dx = dy = 0,5`. Sin embargo, con fines ilustrativos, mostramos un ejemplo en el que dx y dy son diferentes.

```
escala def (lienzo):
    lienzo.escala (0,75, 0,5) coordenadas
    (lienzo)
```

Esto produce una versión reducida "corta y gruesa" de las operaciones mostradas anteriormente.



Figura 2-4: Escalar el sistema de coordenadas



Nota: el escalado también puede mover objetos o partes de objetos fuera de la página, o puede hacer que los objetos "se reduzcan a nada".

El escalado y la traducción se pueden combinar, pero el orden de las operaciones es importante.

```
def scaletranslate(canvas): de
    reportlab.lib.units importa pulgadas canvas.setFont("Courier-
BoldOblique", 12) # guarda el estado canvas.saveState() # escala
y luego traduce
    canvas.scale(0.3, 0.5)
    canvas.translate (2.4*pulgadas,
1.5*pulgadas) canvas.drawString(0,
2.7*pulgadas, "Escalar y luego traducir") coords(canvas) #
olvidar la escala y traducir... canvas.restoreState() # traducir y luego escalar
    canvas.translate
    (2.4*pulgadas, 1.5*pulgadas) canvas.scale(0.3, 0.5)
    canvas.drawString(0, 2.7*pulgadas,
"Traducir y escalar") coords(canvas)
```

Esta función de ejemplo primero guarda el estado actual del lienzo y luego realiza una escala seguida de una traducción. Luego, la función restaura el estado (eliminando efectivamente los efectos de la escala y la traducción) y luego realiza las mismas operaciones en un orden diferente. Observe el efecto a continuación.

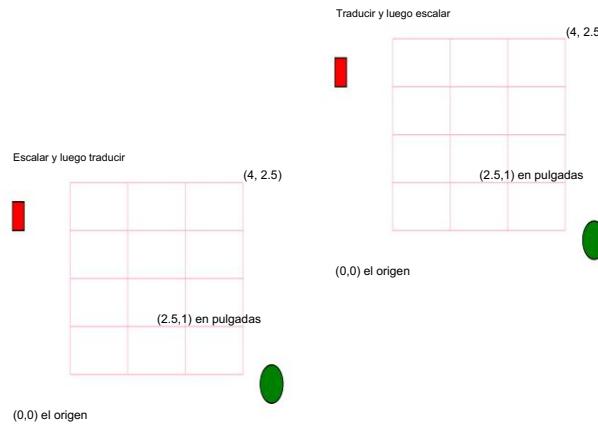


Figura 2-5: Escalado y traducción



Nota: la escala reduce o aumenta todo, incluidos los anchos de línea, por lo que usar el método `canvas.scale` para representar un dibujo microscópico en unidades microscópicas escaladas puede producir una mancha (porque todos los anchos de línea se expandirán enormemente). Además, renderizar el ala de un avión en metros escalados a centímetros puede hacer que las líneas se reduzcan hasta el punto de desaparecer. Para fines científicos o de ingeniería como estos, escale y traslade las unidades externamente antes de renderizarlas usando el lienzo.

Guardar y restaurar el estado del lienzo : `saveState` y `restoreState`

La función `scaletranslate` utilizó una característica importante del objeto del lienzo: la capacidad de guardar y restaurar los parámetros actuales del lienzo. Al encerrar una secuencia de operaciones en un par coincidente de operaciones `canvas.saveState()` y `canvas.restoreState()`, todos los cambios de fuente, color, estilo de línea, escala, traducción u otros aspectos del estado de los gráficos del lienzo se pueden restaurar al estado actual. esto en el punto de `saveState()`. Recuerde que las llamadas de guardar/restaurar deben coincidir: una operación de guardar o restaurar perdida puede provocar un comportamiento inesperado e indeseable. Además, recuerde que no se conserva ningún estado del lienzo en los saltos de página y que el mecanismo de guardar/restaurar no funciona en los saltos de página.

Imagen de espejo

Es interesante, aunque quizás no demasiado útil, señalar que los factores de escala pueden ser negativos. Por ejemplo la siguiente función

```
def espejo(lienzo): desde
    reportlab.lib.units importe pulgadas
    lienzo.translate(5.5*pulgadas, 0)
    lienzo.escala(-1.0, 1.0) coords(lienzo)
```

crea una imagen especular de los elementos dibujados por la función `coord`.

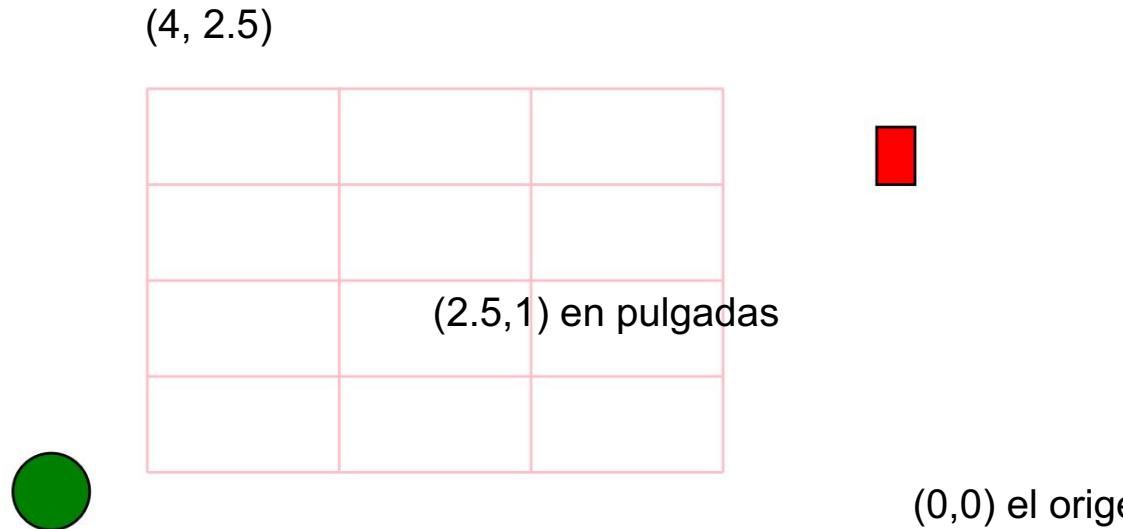


Figura 2-6: Imágenes reflejadas

Observe que las cadenas de texto están pintadas al revés.

2,8 colores

Generalmente se utilizan dos tipos de colores en PDF según el medio donde se utilizará el PDF. El modelo de colores de pantalla más conocido, RGB, se puede utilizar en PDF; sin embargo, en la impresión profesional se utiliza principalmente otro modelo de color, CMYK, que proporciona más control sobre cómo se aplican las tintas al papel. Más información sobre estos modelos de color a continuación.

Colores RGB

La representación RGB o de color aditivo sigue la forma en que una pantalla de computadora agrega diferentes niveles de luz roja, verde y azul para crear cualquier color intermedio, donde el blanco se forma al encender las tres luces por completo (1,1,1).

Hay tres formas de especificar colores RGB en pdfgen: por nombre (usando el módulo de color, por valor rojo/verde/azul (aditivo, RGB) o por nivel de gris. La función de colores a continuación ejercita cada uno de los cuatro métodos.

```
def coloresRGB(lienzo):
    desde reportlab.lib importar colores desde reportlab.lib.units
    importar pulgadas negro = colores.negro y = x = 0; dy=pulgada*3/4,0;
    dx=pulgada*5,5/5; w=h=dy/2; rdx=(dx-
    w)/2 rdy=h/5,0; texty=h+2*rdy canvas.setFont("Helvetica",10) para [namedcolor, nombre] en ( [colors.lavenderblush,
    "lavenderblush"], [colors.lawngreen,
    "lawngreen"], [colors.lemonchiffon, "lemonchiffon"],
    [colors.lightblue, "lightblue"], [colors.lightcoral,
    "lightcoral"]):

        canvas.setFillColor(namedcolor) canvas.rect(x+rdx, y+rdy,
        w, h, fill=1) canvas.setFillColor(negro) canvas.drawCentredString(x+dx/2,
        y+texty, nombre)

        x = x+dx y = y
        + dy; x = 0 para rgb en [(1,0,0),
        (0,1,0), (0,0,1), (0,5,0,3,0,1), (0,4,0,5,0,3)]:
            r,g,b = rgb
            lienzo.setFillColorRGB(r,g,b) lienzo.rect(x+rdx, y+rdy, w,
            h, relleno=1) lienzo.setFillColor(negro)
```

```

    canvas.drawCentredString(x+dx/2, y+texty, "r%sg%s b%s"%rgb)
    x = x+dx

    y = y + dy; x = 0
    para gris en (0,0, 0,25, 0,50, 0,75, 1,0):
        lienzo.setFillGray(gris)
        lienzo.rect(x+rdx, y+rdy, w, h, relleno=1)
        lienzo.setFillColor(negro)
        canvas.drawCentredString(x+dx/2, y+texty, "gris: %s"%gray)
        x = x+dx

```

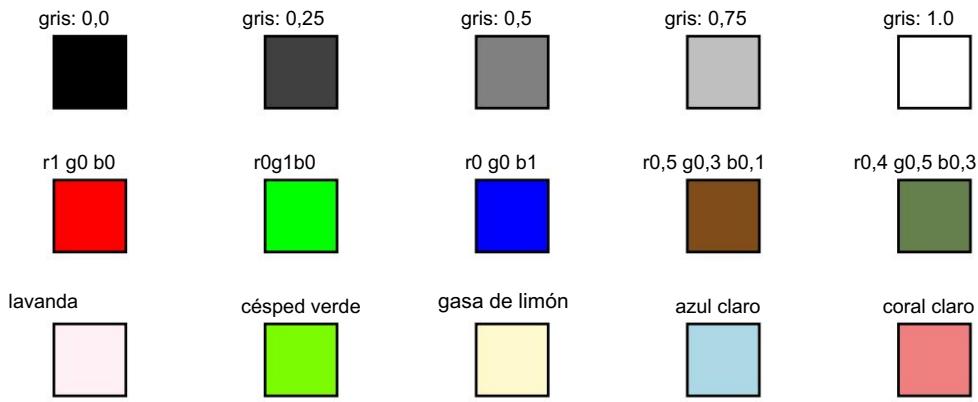


Figura 2-7: Modelos de color RGB

Transparencia de color RGB

Los objetos se pueden pintar sobre otros objetos con buenos resultados en pdfgen. Generalmente hay dos modos de manejar objetos que se superponen en el espacio, los objetos predeterminados en la capa superior ocultarán cualquier parte de otros objetos que cae debajo de él. Si necesitas transparencia, tienes dos opciones:

1. Si su documento está destinado a ser impreso de forma profesional y está trabajando en color CMYK espacio entonces puedes usar `overPrint`. Al sobreimprimir, los colores se mezclan físicamente en la impresora y, por lo tanto, se crea un nuevo color. es obtenido. De forma predeterminada, se aplicará un knockout y solo aparecerá el objeto superior. Lea la sección CMYK si esto es lo que pretendes utilizar.
2. Si su documento está diseñado para salida en pantalla y está utilizando colores RGB, puede establecer un valor alfa. valor, donde alfa es el valor de opacidad del color. El valor alfa predeterminado es 1 (completamente opaco) y puede Utilice cualquier valor de número real en el rango 0-1.

La transparencia alfa (alfa) es similar a la sobreimpresión pero funciona en el espacio de color RGB, en el siguiente ejemplo demuestra la funcionalidad alfa. Consulte nuestro sitio web <http://www.reportlab.com/snippets/> y busque fragmentos de `overPrint` y `alpha` para ver el código que genera el siguiente gráfico.

```

def alfa(lienzo):
    desde reportlab.graphics.shapes importar Rect
    de reportlab.lib.colors importar Color, negro, azul, rojo
    rojo50transparente = Color( 100, 0, 0, alfa=0.5)
    c = lienzo
    c.setFillColor(negro)
    c.setFont('Helvética', 10)
    c.drawString(25,180, 'sólido')
    c.setFillColor(azul)
    c.rect(25,25,100,100, relleno=True, trazo=False)
    c.setFillColor(rojo)
    c.rect(100,75,100,100, relleno=True, trazo=False)
    c.setFillColor(negro)
    c.drawString(225,180, 'transparente')

```

```
c.setFillColor(azul) c.rect(225,25,100,100,
relleno=Verdadero, trazo=False) c.setFillColor(rojo50transparent) c.rect(300,75,100,100,
relleno=Verdadero, trazo=False)
```

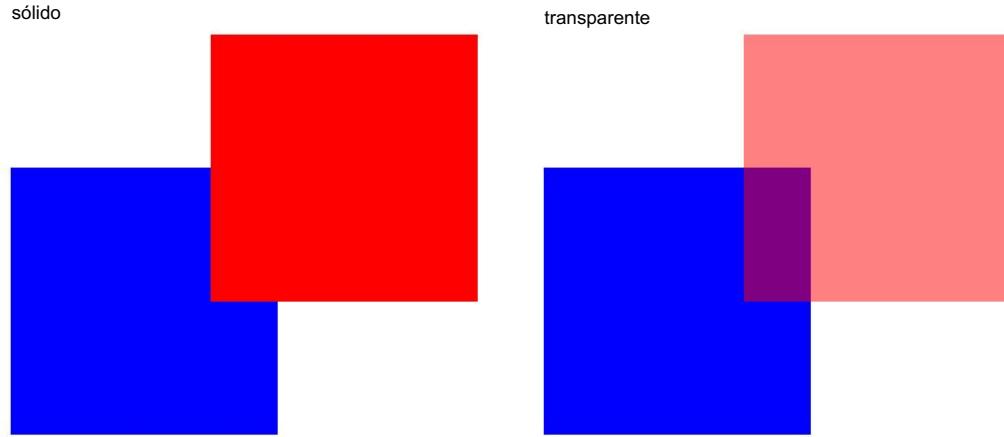


Figura 2-8: Ejemplo alfa

Colores CMYK

El método CMYK o sustractivo sigue la forma en que una impresora mezcla tres pigmentos (cian, magenta y amarillo) para formar colores. Dado que mezclar productos químicos es más difícil que combinar luz, existe un cuarto parámetro para la oscuridad. Por ejemplo, una combinación química de pigmentos CMY generalmente nunca produce un negro perfecto, sino que produce un color turbio, por lo que, para obtener impresoras negras, no use pigmentos CMY, sino tinta negra directa. Debido a que CMYK se corresponde más directamente con la forma en que funciona el hardware de la impresora, puede darse el caso de que los colores especificados en CMYK proporcionen una mejor fidelidad y un mejor control al imprimir.

Hay dos formas de representar el color CMYK: cada color se puede representar mediante un valor real entre 0 y 1 o un valor entero entre 0 y 100. Dependiendo de sus preferencias, puede usar CMYKColor (para valores reales) o PCMYKColor (para valores enteros). 0 significa "sin tinta", por lo que imprimir en papeles blancos produce blanco. 1 (o 100 si usa PCMYKColor) significa "la cantidad máxima de tinta". p.ej CMYKColor(0,0,0,1) es negro, CMYKColor(0,0,0,0) significa "sin tinta" y CMYKColor(0.5,0,0,0) significa 50 por ciento de color cian.

```
def coloresCMYK(lienzo): de reportlab.lib.colors
    importa CMYKColor, PCMYKColor de reportlab.lib.units importa pulgadas # crea un CMYK negro;
    CMYKColor usa valores reales black = CMYKColor(0,0,0,1) # crea un
    CMYK cian; PCMYKColor utiliza valores enteros cian = PCMYKColor(100,0,0,0) y = x = 0;
    dy=pulgada*3/4;0; dx=pulgada*5/5; w=h=dy/2;
    rdx=(dx-w)/2 rdy=h/5.0; texty=h+2*rdy canvas.setFont("Helvetica",10) y = y + dy; x = 0 para cmyk en
    [(1,0,0,0), (0,1,0,0), (0,0,1,0), (0, 0,0,0)]:

    c,m,y1,k = cmyk
    lienzo.setFillColorCMYK(c,m,y1,k) lienzo.rect(x+rdx, y+rdy, w, h,
    relleno=1) lienzo.setFillColor(negro) lienzo.drawCentredString (x+dx/2,
    y+texty, "%sm%sy%s k%s%cmyk) x = x+dx
```

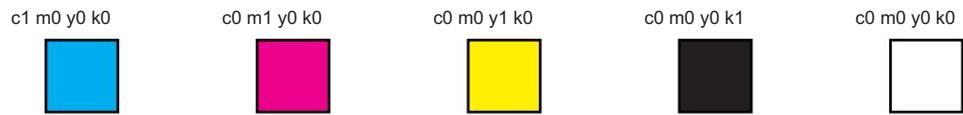


Figura 2-9: Modelos de color CMYK

2.9 Comprobación del espacio de color

El argumento enforceColorSpace del lienzo se utiliza para imponer la coherencia del modelo de color utilizado en un documento. Acepta estos valores: CMYK, RGB, SEP, SEP_BLACK, SEP_CMYK. 'SEP' se refiere a separaciones de colores con nombre, como colores directos Pantone, que se pueden mezclar con CMYK o RGB según el parámetro utilizado. El valor predeterminado es 'MIXED', que le permite utilizar colores de cualquier espacio de color. Se plantea una excepción si alguno de los colores utilizados no es convertible al modelo especificado, por ejemplo, rgb y cmyk (más información en test_pdffgen_general). Este enfoque no verifica las imágenes externas incluidas en el documento.

2.10 Sobreimpresión en color

Cuando dos objetos de color CMYK se superponen en la impresión, el objeto "encima" eliminará el color del que está debajo, o los colores de los dos objetos se mezclarán en el área superpuesta. este comportamiento se puede configurar usando la propiedad overPrint.

La función sobreimprimir hará que se mezclen áreas de color superpuestas. En el siguiente ejemplo, los colores de Los rectángulos de la izquierda deberían aparecer mezclados donde se superponen. Si no puedes ver este efecto, entonces puedes Es necesario habilitar la opción 'vista previa de sobreimpresión' en su software de visualización de PDF. Algunos visores de PDF como evince no admite sobreimpresión; sin embargo, Adobe Acrobat Reader lo admite.

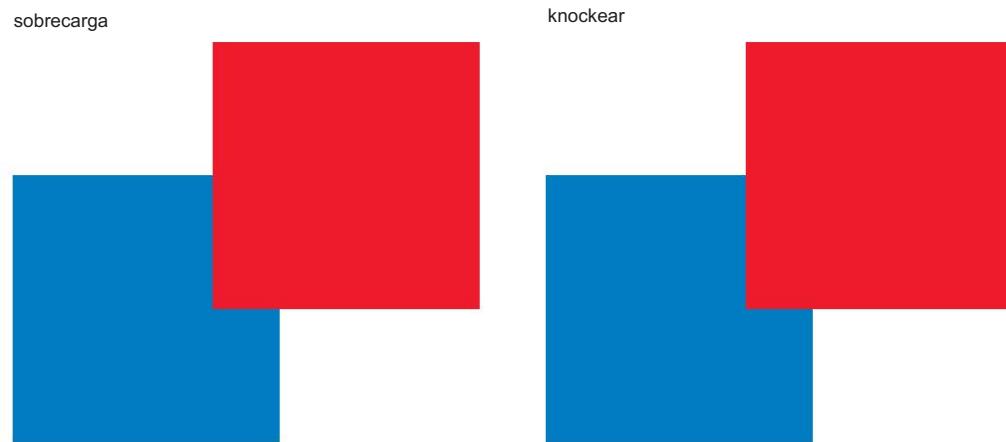


Figura 2-10: ejemplo de sobreimpresión

Otros ejemplos de orden de impresión de objetos

La palabra "SPUMONI" está pintada en blanco sobre los rectángulos de colores, con el efecto aparente de "eliminar" el color dentro del cuerpo de la palabra.

```
def spumoni(lienzo):
    importa pulgadas de reportlab.lib.colors importa rosa, verde, marrón,
    blanco x = 0; dx = 0,4*pulgada para i en el rango(4): para color en (rosa, verde, marrón): canvas.setFillColor(color)
    canvas.rect(x,0,dx,3*inch,stroke=0,fill=
    1) x = x+dx
```

```
canvas.setFillColor(blanco)
canvas.setStrokeColor(blanco) canvas.setFont("Helvetica-Bold", 85) canvas.drawCentredString(2,75*pulgadas, 1,3*pulgadas,
"SPUMONI")
```

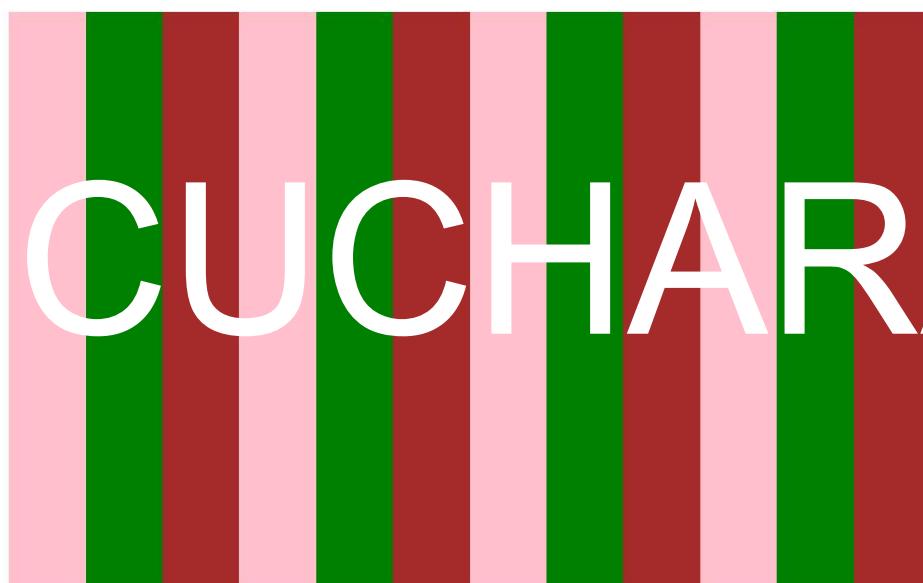


Figura 2-11: Pintar sobre colores

Las últimas letras de la palabra no son visibles porque el fondo predeterminado del lienzo es blanco y pintar letras blancas sobre un fondo blanco no deja ningún efecto visible.

Este método de construir pinturas complejas en capas se puede realizar en muchas capas en pdfgen; hay menos limitaciones físicas que cuando se trata de pinturas físicas.

```
def spumoni2(lienzo):
    de reportlab.lib.units importar pulgadas de reportlab.lib.colors importar
    rosa, verde, marrón, blanco, negro # dibuja el dibujo anterior spumoni(lienzo) # ahora pon un cono de helado encima: #
    primero dibuja un triángulo (cono de helado) p =
    lienzo.beginPath() xcenter
    = 2,75*pulgada radio = 0,45*pulgada p.moveTo(xcenter-radius, 1,5*pulgada)
    p.lineTo(xcenter+radius, 1,5*pulgada) p.lineTo( xcenter, 0)
    canvas.setFillColor(brown)
    canvas.setStrokeColor(black)
    canvas.drawPath(p, fill=1) # dibuja
    algunos círculos (cucharadas) y = 1,5*pulgadas para colorear en
    (rosa, verde, marrón): lienzo .setFillColor(color) lienzo.circle(xcenter,
    y, radio, relleno=1) y = y+radio
```

La función spumoni2 coloca un cono de helado sobre el dibujo de spumoni. Tenga en cuenta que las diferentes partes del cono y las palas también se superponen.



Figura 2-12: creación de un dibujo en capas

2.11 Fuentes estándar y objetos de texto

El texto se puede dibujar en muchos colores, fuentes y tamaños diferentes en pdfgen. La función tamaño del texto demuestra cómo cambiar el color, la fuente y el tamaño del texto y cómo colocarlo en la página.

```
def tamaño de texto (lienzo):
    de reportlab.lib.units importar pulgadas de reportlab.lib.colors importar
    magenta, rojo canvas.setFont("Times-Roman", 20) canvas.setFillColor(rojo)
    canvas.drawCentredString(2,75"inch, 2,5"inch, "Font ejemplos
    de tamaño") canvas.setFillColor(magenta)
```

```
tamaño = 7 y =
2,3*pulgada x =
1,3*pulgada para la
línea en la letra:
    canvas.setFont("Helvetica", tamaño) canvas.drawString(x,y,"%s"
    puntos: " % tamaño) canvas.drawString(x ,y, línea) y = tamaño y*1.2 tamaño = tamaño+1.5
```

La función textsize genera la siguiente página.

Ejemplos de tamaño de fuente

7 puntos: bueno, le dio a Net Solutions
 8,5 puntos: y registró su propio sitio .com. Ahora 10,0 puntos:
 y lo llenó con fotografías de perfil de Yahoo.
11,5 puntos: se comió en una noche
ahora 13,0 puntos: y ganó 50 millones cuando Hugh Hefner
14,5 puntos: compró los derechos ahora 16,0
puntos: y se divertirá, divertido, divertido 17,5 puntos:
hasta que su papá le quite el teclado

Figura 2-13: texto en diferentes fuentes y tamaños

En pdfgen siempre hay disponibles varias fuentes diferentes.

```
def fonts(canvas): de
    reportlab.lib.units import inch text = "Ahora es el momento de que todos
    los hombres buenos..." x = 1,8*pulgadas y = 2,7*pulgadas para fuente en
    canvas.getAvailableFonts():
        lienzo .setFont(fuente,
        10) lienzo.drawString(x,y,Texto) lienzo.setFont("Helvetica", 10)
        lienzo.drawString(x-10,y, fuente+":") y =
        y-13
```

La función de fuentes enumera las fuentes que siempre están disponibles. No es necesario almacenarlos en un documento PDF, ya que se garantiza que estarán presentes en Acrobat Reader.

Mensajero: Ahora es el momento de que todos los hombres buenos...
 Courier-Bold: Ahora es el momento de que todos los hombres buenos...
 Courier-BoldOblique: Ahora es el momento de que todos los hombres buenos...
 Courier-Oblique: Ahora es el momento de que todos los hombres buenos...
 Helvetica: Ahora es el momento de que todos los hombres buenos...
 Helvetica-Bold: Ahora es el momento de que todos los hombres buenos...
 Helvetica-BoldOblique: Ahora es el momento de que todos los hombres buenos...
 Helvetica-Oblique: Ahora es el momento de que todos los hombres buenos...
 Símbolo: ...
 Times-Bold: Ahora es el momento de que todos los hombres buenos...
 Times-BoldItalic: Ahora es el momento de que todos los hombres buenos...
 Times-Italic: Ahora es el momento de que todos los hombres buenos...
 Times-Roman: Ahora es el momento de que todos los hombres buenos...
 ZapfDingbats:

Figura 2-14: las 14 fuentes estándar

Las fuentes Symbol y ZapfDingbats no se pueden mostrar correctamente porque los glifos requeridos no están presentes en esas fuentes.

Para obtener información sobre cómo utilizar fuentes arbitrarias, consulte el siguiente capítulo.

2.12 Métodos de objetos de texto

Para la presentación dedicada de texto en un documento PDF, utilice un objeto de texto. La interfaz de objetos de texto proporciona un control detallado de los parámetros de diseño de texto que no están disponibles directamente en el nivel del lienzo. Además, da como resultado un PDF más pequeño que se procesará más rápido que muchas llamadas separadas a los métodos `drawString`.

```
textobject.setTextOrigin(x,y)
textobject.setTextTransform(a,b,c,d,e,f)
textobject.moveCursor(dx, dy) # desde el inicio de la LÍNEA actual
(x,y) = objetotexto.getCursor()
x = objetotexto.getX(); y = objetotexto.getY()
textobject.setFont(psfontname, tamaño, interlineado = Ninguno)
objetotexto.textOut(texto)
textobject.textLine(texto="")
textobject.textLines(cosas, recortar=1)
```

Los métodos de objetos de texto que se muestran arriba se relacionan con la geometría de texto básica.

Un objeto de texto mantiene un cursor de texto que se mueve por la página cuando se dibuja el texto. Por ejemplo, `setTextOrigin` coloca el cursor en una posición conocida y los métodos `textLine` y `textLines` mueven el cursor de texto más allá de las líneas que faltan.

```
def cursormoves1(canvas): de
reportlab.lib.units import pulgadas
textobject =
canvas.beginText() textobject.setTextOrigin(inch,
2.5*inch) textobject.setFont("Helvetica-Oblique", 14) para línea en
letras:
```

```
textobject.textLine(línea)
textobject.setFillGray(0.4)
textobject.textLines("")  
Con muchas disculpas a los Beach Boys y a cualquiera  
que encuentre este "") objetable canvas.drawText(textobject)
```

La función cursormoves se basa en el movimiento automático del cursor de texto para colocar el texto después de que se haya establecido el origen.

Bueno, llegó a Net Solutions y
registró su propio sitio .com ahora y lo llenó con fotos
de perfil de Yahoo que se comió en una noche
y ganó 50 millones cuando Hugh
Hefner compró los derechos ahora y se divertirá,
divertido, divertido hasta su
papá le quita el teclado. Con
muchas disculpas a los Beach Boys y a cualquiera
que encuentre esto objetable.

Figura 2-15: Cómo se mueve el cursor de texto

También es posible controlar el movimiento del cursor de forma más explícita utilizando el método moveCursor (que mueve el cursor como un desplazamiento desde el inicio de la línea actual , NO el cursor actual, y que también tiene desplazamientos y positivos hacia abajo (en contraste a la geometría normal donde la y positiva generalmente se mueve hacia arriba).

```
def cursormoves2(canvas): de
    reportlab.lib.units importar pulgadas textobject =
    canvas.beginText() textobject.setTextOrigin(2,
    2.5*inch) textobject.setFont("Helvetica-Oblique", 14) para
    línea en letras: textobject. textOut(line) textobject.moveCursor(14,14)
    # ¡¡¡POSITIVA Y se mueve
    # hacia abajo!!!
    textobject.setFillColorRGB(0.4,0,1) textobject.textLines("")
```

Con muchas disculpas a los Beach Boys y a cualquiera
que encuentre este "") objetable canvas.drawText(textobject)

Aquí, textOut no mueve una línea hacia abajo en contraste con la función textLine que sí se mueve hacia abajo.

Bueno, llegó a Net Solutions y
registró su propio sitio .com ahora y lo llenó con
fotografías de perfil de Yahoo que se comió en
una noche y ganó 50 millones cuando
Hugh Hefner compró los derechos.

y ella se divertirá, divertirá,
divertirá hasta que su papá le quite el teclado.
Con muchas disculpas a los Beach Boys y a
cualquiera que encuentre esto objetable.

Figura 2-16: Cómo se mueve nuevamente el cursor de texto

Espaciado entre caracteres

```
textobject.setCharSpace(charSpace)
```

El método setCharSpace ajusta uno de los parámetros del texto: el espacio entre caracteres.

```
def charspace(canvas): de
    reportlab.lib.units importar pulgadas textobject =
        canvas.beginText() textobject.setTextOrigin(3,
            2.5*inch) textobject.setFont("Helvetica-Oblique", 10)
        charspace = 0 para línea en letras : textobject.setCharSpace(charspace)
        textobject.textLine("%s" %(charspace,line))
        charspace = charspace+0.5

    textobject.setFillGray(0.4)
    textobject.textLines("")
    Con muchas disculpas a los Beach Boys y a cualquiera
    que encuentre este "") objetable canvas.drawText(textobject)
```

La función charspace ejerce varias configuraciones de espaciado. Produce la siguiente página.

0: bueno, llegó a Net Solutions 0.5: y
registró su propio sitio .com ahora 1.0: y lo llenó con fotos de perfil de
Yahoo 1.5: se zambulló en una noche ahora 2.0: y ganó 50 millones
cuando Hugh Hefner 2.5: compró los derechos ahora 3.0: y
ella se divertirá, divertido, divertido 3.5: hasta que su papá le quite el teclado Con muchas
disculpas a los Beach Boys y a cualquiera que encuentre esto
objetable

Figura 2-17: Ajuste del espaciado entre caracteres

Espaciado de palabras

```
textobject.setWordSpace(palabraSpace)
```

El método setWordSpace ajusta el espacio entre palabras.

```
def espacio de palabras (lienzo):
    de reportlab.lib.units importar pulgadas textobject =
    canvas.beginText() textobject.setTextOrigin(3,
    2.5*inch) textobject.setFont("Helvetica-Oblique", 12)
    espacio de palabras = 0 para línea en letras :

    textobject.setWordSpace(espacio
        de palabras) textobject.textLine("%s: %s" %(espacio
        de palabras, línea)) espacio de palabras = espacio de palabras+2.5
        textobject.setFillColorCMYK(0.4,0,0.4,0.2)
    textobject.textLines("")
```

Con muchas disculpas a los Beach Boys y a cualquiera
que encuentre este "") objetable canvas.drawText(textobject)

La función de espacio de palabras muestra cómo se ven varias configuraciones de espacio de palabras a continuación.

0: bueno, llegó a Net Solutions 2.5:
y ahora registró su propio sitio .com 5.0: y lo llenó con fotos de
perfil de Yahoo 7.5: se comió una noche ahora 10.0: y ganó 50
millones cuando Hugh Hefner 12.5: compró los
derechos ahora 15.0: y ella se divertirá, divertido, divertido 17.5: Con muchas
disculpas a los Beach Boys y a cualquiera que
encuentre esto.
hasta que su papá le quita el teclado

objetable

Figura 2-18: Ajuste del espaciado entre palabras

Escala horizontal

```
textobject.setHorizScale(horizScale)
```

Las líneas de texto se pueden estirar o reducir horizontalmente mediante el método setHorizScale.

```
def escala_horizontal (lienzo):
    de reportlab.lib.units import inch
    textobject = canvas.beginText()
    textobject.setTextOrigin(3, 2.5*inch)
    textobject.setFont("Helvetica-Oblique", 12)
    horizontalscale = 80 # 100
    es el valor predeterminado para la línea en la letra:
    textobject.setHorizScale(escalahorizontal)
    textobject.textLine("%s: %s" %(escalahorizontal, linea))
    escalahorizontal = escalahorizontal+10
    textobject.setFillColorCMYK(0.0,0.4,0.4,0.2)
    textobject.textLines("")
```

Con muchas disculpas a los Beach Boys y a cualquiera que encuentre
este "") objetable canvas.drawText(textobject)

El parámetro de escala horizontal horizScale se proporciona en porcentajes (con 100 como valor predeterminado), por lo que la configuración de 80 que se muestra a continuación parece delgada.

80: bueno, llegó a Net
 Solutions 90: y ahora registró su propio sitio .com
 100: y lo llenó con fotos de perfil de Yahoo 110:
 ahora se zambulló en una noche 120: y ganó
 50 millones cuando Hugh Hefner 130: compró los derechos ahora 140: y
 ella se divertirá, divertido, divertido 150: hasta
 que su papá le quite el teclado **Con muchas disculpas**
 a los Beach Boys y a cualquiera que encuentre esto objetable

Figura 2-19: ajuste de la escala de texto horizontal

Interlineado (interlineado)

```
textobject.setLeading(principal)
```

El desplazamiento vertical entre el punto en el que comienza una línea y donde comienza la siguiente se denomina desplazamiento inicial. El método setLeading ajusta el desplazamiento inicial.

```
def líder (liendo): de
    reportlab.lib.units importar pulgadas textobject =
        canvas.beginText() textobject.setTextOrigin(3,
            2.5*inch) textobject.setFont("Helvetica-Oblique", 14) líder
            = 8 para línea en letras : textobject.setLeading(principal)

    textobject.textLine("%s: %s"
        %(principal,línea)) principal = principal+2.5
    textobject.setFillColorCMYK(0.8,0,0,0.3) textobject.textLines("")
```

Con muchas disculpas a los Beach Boys y a cualquiera que encuentre este "") objetable **canvas.drawText(textobject)**

Como se muestra a continuación, si el desplazamiento inicial se establece en caracteres demasiado pequeños de una línea, escribo sobre las partes inferiores de los caracteres de la línea anterior.

8: bueno, llegó a Net Solutions 10.5:
y registró su propio sitio .com ahora 13.0: y lo llenó con fotos de
perfil de Yahoo 15.5: se comió en una noche ahora 18.0:
y ganó 50 millones cuando Hugh Hefner 20.5:
compró los derechos ahora 23.0: y ella se divertirá, divertido,
divertido.

25.5: hasta que su papá le quita el teclado

Con muchas disculpas a los Beach Boys

y cualquier otra persona que encuentre esto objetable

Figura 2-20: ajuste del avance

Otros métodos de objetos de texto

```
textobject.setTextRenderMode(modo)
```

El método setTextRenderMode permite utilizar texto como primer plano para recortar dibujos de fondo, por ejemplo.

<pre>textobject.setRise(subir)</pre>	El método setRise o lowers eleva	texto en la línea (para crear superíndices o subíndices, por ejemplo).
--------------------------------------	----------------------------------	--

```
textobject.setFillColor(aColor); textobject.setStrokeColor(self,
aColor) # and similar
```

Estas operaciones de cambio de color cambian el **color** del texto y, por lo demás, son similares a los métodos de color del objeto lienzo.

2.13 Caminos y Líneas

Así como los objetos de texto están diseñados para la presentación específica de texto, los objetos de ruta están diseñados para la construcción específica de figuras gráficas. Cuando los objetos de trazado se dibujan en un lienzo, se dibujan como una figura (como un rectángulo) y se puede ajustar el modo de dibujo de toda la figura: las líneas de la figura se pueden dibujar (trazar) o no; el interior de la figura puede estar relleno o no; Etcétera.

Por ejemplo, la función estrella utiliza un objeto de ruta para dibujar una estrella.

```
def star(canvas, title="Título aquí", también conocido como="Comentar aquí.", xcenter=Ninguno,
        ycenter=Ninguno, nvertices=5): desde matemáticas importar pi desde
reportlab.lib.units importar pulgadas
radio=pulgadas/3.0 si xcenter es Ninguno: xcenter=2.75*inch si
ycenter es Ninguno:
ycenter=1.5*inch canvas.drawCentredString(xcenter, ycenter+1.3*radius,
title) canvas.drawCentredString(xcenter, ycenter-1.4*radius, también
conocido como) p = canvas.beginPath() p.moveTo(xcenter,ycenter+radius) de importación matemática pi, cos, sin
ángulo = (2*pi)*2/5.0 startangle = pi/2.0
```

```

para vértice en el rango (nvertices-1): siguienteángulo =
    ángulo*(vértice+1)+ángulo inicial x = xcetro + radio*cos(siguiente) y =
        ycetro + radio*sin(siguiente) p.lineTo(x,y) if nvertices==5: p.closePath()
    lienzo.drawPath(p)

```

La función de estrella ha sido diseñada para ser útil a la hora de ilustrar varios parámetros de estilo de línea admitidos por pdfgen.



Figura 2-21: parámetros de estilo de línea

Configuración de unión de línea

El método setLineJoin puede ajustar si los segmentos de línea se encuentran en un punto, un cuadrado o un vértice redondeado.

```

def une(lienzo): de reportlab.lib.units
    importar pulgada # hacer líneas grandes lienzo.setLineWidth(5)
    estrella(lienzo, "Predeterminado":
        unión en inglés", "0: puntiagudo", xcenter
        = 1*pulgada) lienzo.setLineJoin (1) star(canvas, "Unión redonda", "1: redondeado") canvas.setLineJoin(2) star(canvas, "Unión biselada",
        "2: cuadrado", xcenter=4.5"inch)

```

La configuración de unión de líneas sólo es de interés para líneas gruesas porque no se puede ver claramente en líneas finas.



Figura 2-22: diferentes estilos de unión de líneas

Configuración de límite de linea

La configuración del límite de línea, ajustada mediante el método `setLineCap`, determina si una línea final termina en un cuadrado exactamente en el vértice, un cuadrado sobre el vértice o un semicírculo sobre el vértice.

```
def caps(lienzo):
    reportlab.lib.units importar pulgadas # hacer líneas
    grandes
    lienzo.setLineWidth(5)
    estrella(lienzo, "Predeterminado", "sin proyección", xcenter = 1*pulgada, nvertices=4)
    lienzo.setLineCap
    (1) star(canvas, "Tapa redonda",
    "1: termina en medio círculo", nvertices=4) canvas.setLineCap(2) star(canvas, "Tapa cuadrada", "2: se
    proyecta medio ancho",
    xcentro=4,5*pulgada,
    nvértices=4)
```

La configuración de límite de línea, al igual que la configuración de unión de líneas, solo es claramente visible cuando las líneas son gruesas.



Figura 2-23: configuración del límite de linea

Guiones y líneas discontinuas

El método `setDash` permite dividir líneas en puntos o guiones.

```
def guiones(lienzo): de
    reportlab.lib.units importar pulgadas # hacer líneas
    grandes lienzo.setDash(6,3)
    estrella(lienzo, "Guiones"
simples", "6 puntos activados, 3 desactivados", xcenter = 1*pulgada) canvas.setDash(1,2) star(canvas,
"Puntos", "Uno encendido,
dos apagados") canvas.setDash([1,1,3,3,1,4,4,1], 0)
star( lienzo, "Patrón complejo", "[1,1,3,3,1,4,4,1]",
xcentro=4,5*pulgada)
```

Los patrones de guiones o puntos pueden ser un patrón repetitivo simple de encendido/apagado o pueden especificarse en un patrón repetitivo complejo.



Figura 2-24: algunos patrones de guiones

Crear figuras complejas con objetos de ruta. Las combinaciones

de líneas, curvas, arcos y otras figuras se pueden combinar en una sola figura usando objetos de ruta.

Por ejemplo, la función que se muestra a continuación construye dos objetos de ruta usando líneas y curvas. Esta función se utilizará más adelante como parte de la construcción de un icono de lápiz.

```
def punta de lápiz (lienzo, depuración = 1): desde
    reportlab.lib.colors importe tostado, negro, verde desde reportlab.lib.units importe
    pulgada u = pulgada/10.0 lienzo.setLineWidth(4) si
    depura:
        lienzo.scale(2.8, 2.8) # hazlo
        grande
        canvas.setLineWidth(1) # líneas pequeñas

        canvas.setStrokeColor(negro)
        canvas.setFillColor(bronceado) p =
        canvas.beginPath()
        p.moveTo(10*u,0)
        p.lineTo(0,5*u)
        p.lineTo(10*u,10*u )
        p.curveTo(11.5*u,10*u, 11.5*u,7.5*u, 10*u,7.5*u) p.curveTo(12*u,7.5*u,
11*u,2.5*u, 9.7 *u,2.5*u) p.curveTo(10.5*u,2.5*u, 11*u,0, 10*u,0)
        canvas.drawPath(p, trazo=1, relleno=1) canvas.setFillColor(negro )
        p = lienzo.beginPath() p.moveTo(0.5*u,
```

```

p.lineTo(4*u,3*u)
p.lineTo(5*u,4.5*u)
p.lineTo(3*u,6.5*u)
lienzo.drawPath(p, trazo=1, relleno=1) si se depura:

canvas.setStrokeColor(green) # poner en un marco de referencia canvas.grid([0.5*u,10*u,15*u],
[0.5*u,10*u])

```

Tenga en cuenta que el interior de la punta del lápiz se rellena como un solo objeto aunque esté construido a partir de varias líneas y curvas. Luego, la mina del lápiz se dibuja sobre él utilizando un nuevo objeto de ruta.

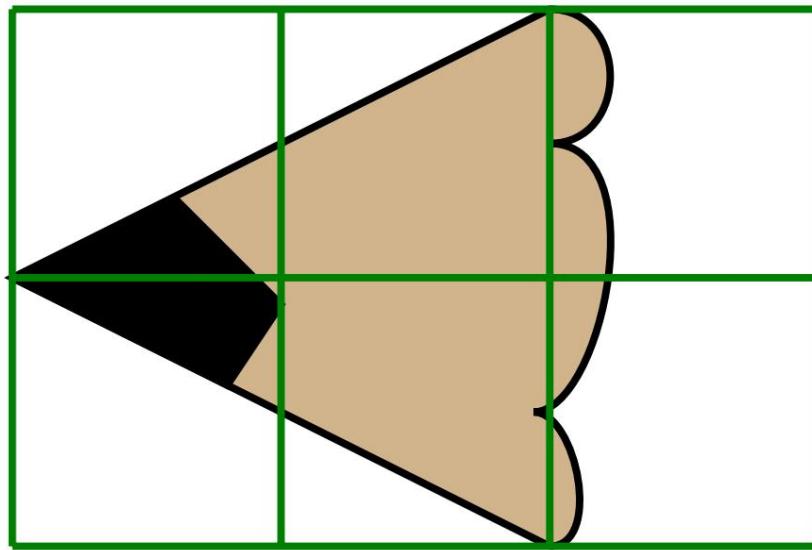


Figura 2-25: punta de un lápiz

2.14 Rectángulos, círculos, elipses

El módulo pdfgen admite una serie de formas generalmente útiles, como rectángulos, rectángulos redondeados, elipses y círculos. Cada una de estas figuras se puede utilizar en objetos de ruta o se puede dibujar directamente en un lienzo.

Por ejemplo, la siguiente función de lápiz dibuja un ícono de lápiz usando rectángulos y rectángulos redondeados con varios colores de relleno y algunas otras anotaciones.

```

def lápiz(lienzo, texto="No.2"):
    de reportlab.lib.colors importa amarillo, rojo, negro, blanco de reportlab.lib.units importa
    pulgadas u = pulgadas/10.0 canvas.setStrokeColor(black)
    canvas.setLineWidth(4)
    # dibuja borrador canvas.setFillColor(red)
    lienzo.árculo(30*u, 5*u, 5*u,
    trazo=1, relleno=1) #
    dibuja todo lo demás menos la punta
    (principalmente rectángulos con diferentes rellenos) canvas.setFillColor(amarillo)
    canvas.rect(10*u, 0.20*u,10*u, trazo=1, relleno=1) lienzo.setFillColor(negro) lienzo.rect(23*u,0.8*u,10*u,relleno=1)
    lienzo.roundRect( 14*u, 3.5*u, 8*u, 3*u,
    1.5*u, trazo=1, relleno=1) lienzo.setFillColor(blanco) lienzo.rect(25*u,u,1.2*u,8*
    u, relleno=1,trazo=0)
    canvas.rect(27.5*u,u,1.2*u,8*u, relleno=1, trazo=0)
    canvas.setFont("Times-Roman", 3*u) canvas.drawCentredString(18*u, 4*u, text) # ahora dibuja la punta
    penciltip(canvas,debug=0) # dibuja líneas
    discontinuas a lo largo del cuerpo. lienzo.setDash([10,5,16,10],0)
    lienzo.line(11*u,2.5*u,22*u,2.5*u) lienzo.line(22*u,7.5*u,12*u,7.5*u)

```



Tenga en cuenta que esta función se utiliza para crear el "lápiz de margen" a la izquierda. También tenga en cuenta que el orden en el que se dibujan los elementos es importante porque, por ejemplo, los rectángulos blancos "borran" partes de un rectángulo negro y la "punta" pinta sobre parte del rectángulo amarillo.

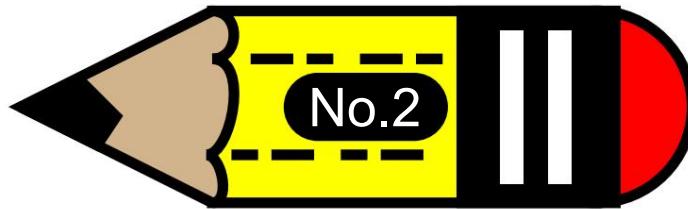


Figura 2-26: un lápiz entero

2.15 curvas de Bézier

Los programas que desean construir figuras con bordes curvos generalmente usan curvas de Bézier para formar los bordes.

```
def bezier(lienzo): de
    reportlab.lib.colors importar amarillo, verde, rojo, negro de reportlab.lib.units importar pulgadas i = pulgadas d = i/4
    # definir los puntos de control de la curva Bézier x1,y1, x2,y2 , x3,y3,
    x4,y4 = d,1.5*i,
    1.5*i,d, 3*i,d,
    5.5*i,d # dibuja una figura que encierra los puntos de control
    canvas.setFillColor(amarillo) p = canvas.beginPath() p.moveTo(x1,y1) para (x,y) en [(x2,y2), (x3,y3), (x4,y4)]: p.lineTo(x,y)

    canvas.drawPath(p, fill=1, Stroke=0) # dibuja las líneas tangentes
    canvas.setLineWidth(inch*0.1)
    canvas.setStrokeColor(green) canvas.line(x1,y1,x2,y2)
    canvas.setStrokeColor(red ) canvas.line(x3,y3,x4,y4)
    # finalmente dibuja la curva
    canvas.setStrokeColor(black) canvas.bezier(x1,y1,
    x2,y2, x3,y3, x4,y4)
```

Una curva de Bézier está especificada por cuatro puntos de control (x_1, y_1) , (x_2, y_2) , (x_3, y_3) , (x_4, y_4) . La curva comienza en (x_1, y_1) y termina en (x_4, y_4) y el segmento de recta de (x_1, y_1) a (x_2, y_2) y el segmento de recta de (x_3, y_3) a (x_4, y_4) ambos forman tangentes a la curva. Además, la curva está completamente contenida en la figura convexa con vértices en los puntos de control.

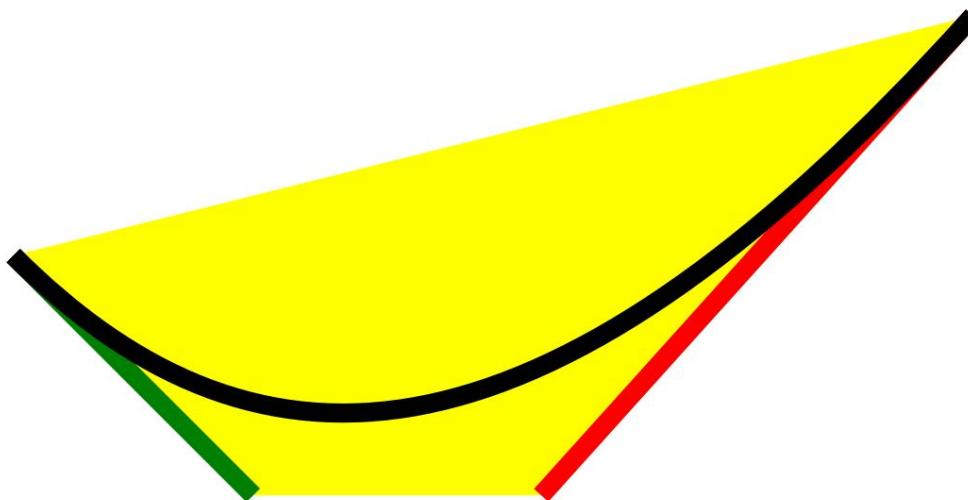


Figura 2-27: curvas Bézier básicas

El dibujo de arriba (el resultado de testbezier) muestra curvas de Bézier, las líneas tangentes definidas por los puntos de control y la figura convexa con vértices en los puntos de control.

Unir suavemente secuencias de curvas Bézier

A menudo resulta útil unir varias curvas Bézier para formar una única curva suave. Para construir una curva suave más grande a partir de varias curvas Bézier, asegúrese de que las líneas tangentes a las curvas Bézier adyacentes que se unen en un punto de control se encuentren en la misma línea.

```
def bezier2(lienzo):
    de reportlab.lib.colors importe amarillo, verde, rojo, negro de reportlab.lib.units importe
    pulgadas # haga una secuencia de puntos de control
    xd,yd = 5.5*pulgadas/2, 3*pulgadas/2 xc,yc = xd, yd
    dxdy = [(0,0,33), (0,33,0,33), (0,75,1),
    (0,875,0,875),
    (0,875,0,875), (1,0,75), (0,33,0,33), (0,33,0 )]

    lista de puntos = []
    para xoffset en (1,-1): yoffset =
        xoffset para (dx,dy) en
            dxdy: px = xc + xd*xoffset*dx
            py = yc + yd*yoffset*dy
            pointlist.append((px ,py)) yoffset =
            -xoffset para (dy,dx) en dxdy: px = xc +
            xd*xoffset*dx py = yc +
            yd*yoffset*dy
            pointlist.append((px,py)) # dibuja
            líneas tangentes y curvas
            canvas.setLineWidth(pulgadas*0.1)
    while lista de puntos: [(x1,y1),(x2,y2),(x3,y3),
    (x4,y4)] = lista de puntos[:4] del lienzo lista de

    puntos[:4] .setLineWidth(inch*0.1) canvas.setStrokeColor(green)
    canvas.line(x1,y1,x2,y2)
    canvas.setStrokeColor(red)
    canvas.line(x3,y3,x4,y4) # finalmente dibuja
    el lienzo curvo. setStrokeColor(negro)
    lienzo.bezier(x1,y1, x2,y2, x3,y3, x4,y4)
```

La figura creada por testbezier2 describe una curva compleja y suave porque las líneas tangentes adyacentes se "alinean" como se ilustra a continuación.

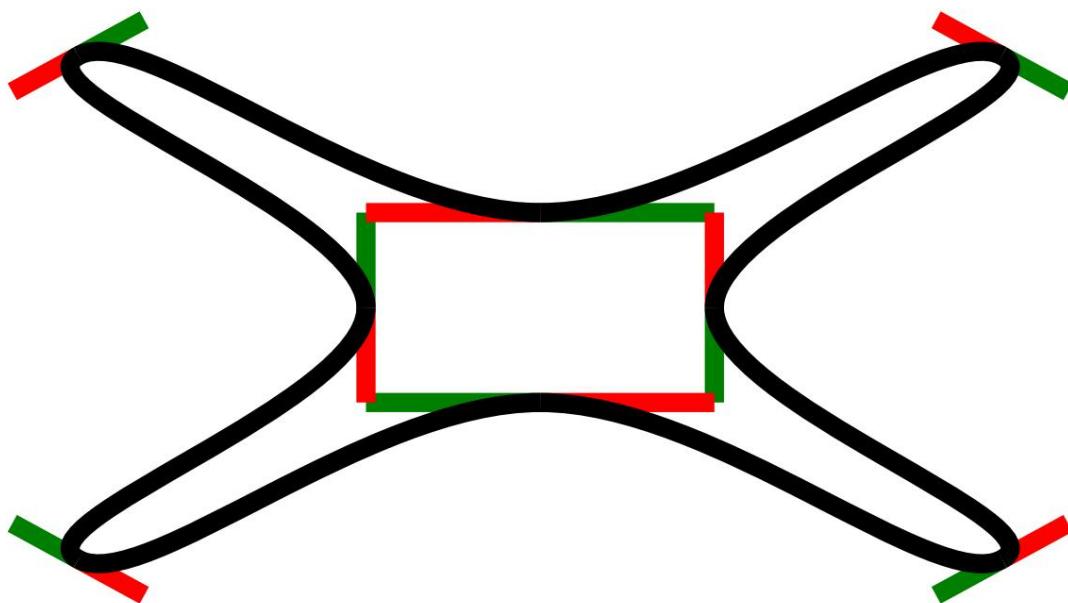


Figura 2-28: curvas de Bézier

2.16 Métodos de objeto de ruta

Los objetos de ruta construyen figuras gráficas complejas colocando el "lápiz" o el "pincel" en un punto inicial del lienzo y dibujando líneas o curvas en puntos adicionales del lienzo. La mayoría de las operaciones aplican pintura sobre el lienzo comenzando en el punto final de la última operación y dejan el pincel en un nuevo punto final.

```
objetoruta.moveTo(x,y)
```

El método `moveTo` levanta el pincel (finalizando cualquier secuencia actual de líneas o curvas, si las hay) y vuelve a colocar el pincel en la nueva ubicación (x,y) en el lienzo para iniciar una nueva secuencia de trazado.

```
rutaobjeto.lineTo(x,y)
```

El método `lineTo` pinta un segmento de línea recta desde la ubicación actual del pincel hasta la nueva ubicación (x,y).

```
objetoruta.curvaA(x1, y1, x2, y2, x3, y3)
```

El método `curveTo` comienza a pintar una curva de Bézier comenzando en la ubicación actual del pincel, usando (x1,y1), (x2,y2) y (x3,y3) como los otros tres puntos de control, dejando el pincel en (x3,y3). .

```
pathobject.arc(x1,y1, x2,y2, startAng=0, extensión=90)
```

```
pathobject.arcTo(x1, y1, x2, y2, startAng = 0, extensión = 90).
```

Los métodos `arc` y `arcTo` pintan elipses parciales. El método del arco primero "levanta el pincel" e inicia una nueva secuencia de formas. El método `arcTo` une el inicio de la elipse parcial a la secuencia de formas actual por segmento de línea antes de dibujar la elipse parcial. Los puntos (x1,y1) y (x2,y2) definen puntos de esquina opuestos de un rectángulo que encierra la elipse. `startAng` es un ángulo (en grados) que especifica dónde comenzar la elipse parcial donde el ángulo 0 es el punto medio del borde derecho del rectángulo circundante (cuando (x1,y1) es la esquina inferior izquierda y (x2,y2) es la esquina superior derecha). La extensión es el ángulo en grados que se recorrerá en la elipse.

```
def arcos(lienzo): de
    reportlab.lib.units importa pulgadas canvas.setLineWidth(4)
    canvas.setStrokeColorRGB(0.8, 1,
    0.6) # dibuja rectángulos que encierran los arcos
    canvas.rect(pulgadas, pulgadas, 1.5*pulgadas, pulgadas)
    canvas.rect(3*pulgadas, pulgadas, pulgadas, 1.5*pulgadas)
    canvas.setStrokeColorRGB(0, 0.2, 0, 4)
```

```
canvas.setFillColorRGB(1, 0.6, 0.8) p =
canvas.beginPath()
p.moveTo(0.2*pulgada, 0.2*pulgada)
p.arcTo(pulgada, pulgada, 2.5*pulgada, 2*pulgada, startAng=-30, extensión=135) p.arc(3*pulgada,
pulgada, 4*pulgada, 2.5*pulgada, startAng=-45, extensión=270) canvas.drawPath(p, relleno=1,
trazo=1)
```

La función de arcos anterior ejercita los dos métodos de elipse parcial. Produce el siguiente dibujo.

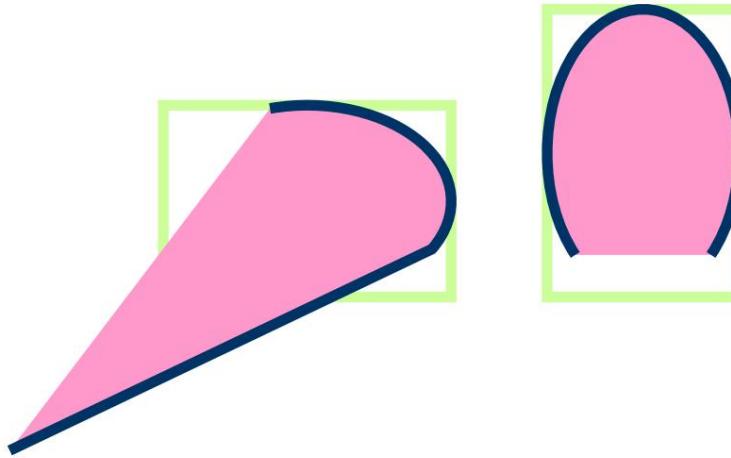


Figura 2-29: arcos en objetos de ruta

```
pathobject.rect(x, y, ancho, alto)
```

El método rect dibuja un rectángulo con la esquina inferior izquierda en (x,y) del ancho y alto especificados.

```
pathobject.ellipse(x, y, ancho, alto)
```

El método de la elipse dibuja una elipse encerrada en el rectángulo con la esquina inferior izquierda en (x,y) del ancho y alto especificados.

```
objetoruta.circle(x_cen, y_cen, r)
```

El método del círculo dibuja un círculo centrado en (x_cen, y_cen) con radio r.

```
def varias formas (lienzo): de
    reportlab.lib.units importar pulgadas pulgadas =
    int(pulgadas)
    canvas.setStrokeGray(0.5)
    canvas.grid(range(0,int(11*inch/2),int(inch/2)) , range(0,int(7*pulgada/2),int(pulgada/2))) canvas.setLineWidth(4)
    canvas.setStrokeColorRGB(0, 0.2,
    0.7) canvas.setFillColorRGB(1, 0.6, 0.8) p =
    canvas.beginPath() p.rect(0,0.5*pulgada, 0.5*pulgada,
    0.5*pulgada, 2*pulgada)
    p.circle(2.75*pulgada, 1.5*pulgada, 0.3*pulgada) p.elipse(3.5*pulgada,
    0.5 *pulgada, 1.2*pulgada, 2*pulgada) lienzo.drawPath(p,
    relleno=1, trazo=1)
```

La función de varias formas anterior muestra un rectángulo, un círculo y una elipse colocados en una cuadricula de marco de referencia.

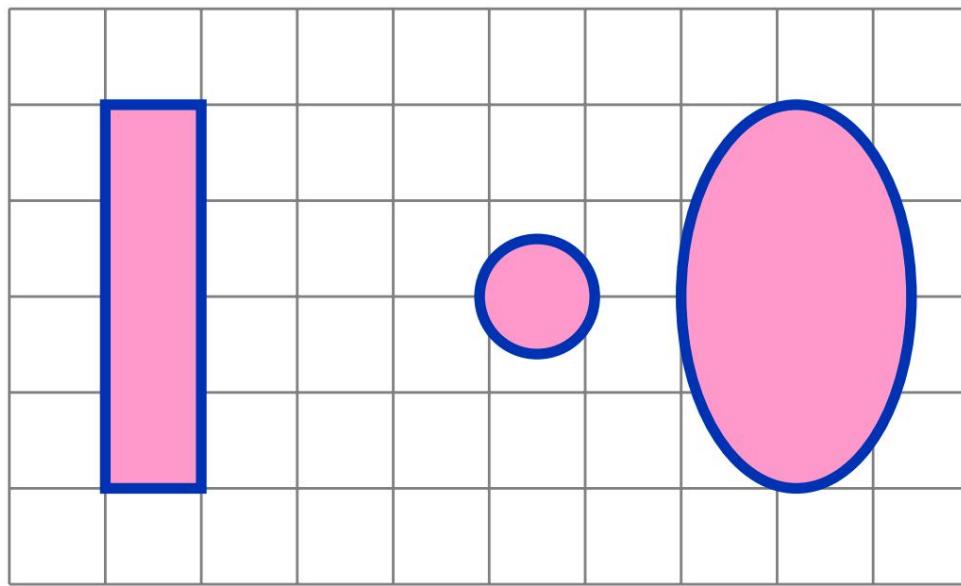


Figura 2-30: rectángulos, círculos y elipses en objetos de ruta

```
objetoruta.cerrar()
```

El método close cierra la figura gráfica actual pintando un segmento de línea desde el último punto de la figura hasta el punto inicial de la figura (el punto más reciente donde se colocó el pincel en el papel mediante moveTo o arc u otras operaciones de colocación) .

```
def figuras de cierre (lienzo): de
    reportlab.lib.units import pulgada h = pulgada/3.0; k =
    pulgada/2.0
    canvas.setStrokeColorRGB(0.2,0.3,0.5)
    canvas.setFillColorRGB(0.8,0.6,0.2)
    canvas.setLineWidth(4) p =
    canvas.beginPath() para i en
    (1,2,3,4) : para j en (1,2):
        xc,yc = pulgada*i,
        pulgada*j p.moveTo(xc,yc)
        p.arcTo(xc-h, yc-k,
        xc+h, yc+k, startAng =0, extensión=60*i) # cerrar solo el primero, no el segundo si j==1:
        p.close() canvas.drawPath(p, fill=1, Stroke=1)
```

La función de figuras de cierre ilustra el efecto de cerrar o no figuras que incluyen un segmento de línea y una elipse parcial.

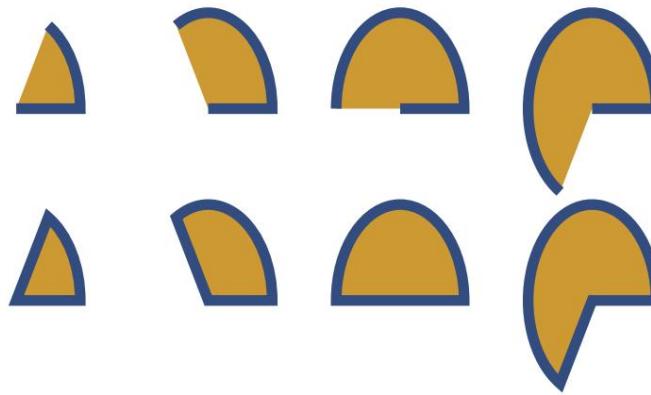


Figura 2-31: cerrar y no cerrar figuras de pathobject

Cerrar o no cerrar figuras gráficas afecta sólo el contorno trazo de una figura, no el relleno de la figura como se ilustra arriba.

Para ver un ejemplo más extenso de dibujo usando un objeto de ruta, examine la función de la mano.

```
def mano(lienzo, depuración=1, relleno=0): (startx, starty) = (0,0)
    curvas = [ ( 0, 2), ( 0, 4), ( 0, 8), # dorso de
    la mano (5, 8),
        (7,10), (7,14), (10,14), (10,13), (7.5, 8), # pulgar (13, 8), (14, 8), (17, 8), (19,
        8), (19, 6), (17, 6), (15, 6), (13, 6), (11, 6), #
        índice, señalando (12, 6 ), (13, 6), (14, 6), (16, 6), (16, 4), (14, 4),
        (13, 4), (12, 4), (11, 4), # medio (11.5, 4), (12, 4),
        (13, 4), (15, 4), (15, 2), (13, 2), (12.5, 2), (11.5,
        2), (11, 2), # anillo (11.5, 2), (12, 2), (12.5, 2), (14, 2), (14, 0), (12.5, 0), (10, 0), (8,
        0), (6, 0), # meñique, luego cerrar ]
```

```
desde reportlab.lib.units importe pulgadas si se depura:
canvas.setLineWidth(6) u = inch*0.2 p = canvas.beginPath()
p.moveTo(startx,
starty) ccopy = list(curves) while ccopy:
    [(x1,y1), (x2,y2), (x3,y3)] = ccopy[-3] del ccopia[-3]
    p.curveTo(x1*u,y1*u,x2*u,y2*u ,x3*u,y3*u) p.closePath() canvas.drawPath(p,
fill=fill) si depurar:
desde reportlab.lib.colors importar rojo, verde (lastx,
lasty) = (startx,
starty) ccopy = lista (curvas) mientras copia:
    [(x1,y1), (x2,y2), (x3,y3)] = ccopy[-3] del ccopia[-3] canvas.setStrokeColor(rojo)
    canvas.line(lastx*u,lasty*u,
x1 *u,y1*u) lienzo.setStrokeColor(verde)
    lienzo.line(x2*u,y2*u, x3*u,y3*u) (lastx,lasty) = (x3,y3)
```

En el modo de depuración (el predeterminado), la función manual muestra los segmentos de línea tangentes a las curvas Bézier utilizadas para componer la figura. Tenga en cuenta que donde los segmentos se alinean, las curvas se unen suavemente, pero donde no se alinean, las curvas muestran un "borde afilado".

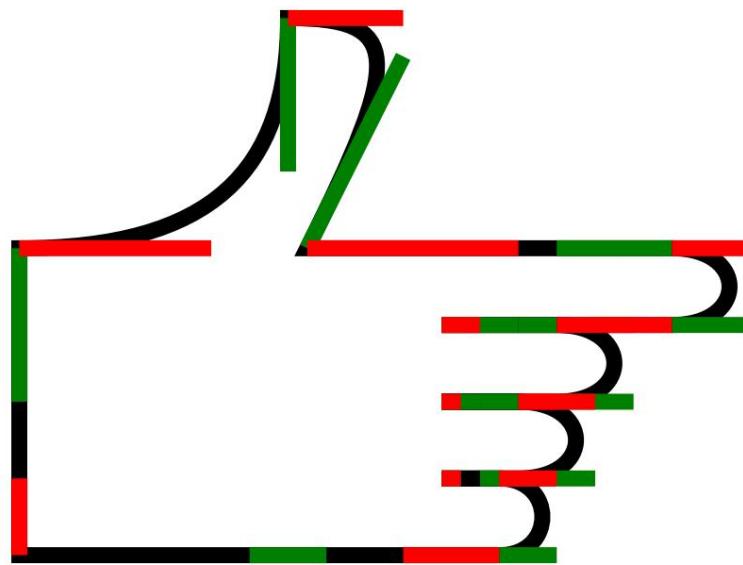


Figura 2-32: contorno de una mano usando curvas de Bézier

Utilizada en modo sin depuración, la función manual solo muestra las curvas de Bézier. Con el parámetro de relleno configurado, la figura se rellena usando el color de relleno actual.

```
def mano2(lienzo):
    lienzo.translate(20,10)
    lienzo.setLineWidth(3)
    lienzo.setFillColorRGB(0.1, 0.3, 0.9) lienzo.setStrokeGray(0.5)
    mano(lienzo, depuración=0, relleno=1)
```

Tenga en cuenta que el "trazo" del borde se extiende sobre el relleno interior donde se superponen.

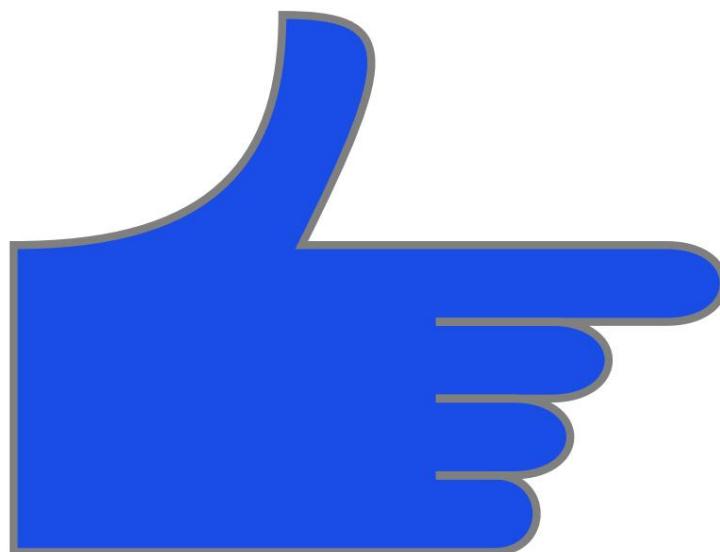


Figura 2-33: la mano terminada, llena

2.17 Lecturas adicionales: Biblioteca de gráficos ReportLab

Hasta ahora los gráficos que hemos visto estaban creados a un nivel bastante bajo. Sin embargo, cabe señalar que existe otra forma de crear gráficos mucho más sofisticados utilizando la biblioteca de gráficos ReportLab dedicada de alto nivel.

Se puede utilizar para producir gráficos reutilizables, independientes de la plataforma y de alta calidad para diferentes formatos de salida (vectoriales y de mapa de bits) como PDF, EPS, SVG, JPG y PNG.

Ahora se incluye una descripción más detallada de su filosofía y características en el Capítulo 11 de este documento, Gráficos, que contiene información sobre los componentes existentes y cómo crear componentes personalizados.

El Capítulo 11 también contiene detalles del paquete de gráficos ReportLab y sus componentes (etiquetas, ejes, leyendas y diferentes tipos de gráficos como gráficos de barras, líneas y circulares) que se basan directamente en la biblioteca de gráficos.

Capítulo 3 Fuentes y codificaciones

Este capítulo cubre fuentes, codificaciones y capacidades de idiomas asiáticos. Si lo único que le preocupa es generar archivos PDF para idiomas de Europa occidental, puede leer la sección "Unicode es el valor predeterminado" a continuación y omita el resto en una primera lectura. Esperamos que esta sección crezca considerablemente con el tiempo. Esperamos que se abra Source nos permitirá brindar un mejor soporte para más idiomas del mundo que otras herramientas, y agradecemos los comentarios y la ayuda en esta área.

3.1 Unicode y UTF8 son las codificaciones de entrada predeterminadas

A partir de la versión 2.0 de reportlab (mayo de 2006), toda la entrada de texto que proporcione a nuestras API debe estar en UTF8 o como objetos Python Unicode. Esto se aplica a los argumentos de canvas.drawString y las API relacionadas, el contenido de las celdas de la tabla, los parámetros del objeto de dibujo y el texto fuente del párrafo.

Consideramos hacer que la codificación de entrada fuera configurable o incluso dependiente de la configuración regional, pero decidimos que "explicito es mejor que implícito".

Esto simplifica muchas cosas que solíamos hacer anteriormente con respecto a las letras y símbolos griegos, etc. para mostrar cualquier carácter, averíguese su punto de código Unicode y asegúrese de que la fuente que está utilizando pueda mostrarlo.

Si está adaptando una aplicación ReportLab 1.x, o leyendo datos de otra fuente que contiene datos de un solo byte (por ejemplo, latin-1 o WinAnsi), necesita realizar una conversión a Unicode. El paquete de códecs de Python ahora incluye convertidores para todas las codificaciones comunes, incluidas las asiáticas.

Si sus datos no están codificados como UTF8, obtendrá un UnicodeDecodeError tan pronto como los introduzca en un formato que no sea ASCII. Por ejemplo, este fragmento a continuación intenta leer e imprimir una serie de nombres, incluido uno con acento francés: Marc-André Lemburg. El error estándar es bastante útil y te indica qué personaje no le gusta:

```
>>> desde reportlab.pdfgen.canvas importar lienzo
>>> c = Lienzo('temp.pdf')
>>> y = 700
>>> para la línea en el archivo('latin_python_gurus.txt','r'):
...     c.drawString(100, y, línea.strip())
...
Rastreo (llamadas recientes más última):
...
UnicodeDecodeError: el códec 'utf8' no puede decodificar bytes en la posición 9-11: datos no válidos
-->é L<--emburgo
>>>
```

La solución más simple es simplemente convertir sus datos a Unicode, indicando de qué codificación provienen, así:

```
>>> para la línea en el archivo('latin_input.txt','r'):
...     uniLine = unicode(línea, 'latin-1')
...     c.drawString(100, y, uniLine.strip())
>>>
>>> c.guardar()
```

3.2 Sustitución automática de fuentes de salida

Todavía hay varios lugares en el código, incluido el parámetro rl_config defaultEncoding y argumentos pasados a varios constructores de fuentes, que se refieren a codificaciones. Estos fueron útiles en el pasado cuando las personas necesitaban usar glifos en las fuentes Symbol y ZapfDingbats que son compatibles con la visualización de PDF dispositivos. Por defecto, las fuentes estándar (Helvetica, Courier, Times Roman) ofrecerán los glifos disponibles en Latin-1. Sin embargo, si nuestro motor detecta un carácter que no está en la fuente, intentará cambiar a Symbol o Za-pfDingbats para mostrarlos. Por ejemplo, si incluye el carácter Unicode para un par de tijeras orientadas hacia la derecha, \u2702, en una llamada a drawString, debería verlos (hay un ejemplo en test_pdfgen_general.py/pdf). No es necesario cambiar las fuentes en su código.

3.3 Uso de fuentes Tipo 1 no estándar

Como se analizó en el capítulo anterior, cada copia de Acrobat Reader viene con 14 fuentes estándar integradas.

Por lo tanto, la biblioteca PDF de ReportLab solo necesita hacer referencia a estos por su nombre. Si desea utilizar otras fuentes, deben estar disponibles para su código y se incrustarán en el documento PDF.

Puede utilizar el mecanismo que se describe a continuación para incluir fuentes arbitrarias en sus documentos. Tenemos una fuente de código abierto llamada DarkGardenMK que podemos usar con fines de prueba y/o documentación (y que usted también puede usar). Viene incluido con la distribución ReportLab en el directorio reportlab/fonts.

En este momento, la incrustación de fuentes se basa en archivos de descripción de fuentes en formato Adobe AFM ('Adobe Font Metrics') y PFB ('Printer Font Binary'). El primero es un archivo ASCII y contiene información sobre los caracteres ("glifos") de la fuente, como altura, ancho, información del cuadro delimitador y otras "métricas", mientras que el segundo es un archivo binario que describe las formas de la fuente. El directorio reportlab/fonts contiene los archivos 'DarkGardenMK.afm' y 'DarkGardenMK.pfb' que se utilizan como fuente de ejemplo.

En el siguiente ejemplo, ubique la carpeta que contiene la fuente de prueba y regístrela para uso futuro con el módulo pdfmetrics, después de lo cual podremos usarla como cualquier otra fuente estándar.

```
importar os
importar carpeta reportlab
= os.path.dirname(reportlab.__file__) + os.sep + 'fonts'
afmFile = os.path.join(carpeta, 'DarkGardenMK.afm')
pfbFile = os.path.join(carpeta, 'DarkGardenMK.pfb')

de reportlab.pdfbase importar pdfmetrics justFace =
pdfmetrics.EmbeddedType1Face(afmFile, pfbFile) faceName = 'DarkGardenMK' # extraido
del archivo AFM pdfmetrics.registerTypeFace(justFace) justFont =
pdfmetrics.Font('DarkGardenMK', faceName,
'WinAnsiEncoding') pdfmetrics.registrarFuente(soloFuente)

canvas.setFont('DarkGardenMK', 32)
canvas.drawString(10, 150, 'Esto debería estar en') canvas.drawString(10,
100, 'DarkGardenMK')
```

Tenga en cuenta que el argumento "WinAnsiEncoding" no tiene nada que ver con la entrada; es decir qué conjunto de caracteres dentro del archivo de fuente estarán activos y disponibles.

Esto debería estar en DarkGardenMK

Figura 3-1: Uso de una fuente no estándar

El nombre de la fuente proviene del campo FontName del archivo AFM. En el ejemplo anterior, conocíamos el nombre de antemano, pero muy a menudo los nombres de los archivos de descripción de fuentes son bastante crípticos y es posible que desee recuperar el nombre de un archivo AFM automáticamente. Cuando falte un método más sofisticado puedes usar algún código tan simple como este:

```
clase FontNameNotFoundError (excepción):
    aprobar

def buscarNombreFuente(ruta):
    "Extrae el nombre de una fuente de un archivo AFM".

    f = abierto (ruta)

    encontrado = 0
    mientras no se encuentra:
        línea = f.readline()[:-1] si no se encuentra y
        línea[16] == 'StartCharMetrics': genera FontNameNotFoundError, ruta si línea[8] == 'FontName':
            fontName = línea[9:] encontrada = 1

    devolver nombre de fuente
```

En el ejemplo de DarkGardenMK especificamos explícitamente el lugar de los archivos de descripción de fuentes que se cargarán. En general, preferirás almacenar tus fuentes en algunas ubicaciones canónicas y hacer que el mecanismo de incrustación las tenga en cuenta. Usando el mismo mecanismo de configuración que ya hemos visto al principio de esta sección podemos indicar una ruta de búsqueda predeterminada para fuentes Type-1.

Desafortunadamente, todavía no existe un estándar confiable para dichas ubicaciones (ni siquiera en la misma plataforma) y, por lo tanto, es posible que deba editar el archivo reportlab/rl_config.py para modificar el valor del identificador T1SearchPath para que contenga directorios adicionales. Nuestra propia recomendación es utilizar la carpeta reportlab/fonts en desarrollo; y tener las fuentes necesarias como partes empaquetadas de su aplicación en cualquier tipo de implementación de servidor controlada. Esto lo protege de fuentes instaladas y desinstaladas por otro software o administrador del sistema.

Advertencias sobre glifos faltantes

Si especifica una codificación, generalmente se supone que el diseñador de fuentes ha proporcionado todos los glifos necesarios. Sin embargo, esto no siempre es cierto. En el caso de nuestra fuente de ejemplo, las letras del alfabeto están presentes, pero faltan muchos símbolos y acentos. El comportamiento predeterminado es que la fuente imprima un carácter 'notdef' (normalmente una mancha, un punto o un espacio) cuando se le pasa un carácter que no puede dibujar. Sin embargo, puede pedirle a la biblioteca que le avise; El siguiente código (ejecutado antes de cargar una fuente) generará advertencias para cualquier glifo que no esté en la fuente cuando la registre.

```
importar reportlab.rl_config
reportlab.rl_config.warnOnMissingFontGlyphs = 0
```

3.4 Codificaciones de fuentes estándar de un solo byte

Esta sección le muestra los glifos disponibles en las codificaciones comunes.

El siguiente cuadro de códigos muestra los caracteres en WinAnsiEncoding. Ésta es la codificación estándar en Windows y en muchos sistemas Unix en Estados Unidos y Europa occidental. También se conoce como página de códigos 1252 y es prácticamente idéntica a ISO-Latin-1 (contiene uno o dos caracteres extra). Esta es la codificación predeterminada utilizada por la biblioteca PDF de Reportlab. Se generó a partir de una rutina estándar en reportlab/lib/codecharts.py, que se puede utilizar para mostrar el contenido de las fuentes. Los números de índice a lo largo de los bordes están en hexadecimal.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	A	B	C	D	E	F
00																																
20	!	ps			()	:	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	?					
40	@	ABCDEF	GHIJKL	MNO	PQR	S	TU	VW	XYZ	[]	abc	defghijklmn	opqrstuvwxyz	{	}	~									\	^					
60																																
80	€	•	f	...++	+f	n	y	!S	^%	s	c	OE	•	Z	•	~	TN	š	œ	•	ž	'	"	*	•	--						
A0	©	a	μ	¶	·	1/4	¼	¾	½	»	«	–	®					±	2	3	10											
C0	À	Á	Ã	Ä	Å	Æ	È	É	Í	Ó	Ñ	Ò	Ó	Ô	×	Ø	Ù	Ú	Ü	Ý	Þ											
E0	à	á	ã	ä	å	æ	è	é	í	ó	ñ	ò	ó	ô	×	ø	ù	ú	ü	ý	þ											

Figura 3-2: Codificación WinAnsi

El siguiente cuadro de códigos muestra los caracteres de MacRomanEncoding. como suena, este es el estándar codificación en computadoras Macintosh en América y Europa occidental. Como es habitual con codificaciones no Unicode, Los primeros 128 puntos de código (las 4 filas superiores en este caso) son el estándar ASCII y concuerdan con el código WinAnsi. cuadro de arriba; pero las 4 filas inferiores difieren.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	A	B	C	D	E	F
00																																
20	!	ps			()	:	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	?					
40	@	ABCDEF	GHIJKL	MNO	PQR	S	TU	VW	XYZ	[]	abc	defghijklmn	opqrstuvwxyz	{	}	~								\	^						
60																																
80	À	Á	Ã	Ä	Å	Æ	È	É	Í	Ó	Ñ	Ò	Ó	Ô	×	Ø	Ù	Ú	Ü	Ý	Þ											
A0	ç	é	ñ	ú	á	â	ã	ä	å	æ	è	é	í	ó	ô	ø	û	ú	ü	ý	þ											
C0	ç	é	ñ	ú	á	â	ã	ä	å	æ	è	é	í	ó	ô	ø	û	ú	ü	ý	þ											
E0	,	"																														

Figura 3-3: Codificación MacRoman

Estas dos codificaciones están disponibles para las fuentes estándar (Helvetica, Times-Roman y Courier y sus variantes) y estarán disponibles para la mayoría de las fuentes comerciales, incluidas las de Adobe. Sin embargo, algunas fuentes contienen glifos que no son texto y el concepto no se aplica realmente. Por ejemplo, ZapfDingbats y Symbol pueden cada uno ser tratados como si tuvieran su propia codificación.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	A	B	C	D	E	F
00																																
20																																
40																																
60																																
80																																
A0																																
C0																																
E0																																

Figura 3-4: ZapfDingbats y su única codificación

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	A	B	C	D	E	F
00																																
20	!	#	%	y	()	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	?						
40	A	B	X	D	E	F	C	H	I	K	L	M	N	O	P	T	R	S	Y	s	o	x	ψ	Z	[]						
60	a	b	d	e	f	g	i	c	f	l	m	n	o	p	r	t	s	p	x	ψ	z	{	}									
80	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n				
A0	€																															
C0	◊																															
E0	Σ																															

Figura 3-5: Símbolo y su única codificación

3.5 Compatibilidad con fuentes TrueType

Marius Gedminas (mgedmin@delfi.lt) con la ayuda de Viktorija Zaksiene (vika@pov.lt) han contribuido al soporte para fuentes TrueType integradas. Las fuentes TrueType funcionan en Unicode/UTF8 y no están limitadas a 256 caracteres.

Usamos reportlab.pdfbase.ttfonts.TTFont para crear un objeto de fuente de tipo verdadero y registrarlo usando reportlab.pdfbase.pdfmetrics.registerFont.

En pdffgen dibujando directamente en el lienzo podemos hacer

```
# sabemos que faltan algunos glifos, suprimir las advertencias import
reportlab.rl_config
reportlab.rl_config.warnOnMissingFontGlyphs = 0

desde reportlab.pdfbase importar pdfmetrics desde
reportlab.pdfbase.ttfonts importar TTFont
pdfmetrics.registerFont(TTFont('Vera', 'Vera.ttf'))
pdfmetrics.registerFont(TTFont('VeraBd', 'VeraBd.ttf')) pdfmetrics.
RegisterFont(TTFont('Veralt', 'Veralt.ttf')) pdfmetrics.registerFont(TTFont('VeraBI',
'VeraBI.ttf')) canvas.setFont('Vera', 32) canvas.drawString(10, 150, "Algo de texto
codificado en UTF-8")
canvas.drawString(10, 100, "¡En la fuente Vera TT!")
```

Parte del texto codificado en UTF-8 en la fuente Vera TT!

Figura 3-6: Uso de la fuente Vera TrueType

En el ejemplo anterior, el objeto de fuente de tipo verdadero se crea usando

```
TTFont(nombre,nombre de archivo)
```

de modo que el nombre interno de ReportLab viene dado por el primer argumento y el segundo argumento es una cadena (u objeto similar a un archivo) que indica el archivo TTF de la fuente. En el parche original de Marius, se suponía que el nombre del archivo era exactamente correcto, pero hemos modificado cosas para que si el nombre del archivo es relativo, entonces se realice una búsqueda del archivo correspondiente en el directorio actual y luego en los directorios especificados por reportlab.rl_config.TTFSearchpath !

Antes de usar las fuentes TT en Platypus, debemos agregar una asignación del nombre de la familia a los nombres de fuentes individuales que describen el comportamiento bajo los atributos y <i>.

```
de reportlab.pdfbase.pdfmetrics importar RegisterFontFamily
RegisterFontFamily('Vera',normal='Vera',bold='VeraBd',italic='Veralt',boldItalic='VeraBI')
```

Si solo tenemos una fuente Vera normal, sin negrita ni cursiva, entonces debemos asignar todas al mismo nombre de fuente interno. Las etiquetas y <i> ahora se pueden utilizar de forma segura, pero no tienen ningún efecto. Despues de registrar y mapear la fuente Vera como

arriba podemos usar texto de párrafo como

```
<font name="Times-Roman" size="14">Esto  
está en Times-Roman</font> <font  
name="Vera" color="magenta" size="14">y  
esto está en magenta <b><Verde</b></font>
```

¡Esto está en Times-Roman
y esto está en magenta Vera!

Figura 3-7: Uso de fuentes TTF en párrafos

3.6 Compatibilidad con fuentes asiáticas

La biblioteca PDF de Reportlab tiene como objetivo ofrecer soporte total para fuentes asiáticas. PDF es la primera solución realmente portátil para el manejo de textos asiáticos. Hay dos enfoques principales para esto: los paquetes de idiomas asiáticos de Adobe o las fuentes TrueType.

Paquetes de idiomas asiáticos

Este enfoque ofrece el mejor rendimiento ya que no es necesario incrustar nada en el archivo PDF; Al igual que con las fuentes estándar, todo depende del lector.

Adobe pone a disposición complementos para cada idioma principal. En Adobe Reader 6.0 y 7.0, se le pedirá que los descargue e instale tan pronto como intente abrir un documento usándolos. En versiones anteriores, veías un mensaje de error al abrir un documento asiático y tenías que saber qué hacer.

Japonés, chino tradicional (Taiwán/Hong Kong), chino simplificado (China continental) y coreano son compatibles y nuestro software conoce las siguientes fuentes: chs = chino simplificado

- (continental): 'STSong-Light' cht = chino tradicional (Taiwán):
- 'MSung-Light', 'MHei-Medium' kor = coreano: 'HYSMYeongJoStd-Medium', 'HYGothic-Medium'
- jpn = japonés: 'HeiseiMin-W3', 'HeiseiKakuGo-W5'
-

Dado que muchos usuarios no tendrán instalados los paquetes de fuentes, hemos incluido un mapa de bits bastante granulado de algunos caracteres japoneses. Discutiremos a continuación lo que se necesita para generarlos.

Aquí debería haber aparecido una imagen.

Antes de la versión 2.0, había que especificar una de las muchas codificaciones nativas al registrar una fuente CID. En la versión 2.0 debería crear una nueva clase UnicodeCIDFont.

```
desde reportlab.pdfbase importar pdfmetrics desde  
reportlab.pdfbase.cidfonts importar UnicodeCIDFont  
pdfmetrics.registerFont(UnicodeCIDFont('HeiseiMin-W3')) canvas.setFont('HeiseiMin-W3', 16)  
  
# los dos caracteres Unicode siguientes son "Tokyo" msg =  
u'\u6771\u4EAC: fuente Unicode, entrada Unicode' canvas.drawString(100, 675,  
msg)
```

El antiguo estilo de codificación con codificaciones explícitas aún debería funcionar, pero ahora sólo es relevante si necesita construir texto vertical. Nuestro objetivo es agregar opciones más legibles para texto horizontal y vertical al constructor UnicodeCID-Font en el futuro. Los siguientes cuatro scripts de prueba generan ejemplos en los idiomas correspondientes:

```
pruebas/test_multibyte_jpn.py pruebas/  
test_multibyte_kor.py pruebas/  
test_multibyte_chs.py pruebas/  
test_multibyte_cht.py
```

En versiones anteriores de la biblioteca PDF de ReportLab, teníamos que utilizar los archivos CMap de Adobe (ubicados cerca de Acrobat Reader si estaban instalados los paquetes de idiomas asiáticos). Ahora que solo tenemos una codificación con la que lidiar, los datos de ancho de caracteres están incrustados en el paquete y los archivos CMap no son necesarios para la generación. La ruta de búsqueda CMap en rl_config.py ahora está obsoleta y no tiene ningún efecto si se restringe a Unico-deCIDFont.

Fuentes TrueType con caracteres asiáticos

Esta es la manera fácil de hacerlo. No se necesita ningún manejo especial para trabajar con fuentes asiáticas TrueType. Los usuarios de Windows que hayan instalado, por ejemplo, japonés como opción en el Panel de control, tendrán una fuente "msmincho.ttf" que podrán utilizar. Sin embargo, tenga en cuenta que lleva tiempo analizar las fuentes y que es posible que sea necesario incrustar subconjuntos bastante grandes en sus archivos PDF. Ahora también podemos analizar archivos que terminan en .ttc, que son una ligera variación de .ttf.

Hacer

Esperamos desarrollar esta área del paquete durante algún tiempo. A continuación se presenta un resumen de las principales prioridades. ¡Agradecemos la ayuda!

- Garantizar que contamos con métricas de caracteres precisas para todas las codificaciones en escritura horizontal y vertical.
- Agregue opciones a UnicodeCIDFont para permitir variantes verticales y proporcionales donde la fuente lo permita.
- Mejorar el código de ajuste de palabras en párrafos y permitir la escritura vertical.

3.7 Pruebas de RenderPM

Este también puede ser el mejor lugar para mencionar la función de prueba de reportlab/graphics/renderPM.py, que puede considerarse el lugar canónico para las pruebas que ejercitan renderPM (el "Pixmap Renderer", a diferencia de renderPDF, renderPS o renderSVG).

Si ejecuta esto desde la línea de comando, debería ver muchos resultados como el siguiente.

```
C:\code\reportlab\graphics>renderPM.py escribió
pmout\renderPM0.gif escribió
pmout\renderPM0.tif escribió
pmout\renderPM0.png escribió
pmout\renderPM0.jpg escribió
pmout\renderPM0.pct
...
escribió pmout\renderPM12.gif
escribió pmout\renderPM12.tif
escribió pmout\renderPM12.png
escribió pmout\renderPM12.jpg
escribió pmout\renderPM12.pct
escribió pmout\index.html
```

Esto ejecuta una serie de pruebas que progresan desde una prueba de "Hola mundo" hasta varias pruebas de Líneas; cadenas de texto en varios tamaños, fuentes, colores y alineaciones; las formas básicas; grupos traducidos y rotados; coordenadas escaladas; cuerdas rotadas; grupos anidados; anclaje y fuentes no estándar.

Crea un subdirectorio llamado pmout, escribe los archivos de imagen en él y escribe una página index.html que facilita la consulta de todos los resultados.

Las pruebas relacionadas con fuentes que quizás desee ver son la prueba n.º 11 ('Cadenas de texto en una fuente no estándar') y la prueba n.º 12 ('Prueba de varias fuentes').

Capítulo 4 Exponer las capacidades especiales de PDF

PDF proporciona una serie de funciones para hacer que la visualización de documentos electrónicos sea más eficiente y cómoda, y nuestra biblioteca expone varios de ellos.

4.1 Formularios

La función Formulario le permite crear un bloque de gráficos y texto una vez cerca del inicio de un archivo PDF y luego simplemente consúltelo en las páginas siguientes. Si se trata de una serie de 5000 formularios comerciales repetitivos (por ejemplo, facturas o recibos de nómina de una página), sólo necesita almacenar el fondo una vez y simplemente dibujar el texto cambiante en cada página. Si se usan correctamente, los formularios pueden reducir drásticamente el tamaño del archivo y el tiempo de producción y, aparentemente, incluso acelerar el trabajo en la impresora.

No es necesario que los formularios hagan referencia a una página completa; Cualquier cosa que pueda repetirse con frecuencia debe colocarse en un forma.

El siguiente ejemplo muestra la secuencia básica utilizada. Un programa real probablemente definiría los formularios desde el principio. y consultarlos desde otra ubicación.

```
formas def (lienzo):
    #primero crea un formulario...
    lienzo.beginForm("SpumoniForm")
    #reutilizar algunas funciones de dibujo de antes
    spumoni (lienzo)
    lienzo.endForm()

    #entonces dibújalo
    lienzo.doForm("SpumoniForm")
```

4.2 Enlaces y Destinos

PDF admite hipervínculos internos. Existe una gama muy amplia de tipos de enlaces, tipos de destinos y eventos que se puede activar con un clic. Por el momento sólo admitimos la capacidad básica de saltar de una parte de un documento a otra y de controlar el nivel de zoom de la ventana después del salto. El método bookmarkPage define un destino que es el punto final de un salto.

```
lienzo.bookmarkPage(nombre,
    encajar="Ajustar",
    izquierda = Ninguno,
    arriba = Ninguno,
    abajo = Ninguno,
    derecha = Ninguno,
    zoom=Ninguno
)
```

De forma predeterminada, el método bookmarkPage define la propia página como destino. Después de saltar a un punto final definido por bookmarkPage, el navegador de PDF mostrará la página completa y la escalará para que se ajuste a la pantalla:

```
lienzo.bookmarkPage(nombre)
```

Se puede indicar al método bookmarkPage que muestre la página de varias maneras diferentes proporcionando un parámetro de ajuste.

adapta	Parámetros requeridos Significado	
Adaptar		Toda la página cabe en la ventana (valor predeterminado)
Top FitH		Coord superior en la parte superior de la ventana, ancho escalado para ajustarse
FitV izquierda		Coord izquierda a la izquierda de la ventana, altura escalada para ajustarse
FitR izquierda abajo derecha arriba		Escalar la ventana para ajustarla al rectángulo especificado
Zoom superior izquierdo XYZ		Control de grano fino. Si omites un parámetro el navegador de PDF lo interpreta como "dejarlo como está"

Tabla 4-1: Atributos requeridos para diferentes tipos de ajuste

Nota: la configuración de ajuste distingue entre mayúsculas y minúsculas, por lo que fit="FIT" no es válido

A veces desea que el destino de un salto sea parte de una página. El ajuste FitR le permite identificar un rectángulo en particular, escalando el área para que se ajuste a toda la página.

Para configurar la visualización en unas coordenadas xey particulares de la página y controlar el zoom directamente, utilice fit="XYZ".

```
lienzo.bookmarkPage('mi_marcador',fit="XYZ",left=0,top=200)
```

Este destino está en el extremo izquierdo de la página con la parte superior de la pantalla en la posición 200. Debido a que no se configuró el zoom, el zoom permanece en lo que el usuario haya configurado.

```
canvas.bookmarkPage('mi_marcador',fit="XYZ",left=0,top=200,zoom=2)
```

Esta vez el zoom está configurado para expandir la página 2 veces su tamaño normal.

Nota: Tanto el tipo de ajuste XYZ como el FitR requieren que sus parámetros posicionales (arriba, abajo, izquierda, derecha) se especifiquen en términos del espacio de usuario predeterminado. Ignoran cualquier transformación geométrica vigente en el estado gráfico del lienzo.



Nota: Se admiten dos métodos de marcadores anteriores, pero están en desuso ahora que bookmarkPage es tan general. Estos son bookmarkHorizontalAbsolute y bookmarkHorizontal.

Definición de enlaces internos

```
canvas.linkAbsolute(contenido, nombre de destino, Rect=Ninguno, addtopage=1, nombre=Ninguno, espesor=0, color=Ninguno, dashArray=Ninguno, **kw)
```

El método linkAbsolute define un punto de partida para un salto. Cuando el usuario explora el documento generado utilizando un visor dinámico (como Acrobat Reader), cuando se hace clic con el mouse y el puntero está dentro del rectángulo especificado por Rect, el visor saltará al punto final asociado con el nombre de destino. Como en el caso de bookmarkHorizontalAbsolute, el rectángulo Rect debe especificarse en términos del espacio de usuario predeterminado. El parámetro de contenidos especifica un fragmento de texto que se muestra en el visor si el usuario hace clic izquierdo en la región.

El rectángulo Rect debe especificarse en términos de una tupla (x1,y1,x2,y2) que identifica los puntos inferior izquierdo y superior derecho del rectángulo en el espacio de usuario predeterminado.

Por ejemplo el código

```
canvas.bookmarkPage("Significado_de_vida")
```

define una ubicación como la totalidad de la página actual con el identificador Significado_de_vida. Para crear un enlace rectangular mientras dibujamos una página posiblemente diferente, usariamos este código:

```
canvas.linkAbsolute("Encontrar el significado de la vida", "Significado_de_la_vida", (pulgadas, pulgadas, 6*pulgadas, 2*pulgadas))
```

De forma predeterminada, durante la visualización interactiva aparece un rectángulo alrededor del enlace. Utilice el argumento de palabra clave Border='[0 0 0]' para suprimir el rectángulo visible alrededor del enlace durante la visualización. Por ejemplo

```
canvas.linkAbsolute("Significado de la vida", "Significado_de_la_vida", (pulgada, pulgada, 6*pulgada, 2*pulgada), Border='[0 0 0]')
```

Los argumentos de espesor, color y dashArray se pueden usar alternativamente para especificar un borde si no se especifica ningún argumento Bor-de. Si se especifica Borde, debe ser una representación de cadena de una matriz PDF o una matriz PDF (consulte el módulo pdfdoc). El argumento de color (que debería ser una instancia de Color) es equivalente a un argumento de palabra clave C que debería resolverse en una definición de color de PDF (normalmente una matriz de PDF de tres entradas).

El método canvas.linkRect tiene una intención similar al método linkAbsolute, pero tiene un argumento adicional relativo=1, por lo que está destinado a obedecer la transformación del espacio de usuario local.

4.3 Árboles de contorno

Acrobat Reader tiene una página de navegación que puede contener un esquema de documento; normalmente debería estar visible al abrir esta guía. Proporcionamos algunos métodos simples para agregar entradas de esquema. Normalmente, un programa para crear un documento (como esta guía del usuario) llamará al método canvas.addOutlineEntry(self, title, key, nivel=0, cerrado=Ninguno) cuando llegue a cada encabezado del documento.

El título es el título que se mostrará en el panel izquierdo. La clave debe ser una cadena que sea única dentro del documento y que nombre un marcador, como ocurre con los hipervínculos. El nivel es cero, el nivel más alto, a menos que se especifique lo contrario, y es un error bajar más de un nivel a la vez (por ejemplo, seguir un encabezado de nivel 0 por un encabezado de nivel 2). Finalmente, el argumento cerrado especifica si el nodo en el panel de esquema está cerrado o abierto de forma predeterminada.

El siguiente fragmento está tomado de la plantilla de documento que da formato a esta guía del usuario. Un procesador central examina cada párrafo por turno y crea una nueva entrada de esquema cuando aparece un nuevo capítulo, tomando el texto del título del capítulo como texto del título. La clave se obtiene del número de capítulo (no se muestra aquí), por lo que el Capítulo 2 tiene la clave 'ch2'. El marcador al que apunta el esquema de entrada apunta a toda la página, pero también podría haber sido un párrafo individual.

```
#código resumido de nuestra plantilla de documento if parrafo.style
== 'Heading1': self.chapter = parrafo.getText()
key = 'ch%d' % self.chapterNo self.canv.bookmarkPage(key)
self.canv.addOutlineEntry (parrafo.getText(),
                          clave, 0, 0)
```

4.4 Efectos de transición de página

```
canvas.setPageTransition(self, nombre del efecto=Ninguno, duración=1,
                        dirección=0,dimensión='H',movimiento='!')
```

El método setPageTransition especifica cómo se reemplazará una página por la siguiente. Al configurar el efecto de transición de página en "disolver", por ejemplo, la página actual parecerá desaparecer cuando sea reemplazada por la página siguiente durante la visualización interactiva. Estos efectos son útiles para darle vida a las presentaciones de diapositivas, entre otros lugares. Consulte el manual de referencia para obtener más detalles sobre cómo utilizar este método.

4.5 Anotaciones de archivos internos

```
lienzo.setAuthor(nombre)
lienzo.setTitle(título)
lienzo.setSubject(asunto)
```

Estos métodos no tienen ningún efecto visible automáticamente en el documento. Añaden anotaciones internas al documento. Estas anotaciones se pueden ver usando el elemento del menú "Información del documento" del navegador y también se pueden usar como una forma estándar simple de proporcionar información básica sobre el documento al software de archivo que no necesita analizar el archivo completo. Para encontrar las anotaciones, vea el archivo de salida *.pdf usando un editor de texto estándar (como el bloc de notas en MS/Windows o vi o emacs en Unix) y busque la cadena /Autor en el contenido del archivo.

anotaciones def (lienzo):

```
de reportlab.lib.units importe pulgadas
canvas.drawString(pulgadas, 2,5*pulgadas,
                  "setAuthor, setTitle, setSubject no tienen ningún efecto visible") canvas.drawString(pulgadas,
pulgadas, "Pero si está viendo este documento dinámicamente") lienzo.drawString(pulgada, 0,5*pulgada, "mire la información
del archivo/documento") canvas.setAuthor("el equipo de ReportLab") canvas.setTitle(" Guía del usuario de
generación de PDF de ReportLab") canvas.setSubject("Cómo
generar PDF archivos usando los módulos ReportLab")
```

Si desea que el tema, el título y el autor se muestren automáticamente en el documento cuando se vea e imprima debes pintarlos en el documento como cualquier otro texto.

setAuthor, setTitle, setSubject no tienen ningún efecto visible

Pero si está viendo este documento dinámicamente

por favor mire la información del archivo/documento

Figura 4-1: Configuración de anotaciones internas del documento

4.6 Cifrado

Acerca del cifrado de archivos PDF

El estándar PDF de Adobe le permite hacer tres cosas relacionadas con un archivo PDF cuando lo cifra:

- Aplicarle protección con contraseña, por lo que el usuario debe proporcionar una contraseña válida antes de poder acceder a él,
- Cifrar el contenido del archivo para hacerlo inútil hasta que se descifre, y
- Controle si el usuario puede imprimir, copiar y pegar o modificar el documento mientras lo visualiza.

El controlador de seguridad de PDF permite especificar dos contraseñas diferentes para un documento:

- La contraseña de 'propietario' (también conocida como 'contraseña de seguridad' o 'contraseña maestra')
- La contraseña de 'usuario' (también conocida como 'contraseña de apertura')

Cuando un usuario proporciona cualquiera de estas contraseñas, el archivo PDF se abrirá, descifrará y mostrará en pantalla.

Si se proporciona la contraseña del propietario, el archivo se abre con control total; puede hacer cualquier cosa con él, incluso cambiar la configuración de seguridad y las contraseñas, o volver a cifrarlo con una nueva contraseña.

Si la contraseña de usuario fue la que se proporcionó, la abre en un modo más restringido. las restricciones se implementaron cuando se cifró el archivo y permitirán o negarán el permiso del usuario para hacer lo siguiente:

- Modificar el contenido del documento
- Copiar texto y gráficos del documento.
- Agregar o modificar anotaciones de texto y campos de formulario interactivos
- Imprimir el documento

Tenga en cuenta que todos los archivos PDF protegidos con contraseña están cifrados, pero no todos los archivos PDF cifrados están protegidos con contraseña. Si la contraseña de usuario de un documento es una cadena vacía, no se solicitará la contraseña cuando se cargue el archivo. Si solo protege un documento con la contraseña del propietario, tampoco se le solicitará la contraseña cuando abra el archivo. Si las contraseñas de propietario y usuario se establecen en la misma cadena al cifrar el Archivo PDF, el documento siempre se abrirá con los privilegios de acceso del usuario. Esto significa que es posible crear un archivo que, por ejemplo, nadie puede imprimir, ni siquiera la persona que lo creó.

¿Contraseña de propietario establecida?	¿Contraseña de usuario establecida?	Resultado
Y	-	No se requiere contraseña al abrir el archivo. Las restricciones se aplican a todos.
-	Y	Se requiere contraseña de usuario al abrir el archivo. Las restricciones se aplican a todos.
Y	Y	Se requiere una contraseña al abrir el archivo. Las restricciones se aplican solo si se proporciona la contraseña del usuario.

Cuando se cifra un archivo PDF, el cifrado se aplica a todas las cadenas y secuencias del archivo. Esto evita que las personas que no tienen la contraseña simplemente la eliminén del archivo PDF para obtener acceso a él; hace que el archivo sea inútil a menos que realmente tenga la contraseña.

Los métodos de cifrado estándar de PDF utilizan el algoritmo de resumen de mensajes MD5 (como se describe en RFC 1321, Algoritmo de resumen de mensajes MD5) y un algoritmo de cifrado conocido como RC4. RC4 es un cifrado de flujo simétrico: se utiliza el mismo algoritmo tanto para el cifrado como para el descifrado, y el algoritmo no cambia la longitud de los datos.

Cómo utilizar el cifrado

Los documentos se pueden cifrar pasando un argumento al objeto lienzo.

Si el argumento es un objeto de cadena, se utiliza como contraseña de usuario para el PDF.

El argumento también puede ser una instancia de la clase reportlab.lib.pdfencrypt.StandardEncryption, que permite un control más detallado sobre la configuración de cifrado.

El constructor StandardEncryption toma los siguientes argumentos:

```
def __init__(self, contraseña de usuario, contraseña de propietario = Ninguna,
            canPrint = 1,
            canModify = 1,
            canCopy = 1,
            canAnnotate = 1,
            fuerza = 40):
```

Los parámetros userPassword y ownerPassword establecen la contraseña relevante en el PDF cifrado.

Los indicadores booleanos canPrint, canModify, canCopy, canAnnotate determinan si un usuario puede realizar las acciones correspondientes en el PDF cuando solo se ha proporcionado una contraseña de usuario.

Si el usuario proporciona la contraseña del propietario al abrir el PDF, se pueden realizar todas las acciones independientemente de las banderas.

Ejemplo

Para crear un documento llamado hello.pdf con una contraseña de usuario 'rptlab' en el que no se permite la impresión, utilice el siguiente código:

```
desde reportlab.pdfgen importar lienzo desde reportlab.lib
importar pdfencrypt

enc=pdfencrypt.StandardEncryption("rptlab",canPrint=0)

def hola(c):
    c.drawString(100,100,"Hola mundo") c =
    canvas.Canvas("hola.pdf",encrypt=enc) hola(c) c.showPage()
    c.save()
```

4.7 Formularios interactivos

Descripción general de los formularios interactivos

El estándar PDF permite varios tipos de elementos interactivos, el kit de herramientas ReportLab actualmente admite sólo una fracción de las posibilidades y debe considerarse un trabajo en progreso. Actualmente permitimos opciones con widgets de casilla de verificación, radio, elección y cuadro de lista ; Los valores de texto se pueden ingresar con un widget de campo de texto . Todos Los widgets se crean llamando a métodos en la propiedad canvas.acroform .

Ejemplo

Esto muestra el mecanismo básico de creación de un elemento interactivo en la página actual.

```
lienzo.acroform.casilla de verificación (
    nombre = 'CBO',
    información sobre herramientas = 'Campo CBO',
    marcado = Verdadero,
    x=72,y=72+4*36,
    buttonStyle='diamante',
    borderStyle='biselado',
    ancho de borde = 2,
    color del borde = rojo,
    fillColor=verde,
    color de texto = azul,
    forzarBorde=Verdadero)
```

NB, tenga en cuenta que la propiedad del lienzo de acroform se crea automáticamente a pedido y que solo hay un formulario permitido en un documento.

Uso de casillas de verificación

El método canvas.acroform.checkbox crea un widget de casilla de verificación en la página actual. El valor de la La casilla de verificación está en Sí o en APAGADO. Los argumentos son

Parámetros de canvas.acroform.checkbox		
Parámetro	Significado	Por defecto
nombre	el nombre del parámetro	Ninguno
X	la posición horizontal en la página (coordenadas absolutas)	0
y	la posición vertical en la página (coordenadas absolutas)	0
tamaño	Las dimensiones del contorno tamaño x tamaño	20
comprobado	si es Verdadero, la casilla de verificación está inicialmente marcada	FALSO
botónEstilo	el estilo de casilla de verificación (ver más abajo)	'controlar'
forma	El esquema del widget (ver más abajo)	'cuadrado'
color de relleno	color que se utilizará para llenar el widget	Ninguno
color de texto	el color del símbolo o texto	Ninguno
ancho del borde	como dice	1
color del borde	el color del borde del widget	Ninguno
estilo de borde	El nombre del estilo de borde.	' sólido '
información sobre herramientas	El texto que se mostrará al pasar el cursor sobre el widget.	Ninguno
anotaciónBanderas	cadena separada en blanco de indicadores de anotación	'imprimir'
campoBanderas	Banderas de campo separadas en blanco (ver más abajo)	'requerido'
fuerzafrontera	cuando es verdadero, una frontera obliga a dibujar una frontera	FALSO
relativo	si es verdadero, obedezca la transformación del lienzo actual	FALSO
guionLen	la línea discontinua que se utilizará si borderStyle=='discontinuo'	3

Uso de radio

El método `canvas.acroform.radio` crea un widget de radio en la página actual. El valor de la radio es el valor del valor seleccionado del grupo de radio o APAGADO si no hay ninguno seleccionado. Los argumentos son

Parámetros de <code>canvas.acroform.radio</code>		
Parámetro	Significado	Por defecto
nombre	el nombre del grupo de la radio (es decir, el parámetro)	Ninguno
valor	el nombre del grupo de radio	Ninguno
X	la posición horizontal en la página (coordenadas absolutas)	0
y	la posición vertical en la página (coordenadas absolutas)	0
tamaño	Las dimensiones del contorno tamaño x tamaño	20
seleccionado	si es Verdadero esta radio es la seleccionada en su grupo	FALSO
botónEstilo	el estilo de casilla de verificación (ver más abajo)	'controlar'
forma	El esquema del widget (ver más abajo)	'cuadrado'
color de relleno	color que se utilizará para llenar el widget	Ninguno
color de texto	el color del símbolo o texto	Ninguno
ancho del borde	como dice	1
color del borde	el color del borde del widget	Ninguno
estilo de borde	El nombre del estilo de borde.	' sólido '
información sobre banderas	El texto que se mostrará al pasar el cursor sobre el widget.	Ninguno
anotaciónBanderas	cadena separada en blanco de indicadores de anotación	'imprimir'
campoBanderas	Banderas de campo separadas en blanco (ver más abajo)	'no se requiere ToggleToOff radio'
fuerzafrontera	cuando es verdadero, una frontera obliga a dibujar una frontera	FALSO
relativo	si es verdadero, obedezca la transformación del lienzo actual	FALSO
guionLen	la línea discontinua que se utilizará si <code>borderStyle=="discontinuo"</code>	3

Uso del cuadro de lista

El método `canvas.acroform.listbox` crea un widget de cuadro de lista en la página actual. El cuadro de lista contiene una lista de opciones, una o más de las cuales (dependiendo de las banderas de campo) pueden seleccionarse.

Parámetros de <code>canvas.acroform.listbox</code>		
Parámetro	Significado	Por defecto
nombre	el nombre del grupo de la radio (es decir, el parámetro)	Ninguno
opciones	Lista o tupla de opciones disponibles	[]
valor	Singleton o lista de cadenas de opciones seleccionadas	[]
X	la posición horizontal en la página (coordenadas absolutas)	0
y	la posición vertical en la página (coordenadas absolutas)	0
ancho	El ancho del widget	120
altura	La altura del widget	36
nombre de la fuente	El nombre de la fuente tipo 1 que se utilizará.	'Helvética'
tamaño de fuente	El tamaño de fuente a utilizar.	12
color de relleno	color que se utilizará para llenar el widget	Ninguno
color de texto	el color del símbolo o texto	Ninguno
ancho del borde	como dice	1
color del borde	el color del borde del widget	Ninguno
estilo de borde	El nombre del estilo de borde.	' sólido '
información sobre banderas	El texto que se mostrará al pasar el cursor sobre el widget.	Ninguno
anotaciónBanderas	cadena separada en blanco de indicadores de anotación	'imprimir'
campoBanderas	Banderas de campo separadas en blanco (ver más abajo)	" "

Parámetros de canvas.acroform.listbox		
Parámetro	Significado	Por defecto
fuerzafrontera	cuando es verdadero, una frontera obliga a dibujar una frontera	FALSO
relativo	si es verdadero, obedezca la transformación del lienzo actual	FALSO
guionLen	la línea discontinua que se utilizará si borderStyle=='discontinuo'	3

Uso de elección

El método canvas.acroform.choice crea un widget desplegable en la página actual. El menú desplegable contiene una lista de opciones, una o más de las cuales (dependiendo de las banderas de campo) se pueden seleccionar. Si agrega edición al campo Banderas , el resultado se puede editar.

Parámetros de canvas.acroform.choice		
Parámetro	Significado	Por defecto
nombre	el nombre del grupo de la radio (es decir, el parámetro)	Ninguno
opciones	Lista o tupla de opciones disponibles	[]
valor	Singleton o lista de cadenas de opciones seleccionadas	[]
X	la posición horizontal en la página (coordenadas absolutas)	0
y	la posición vertical en la página (coordenadas absolutas)	0
ancho	El ancho del widget	120
altura	La altura del widget	36
nombre de la fuente	El nombre de la fuente tipo 1 que se utilizará.	'Helvetica'
tamaño de fuente	El tamaño de fuente a utilizar.	12
color de relleno	color que se utilizará para llenar el widget	Ninguno
color de texto	el color del símbolo o texto	Ninguno
ancho del borde	como dice	1
color del borde	el color del borde del widget	Ninguno
estilo de borde	El nombre del estilo de borde.	' sólido '
información sobre herramientas	El texto que se mostrará al pasar el cursor sobre el widget.	Ninguno
anotaciónBanderas	cadena separada en blanco de indicadores de anotación	'imprimir'
campoBanderas	Banderas de campo separadas en blanco (ver más abajo)	'combinación'
fuerzafrontera	cuando es verdadero, una frontera obliga a dibujar una frontera	FALSO
relativo	si es verdadero, obedezca la transformación del lienzo actual	FALSO
guionLen	la línea discontinua que se utilizará si borderStyle=='discontinuo'	3
maxlen	Ninguno o longitud máxima del valor del widget	Ninguno

Uso del campo de texto

El método canvas.acroform.textfield crea un widget de entrada de campo de texto en la página actual. El campo de texto puede ser editado para cambiar el valor del widget

Parámetros de canvas.acroform.textfield		
Parámetro	Significado	Por defecto
nombre	el nombre del grupo de la radio (es decir, el parámetro)	Ninguno
valor	Valor del campo de texto	"
maxlen	Ninguno o longitud máxima del valor del widget	100
X	la posición horizontal en la página (coordenadas absolutas)	0
y	la posición vertical en la página (coordenadas absolutas)	0
ancho	El ancho del widget	120
altura	La altura del widget	36
nombre de la fuente	El nombre de la fuente tipo 1 que se utilizará.	'Helvetica'

Parámetros de canvas.acroform.textfield		
Parámetro	Significado	Por defecto
tamaño de fuente	El tamaño de fuente a utilizar.	12
color de relleno	color que se utilizará para llenar el widget	Ninguno
color de texto	el color del símbolo o texto	Ninguno
ancho del borde	como dice	1
color del borde	el color del borde del widget	Ninguno
estilo de borde	El nombre del estilo de borde.	' sólido '
información sobre banderas	El texto que se mostrará al pasar el cursor sobre el widget.	Ninguno
anotaciónBanderas	cadena separada en blanco de indicadores de anotación	' imprimir '
campoBanderas	Banderas de campo separadas en blanco (ver más abajo)	"
fuerzafrontera	cuando es verdadero, una frontera obliga a dibujar una frontera	FALSO
relativo	si es verdadero, obedezca la transformación del lienzo actual	FALSO
guiónLen	la línea discontinua que se utilizará si borderStyle='discontinuo'	3

Estilos de botones

El argumento de estilo del botón indica qué estilo de símbolo debe aparecer en el botón cuando se selecciona.

Hay varias opciones

- controlar
- cruz
- círculo
- estrella
- diamante

tenga en cuenta que el procesador de documentos puede hacer que algunos de estos símbolos sean incorrectos para su aplicación prevista.

El lector Ac-robot prefiere usar su propia representación además de lo que la especificación dice que debería mostrarse (especialmente cuando se utilizan las funciones de resaltado de formularios)

Forma del widget

El argumento de forma describe cómo debe aparecer el contorno de la casilla de verificación o del widget de radio que puede usar

- círculo
- cuadrado

El renderizador puede tomar sus propias decisiones sobre cómo debería verse el widget; por eso Acrobat Reader prefiere contornos circulares para las radios.

Estilo de borde

El argumento borderStyle cambia la apariencia 3D del widget en la página.

- sólido
- discontinuo
- recuadro
- biselado
- subrayada

argumento fieldFlags

Los argumentos de fieldFlags pueden ser un número entero o una cadena que contenga tokens separados en blanco, los valores se muestran en la siguiente tabla. Para más información consulte la especificación en PDF.

Fichas y valores de bandera de campo		
Simbólico	Significado	Valor
sololectura	El widget es de solo lectura.	1<<0

Fichas y valores de bandera de campo		
Simbólico	Significado	Valor
requerido	el widget es requerido	1<<1
noExportar	no exportar el valor del widget	1<<2
noToggleToOff	radios solo uno debe estar encendido	1<<14
radio	agregado por el método de radio	1<<15
presionar el botón	si el botón es un pulsador	1<<16
radiosAl Unison	Las radios con el mismo valor se alternan entre sí.	1<<25
multilínea	para widget de texto multilínea	1<<12
contraseña	campo de texto de contraseña	1<<13
seleccionar archivo	widget de selección de archivos	1<<20
No revisar la ortografía	como dice	1<<22
no desplazarse	los campos de texto no se desplazan	1<<23
peine	hacer un texto de estilo peine basado en el valor maxlen	1<<24
Texto rico	si se utiliza texto enriquecido	1<<25
combinación	para campos de elección	1<<17
editar	si la elección es editable	1<<18
clasificar	si los valores deben ordenarse	1<<19
selección múltiple	si la elección permite la selección múltiple	1<<21
confirmarOnSelChange	no utilizado por reportlab	1<<26

anotaciónFlags Argumento

Los widgets de PDF son anotaciones y tienen propiedades de anotación que se muestran en la siguiente tabla

Valores y tokens de bandera de anotación		
Simbólico	Significado	Valor
invisible	El widget no se muestra.	1<<0
oculto	El widget está oculto.	1<<1
imprimir	El widget se imprimirá	1<<2
no hacer zoom	La anotación no se escalará con la página renderizada.	1<<3
anotar	El widget no rotará con la página.	1<<4
no ver	No renderices el widget	1<<5
sololectura	No se puede interactuar con el widget	1<<6
bloqueado	El widget no se puede cambiar.	1<<7
alternarista	El widget puede verse después de algunos eventos.	1<<8
contenido bloqueado	El contenido del widget es fijo.	1<<9

Capítulo 5 PLATYPUS - Diseño de página y tipografía

Usando secuencias de comandos

5.1 Objetivos de diseño

Platypus significa "Diseño de página y tipografía mediante scripts". Es una biblioteca de diseño de páginas de alto nivel que le permite crear mediante programación documentos complejos con un mínimo de esfuerzo.

El diseño de Platypus busca separar lo más posible las decisiones de diseño de "alto nivel" del contenido del documento. Así, por ejemplo, los párrafos se construyen utilizando estilos de párrafo y las páginas se construyen utilizando plantillas de página con la intención de que cientos de documentos con miles de páginas puedan reformatearse según diferentes especificaciones de estilo con modificaciones de unas pocas líneas en un único archivo compartido. que contiene los estilos de párrafo y las especificaciones de diseño de página.

Se puede pensar que el diseño general de Platypus tiene varias capas, de arriba hacia abajo, estas son

DocTemplates el contenedor más externo del documento;

Especificaciones de PageTemplates para diseños de páginas de diversos tipos;

Enmarca especificaciones de regiones en páginas que pueden contener texto o gráficos fluidos.

Texto fluido o elementos gráficos que deben "fluir en el documento (es decir, elementos como imágenes, párrafos y tablas, pero no elementos como pies de página o gráficos de página fijos).

pdfgen.Canvas el nivel más bajo que finalmente recibe la pintura del documento de las otras capas.

Plantilla de documento

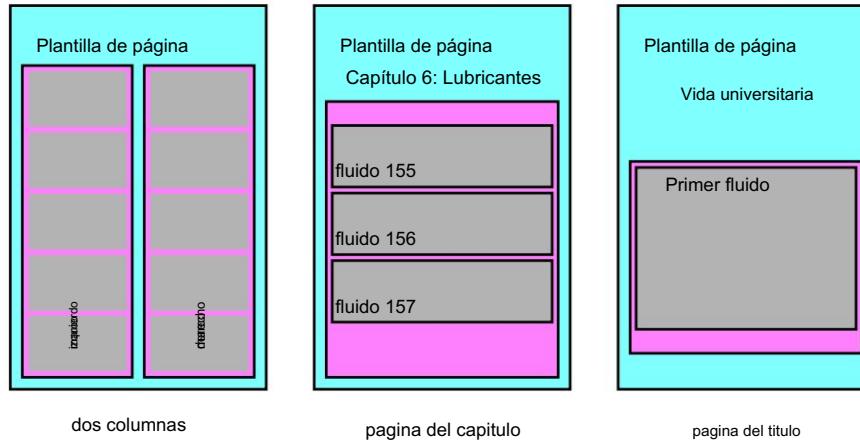


Figura 5-1: Ilustración de la estructura DocTemplate

La ilustración anterior ilustra gráficamente los conceptos de DocTemplates, PageTemplates y Flowables. Sin embargo, es engañoso porque cada una de las PageTemplates en realidad puede especificar el formato para cualquier número de páginas (no sólo una, como podría inferirse del diagrama).

DocTemplates contiene una o más PageTemplates, cada una de las cuales contiene uno o más marcos.

Los fluidos son cosas que pueden fluir hacia un marco, por ejemplo, un párrafo o una tabla.

Para usar ornitorrinco, crea un documento a partir de una clase DocTemplate y pasa una lista de Flowables a su método de compilación. El método de creación de documentos sabe cómo procesar la lista de flujos en algo razonable.

Internamente, la clase DocTemplate implementa el diseño y el formato de la página utilizando varios eventos. Cada uno de los eventos tiene un método de controlador correspondiente llamado handle_XXX donde XXX es el nombre del evento. Un evento típico es frameBegin que ocurre cuando la maquinaria comienza a utilizar un marco por primera vez.

Una historia de Platypus consta de una secuencia de elementos básicos llamados Flowables y estos elementos impulsan el motor de formateo de Platypus basado en datos. Para modificar el comportamiento del motor, un tipo especial de flujo, ActionFlowables, indique al motor de diseño que, por ejemplo, salte a la siguiente columna o cambie a otra PageTemplate.

5.2 Primeros pasos

Considera la siguiente secuencia de código que proporciona un ejemplo muy simple de "hola mundo" para Platypus.

```
desde reportlab.platypus importe SimpleDocTemplate, párrafo, espaciador desde reportlab.lib.styles importe
getSampleStyleSheet desde reportlab.rl_config importe defaultPageSize desde
reportlab.lib.units importe pulgada PAGE_HEIGHT=defaultPageSize[1];
PAGE_WIDTH=defaultPageSize[0] estilos =
getSampleStyleSheet()
```

Primero importamos algunos constructores, algunos estilos de párrafo y otras comodidades de otros módulos.

```
Título = "Hola mundo"
pageinfo = "ejemplo de ornitorrinco" def
myFirstPage(canvas, doc): canvas.saveState()
    canvas.setFont('Times-
Bold',16) canvas.drawCentredString(PAGE_WIDTH/
2.0, PAGE_HEIGHT-108, Título) canvas.setFont('Times-Roman',9) canvas.drawString(pulgadas, 0,75
* pulgadas, "Primera página / %s" % información
de página) canvas.restoreState()
```

Definimos las características fijas de la primera página del documento con la función anterior.

```
def myLaterPages(canvas, doc):
    canvas.saveState()
    canvas.setFont('Times-Roman',9)
    canvas.drawString(pulgadas, 0,75 * pulgadas, "Página %d %s" % (doc.page, pageinfo) ) lienzo.restoreState()
```

Como queremos que las páginas posteriores a la primera se vean diferentes a la primera, definimos un diseño alternativo para las características fijas de las otras páginas. Tenga en cuenta que las dos funciones anteriores utilizan las operaciones de lienzo a nivel de pdfgen para pintar las anotaciones de las páginas.

```
def ir(): doc =
    SimpleDocTemplate("resultado.pdf")
    Historia = [Espaciador(1,2*pulgada)]
    estilo = estilos["Normal"] para i en el
    rango(100): bogustext = ("Este
        es el párrafo número %. " % i) *20 p = Párrafo(bogustext , estilo)

    Historia.append(p)
    Story.append(Espaciador(1,0,2*pulgadas))
    doc.build(Historia, onFirstPage=myFirstPage, onLaterPages=myLaterPages)
```

Finalmente, creamos una historia y construimos el documento. Tenga en cuenta que aquí estamos utilizando una plantilla de documento "preparada" que viene prediseñada con plantillas de página. También utilizamos un estilo de párrafo prediseñado. Aquí solo utilizamos dos tipos de fluidos: espaciadores y párrafos. El primer espaciador garantiza que los párrafos salten la cadena del título.

Para ver el resultado de este programa de ejemplo, ejecute el módulo docs/userguide/examples.py (de la distribución de documentos de ReportLab) como un "script de nivel superior". La interpretación del script python ejemplos.py generará la salida de Platypus phello.pdf.

5.3 Fluidos

Los fluidos son cosas que se pueden dibujar y que tienen métodos de envoltura, dibujo y quizás división.

Flowable es una clase base abstracta para dibujar cosas y una instancia conoce su tamaño y dibuja en su propio sistema de coordenadas (esto requiere que la API base proporcione un sistema de coordenadas absoluto cuando se llama al método Flowable.draw). Para obtener una instancia, utilice f=Flowable().

Cabe señalar que la clase Flowable es una clase abstracta y normalmente solo se usa como clase base.

Para ilustrar la forma general en que se usan Flowables, mostramos cómo se usa y dibuja una clase derivada Paragraph en un lienzo. Los párrafos son tan importantes que ocuparán un capítulo entero para ellos solos.

```
de reportlab.lib.styles importar getSampleStyleSheet de reportlab.platypus importar
párrafo de reportlab.pdfgen.canvas importar Canvas styleSheet
= getSampleStyleSheet() style = styleSheet["BodyText"]

P=Paragraph("Este es un ejemplo muy tonto",estilo) canv = Canvas("doc.pdf") aW = 460 # ancho
y alto disponibles aH = 800 w,h = P.wrap(aW,
aH) if w <=aW y h<=aH:
    # encontrar el espacio requerido
    P.drawOn(canv,0,aH) aH = aH - h
    canv.save() # reducir la altura disponible
demás:
    generar ValueError, "No hay suficiente espacio"
```

Métodos de usuario fluidos

Fluido.draw()

Esto se llamará para pedirle al fluido que realmente se rinda. La clase Flowable no implementa el sorteo. El código de llamada debe garantizar que el flowable tenga un atributo canv, que es pdfgen.Canvas, que debe dibujarse para que el Canvas esté en un estado apropiado (en lo que respecta a las rotaciones de traducción, etc.). Normalmente este método sólo será llamado internamente por el método drawOn. Las clases derivadas deben implementar este método.

Fluido.drawOn(lienzo,x,y)

Este es el método que utilizan los programas de control para representar el flujo en un lienzo en particular. Maneja la traducción a las coordenadas del lienzo (x,y) y garantiza que el fluido tenga un atributo canv para que el método de dibujo (que no está implementado en la clase base) pueda representar en un marco de coordenadas absolutas.

Flowable.wrap (ancho disponible, altura disponible)

Esto será llamado por el marco adjunto antes de que se pregunte a los objetos su tamaño, dibujo o lo que sea. Devuelve el tamaño realmente utilizado.

Flowable.split (yo, ancho de disponibilidad, altura de disponibilidad):

Esto lo llamarán marcos más sofisticados cuando falle el ajuste. Los fluidos estúpidos deberían devolver [], lo que significa que no se pueden dividir. Los fluidos inteligentes deberían dividirse y devolver una lista de fluidos. Depende del código del cliente garantizar que se eviten intentos repetidos de división. Si el espacio es suficiente, el método de división debería devolver [self]. De lo contrario, el fluido debería reorganizarse y devolver una lista [f0,...] de fluidos que se considerarán en orden. El método de división implementado debe evitar cambiarse a sí mismo, ya que esto permitirá que mecanismos de diseño sofisticados realicen múltiples pasadas sobre una lista de elementos fluidos.

5.4 Directrices para un posicionamiento fluido

Dos métodos, que devuelven cero de forma predeterminada, proporcionan orientación sobre el espaciado vertical de los fluidos:

```
Flowable.getSpaceAfter(yo);
Flowable.getSpaceBefore(yo);
```

Estos métodos devuelven cuánto espacio debe seguir o preceder al fluido. El espacio no pertenece al fluido en sí, es decir, el método de dibujo del fluido no debería considerarlo al renderizar. Los programas de control utilizarán los valores devueltos para determinar cuánto espacio requiere un fluido particular en contexto.

Todos los fluidos tienen una propiedad hAlign: ('IZQUIERDA', 'DERECHA', 'CENTRO' o 'CENTRO'). Para los párrafos que ocupan todo el ancho del marco, esto no tiene ningún efecto. Para tablas, imágenes u otros objetos que miden menos que el ancho del marco, esto determina su ubicación horizontal.

Los capítulos que siguen cubrirán los tipos específicos más importantes de fluidos: párrafos y tablas.

5.5 cuadros

Los marcos son contenedores activos que a su vez están contenidos en PageTemplates. Los marcos tienen una ubicación y un tamaño y mantienen un concepto de espacio dibujable restante. El comando

```
Marco (x1, y1, ancho, alto, leftPadding = 6, bottomPadding = 6,
rightPadding=6, topPadding=6, id=Ninguno, showBoundary=0)
```

crea una instancia de Marco con la esquina inferior izquierda en las coordenadas (x1,y1) (en relación con el lienzo en el momento de su uso) y con dimensiones ancho x alto. Los argumentos de Padding son cantidades positivas que se utilizan para reducir el espacio disponible para dibujar. El argumento id es un identificador para usar en tiempo de ejecución, por ejemplo, 'LeftColumn' o 'RightColumn', etc. Si el argumento showBoundary no es cero, entonces el límite del marco se dibujará en tiempo de ejecución (esto es útil a veces).

Métodos de usuario del marco

```
Frame.addFromList (lista de dibujo, lienzo)
```

consume Flowables desde el frente de la lista de dibujo hasta que el marco esté lleno. Si no puede caber en un objeto, genera una excepción.

```
Frame.split(fluido,canv)
```

Solicita al fluido que se divida utilizando el espacio disponible y devuelve la lista de fluidos.

```
Marco.drawBoundary(lienzo)
```

Dibuja el límite del marco como un rectángulo (principalmente para depurar).

Usando marcos

Los marcos se pueden utilizar directamente con lienzos y fluidos para crear documentos. El método Frame.addFromList maneja las llamadas wrap & drawOn por usted. No necesitas toda la maquinaria de Platypus para obtener algo útil en PDF.

```
de reportlab.pdfgen.canvas importar lienzo de reportlab.lib.styles
importar getSampleStyleSheet de reportlab.lib.units importar pulgadas de
reportlab.platypus importar párrafo, marco estilos =
getSampleStyleSheet() estiloN = estilos['Normal'] estiloH = estilos['Título1']
historia = []
```

```
#agregar algunos elementos
fluidos story.append(Paragraph("Este es un encabezado",styleH))
story.append(Paragraph("Este es un párrafo en estilo <i>Normal</i>.",
estiloN)) c =
Lienzo('mydoc.pdf') f = Marco(pulgadas,
pulgadas, 6*pulgadas, 9*pulgadas, showBoundary=1) f.addFromList(story,c) c.save()
```

5.6 Documentos y Plantillas

La clase BaseDocTemplate implementa la maquinaria básica para formatear documentos. Una instancia de la clase contiene una lista de una o más PageTemplates que se pueden usar para describir el diseño de la información en una sola página. El método de compilación se puede utilizar para procesar una lista de Flowables para producir un documento PDF .

La clase BaseDocTemplate

```
BaseDocTemplate(self, nombre de archivo,
                tamaño de página=tamaño de página
                predeterminado, plantillas
                de página=[],
                showBoundary=0, margen
                izquierdo=pulgada, margen
                derecho=pulgada,
                margen superior=pulgada,
                margen inferior=pulgada,
                permitirSplitting=1,
                título=Ninguno,
                autor=Ninguno,
                _pageBreakQuick=1 , cifrar=Ninguno)
```

crea una plantilla de documento adecuada para crear un documento básico. Viene con bastante maquinaria interna, pero sin plantillas de página predeterminadas. El nombre del archivo requerido puede ser una cadena, el nombre de un archivo para recibir el documento PDF creado ; alternativamente, puede ser un objeto que tenga un método de escritura como BytesIO, un archivo o un socket.

Los argumentos permitidos deben explicarse por sí solos, pero showBoundary controla si se dibujan o no los límites del marco, lo que puede resultar útil para fines de depuración. El argumento enableSplitting determina si los métodos integrados deben intentar dividir Flowables individuales en marcos. El argumento _pageBreakQuick determina si un intento de realizar un salto de página debe intentar finalizar todos los marcos de la página o no, antes de finalizar la página. El argumento de cifrado determina si se cifra o no el documento y cómo. De forma predeterminada, el documento no está cifrado. Si cifrar es un objeto de cadena, se utiliza como contraseña de usuario para el pdf. Si cifrar es una instancia de reportlab.lib.pdfencrypt.StandardEncryption, este objeto se utiliza para cifrar el pdf. Esto permite un control más detallado sobre la configuración de cifrado.

Métodos de usuario BaseDocTemplate

Estos son de interés directo para los programadores de clientes porque normalmente se espera que se utilicen.

```
BaseDocTemplate.addPageTemplates(self,pageTemplates)
```

Este método se utiliza para agregar una o una lista de PageTemplates a documentos existentes.

```
BaseDocTemplate.build(self, flowables, nombre de archivo=Ninguno, canvasmaker=canvas.Canvas)
```

Este es el método principal que interesa al programador de aplicaciones. Suponiendo que la instancia del documento está configurada correctamente, el método de compilación toma la historia en la forma de la lista de flujos (el argumento de flujos) y recorre la lista forzando los flujos uno a la vez a través de la maquinaria de formato. Efectivamente, esto hace que la instancia BaseDocTemplate emita llamadas a los métodos handle_XXX de la instancia para procesar los distintos eventos.

Métodos de plantilla de BaseDoc virtual de usuario

Estos no tienen ninguna semántica en la clase base. Están concebidos como puros ganchos virtuales en la máquina de diseño. Los creadores de clases derivadas inmediatamente pueden anularlas sin preocuparse de afectar las propiedades del motor de diseño.

```
BaseDocTemplate.afterInit(yo)
```

Esto se llama después de la inicialización de la clase base; una clase derivada podría anular el método para agregar PageTemplates predeterminadas.

BaseDocTemplate.afterPage(yo)

Esto se llama después del procesamiento de la página e inmediatamente después del método afterDrawPage de la plantilla de página actual. Una clase derivada podría usar esto para hacer cosas que dependen de la información de la página, como la primera y la última palabra de la página de un diccionario.

BaseDocTemplate.beforeDocument(self)

Esto se llama antes de realizar cualquier procesamiento en el documento, pero después de que la maquinaria de procesamiento esté lista. Por lo tanto, se puede utilizar para hacer cosas en el pdfgen.canvas de la instancia y similares.

BaseDocTemplate.beforePage(self)

Esto se llama al comienzo del procesamiento de la página e inmediatamente antes del método beforeDrawPage de la plantilla de página actual. Podría usarse para restablecer los titulares de información específica de la página.

BaseDocTemplate.filterFlowables(self,fluidos)

Esto se llama para filtrar fluidos al inicio del método principal handle_flowable. Al regresar, si flowables[0] se ha establecido en Ninguno, se descarta y el método principal regresa inmediatamente.

BaseDocTemplate.afterFlowable(self, fluido)

Se llama después de que se haya renderizado un fluido. Una clase interesada podría utilizar este enlace para recopilar información sobre qué información está presente en una página o marco en particular.

Métodos del controlador de eventos BaseDocTemplate

Estos métodos constituyen la mayor parte del motor de diseño. Los programadores no deberían tener que llamar o anular estos métodos directamente a menos que estén intentando modificar el motor de diseño. Por supuesto, el programador experimentado que quiera intervenir en un evento particular, XXX, que no corresponde a uno de los métodos virtuales, siempre puede anular y llamar al método base desde la versión de clase dirigida. Hacemos esto fácil al proporcionar un sinónimo de clase base para cada uno de los métodos del controlador con el mismo nombre precedido por un guión bajo '_'.

```
def handle_pageBegin(self): hacerCosas()

BaseDocTemplate.handle_pageBegin(self) hacerMásCosas()

#usando el sinónimo def
handle_pageEnd(self): doStuff()

self._handle_pageEnd()
doMoreStuff()
```

Aquí enumeramos los métodos sólo como una indicación de los eventos que se están manejando. Los programadores interesados pueden echar un vistazo a la fuente.

```
handle_currentFrame(self,fx)
handle_documentBegin(self)
handle_flowable(self,flowables)
handle_frameBegin(self,*args)
handle_frameEnd(self)
handle_nextFrame(self,fx)
handle_nextPageTemplate(self,pt)
handle_pageBegin(self)
handle_pageBreak(self)
handle_pageEnd(self )
```

Usar plantillas de documentos puede ser muy sencillo; SimpleDocTemplate es una clase derivada de BaseDocTemplate que proporciona su propia configuración de PageTemplate y Frame.

```
de reportlab.lib.styles importar getSampleStyleSheet de reportlab.lib.pagesizes
importar carta de reportlab.platypus importar párrafo,
SimpleDocTemplate estilos = getSampleStyleSheet()
```

```
estiloN = estilos['Normal'] estiloH =  
estilos['Encabezado1'] historia = []  
  
#agregar algunos elementos  
fluidos story.append(Paragraph("Este es un encabezado",styleH))  
story.append(Paragraph("Este es un párrafo en estilo <i>Normal</i>.",  
estiloN))  
doc = SimpleDocTemplate('mydoc.pdf',pagesize = carta) doc.build(historia)
```

Plantillas de página

La clase PageTemplate es una clase contenedora con una semántica bastante mínima. Cada instancia contiene una lista de marcos y tiene métodos que deben llamarse al principio y al final de cada página.

Plantilla de página(id=Ninguno,marcos=[],onPage=_doNothing,onPageEnd=_doNothing)

se utiliza para inicializar una instancia, el argumento frames debe ser una lista de marcos, mientras que los argumentos opcionales onPage y onPageEnd son invocables que deben tener la firma def XXX(lienzo,documento)

donde lienzo y documento son el lienzo y el documento que se está dibujando.

Estas rutinas están diseñadas para pintar partes de páginas que no fluyen (es decir, estándar). Estas funciones de atributos son exactamente paralelas a los métodos virtuales puros PageTemplate.beforPage y PageTemplate.afterPage que tienen la firma beforPage(self,canvas,documento). Los métodos permiten utilizar la derivación de clases para definir el comportamiento estándar, mientras que los atributos permiten cambios de instancia. El argumento id se utiliza en tiempo de ejecución para realizar el cambio de PageTemplate, por lo que id='FirstPage' o id='TwoColumns' son típicos.

Capítulo 6 Párrafos

La clase reportlab.platypus.Paragraph es una de las más útiles de Platypus Flowables; puede formatear texto bastante arbitrario y permite cambios de color y estilo de fuente en línea mediante un marcado de estilo XML. La forma general del texto formateado puede estar justificada, irregular o centrada hacia la derecha o hacia la izquierda. El marcado XML se puede utilizar incluso para insertar caracteres griegos o para crear subíndices.

El siguiente texto crea una instancia de la clase Paragraph:

```
Párrafo(texto, estilo, viñetaTexto=Ninguno)
```

El argumento de texto contiene el texto del párrafo; El exceso de espacio en blanco se elimina del texto en los extremos e internamente después de los avances de línea. Esto permite un uso sencillo del texto con sangría y comillas triples en los scripts de Python . El argumento BulletText proporciona el texto de una viñeta predeterminada para el párrafo. La fuente y otras propiedades para el texto del párrafo y la viñeta se configuran mediante el argumento de estilo.

El argumento de estilo debe ser una instancia de la clase ParagraphStyle obtenida normalmente usando

```
desde reportlab.lib.styles importar ParagraphStyle
```

Esta clase contenedora proporciona la configuración de múltiples atributos de párrafo predeterminados de forma estructurada. Los estilos están organizados en un objeto de estilo de diccionario llamado hoja de estilo que permite acceder a los estilos como hoja de estilo ['BodyText']. Se proporciona una hoja de estilo de muestra.

```
de reportlab.lib.styles importar getSampleStyleSheet hoja de estilo=getSampleStyleSheet()
normalStyle = hoja de estilo['Normal']
```

Las opciones que se pueden configurar para un párrafo se pueden ver en los valores predeterminados de ParagraphStyle. Los valores con guión bajo inicial ('_') se derivan de los valores predeterminados en el módulo reportlab.rl_config, que se derivan del módulo reportlab.rl_settings.

clase Estilo de párrafo

```
clase Estilo de párrafo (Conjunto de propiedades):
valores predeterminados =
    'fontName':_baseFontName, 'fontSize':10,
    'leading':12,
    'leftIndent':0,
    'rightIndent':0,
    'firstLineIndent':0,
    'alignment':TA_LEFT,
    'spaceBefore':0, 'spaceAfter ':0,
    'bulletFontName':_baseFontName,
    'bulletFontSize':10,
    'bulletIndent':0, 'textColor':
negro, 'backColor':Ninguno,
    'wordWrap':Ninguno,
    'borderWidth': 0,
    'borderPadding': 0,
    'borderColor': Ninguno,
    'borderRadius': Ninguno,
    'allowWidows': 1, 'allowOrphans':
0, 'textTransform': Ninguno,
    'endDots': Ninguno,
    'splitLongWords':1, 'underlineWidth':
_baseUnderlineWidth,
    'bulletAnchor': 'start',
    'justifyLastLine': 0, 'justifyBreaks': 0, 'spaceShrinkage':
_spaceShrinkage, 'strikeWidth':
_baseStrikeWidth, #ancho de
    trazo 'underlineOffset':
    _baseUnderlineOffset, #gap para subrayado doble/
    triple 'underlineGap ': _baseUnderlineGap, 'strikeOffset': _baseStrikeOffset,
    #fracción del tamaño de fuente para compensar el tachado #fracción del tamaño de fuente para compensar los subrayados
```

```
'strikeGap': _baseStrikeGap, 'linkUnderline': #brecha para strike doble/triple
    _platypus_link_underline, #underlineColor: Ninguno, #strikeColor: Ninguno,
    'hyphenationLang': _hyphenationLang,
    'uriWasteReduce': _uriWasteReduce,
    'embeddedHyphenation': _embeddedHyphenation, }
```

6.1 Uso de estilos de párrafo

Las clases Paragraph y ParagraphStyle juntas manejan las necesidades de formato más comunes. Los siguientes ejemplos dibujan párrafos en varios estilos y agregan un cuadro delimitador para que pueda ver exactamente qué espacio se ocupa.

```
alineación = 0
permitirHuérfanos = 0
permitirViudas = 1 backColor
= Ninguno
borderColor = Ninguno borderPadding
= 0 borderRadius = Ninguno
borderWidth = 0 BulletAnchor = inicio

Por la presente se le acusa de que el 28 de mayo
de 1970 usted publicó intencionalmente, ilegalmente
y con malicia premeditada un supuesto libro
de frases inglés-húngaro con la intención de
causar una alteración del orden público. ¿Cómo
se declara?

BulletFontName = Helvetica BulletFontSize = 10

balanceIndent = 0
incrustadoHyphenation = 1 endDots = Ninguno

firstLineIndent = 0 fontName = Helvetica

Tamaño de fuente = 10
hyphenationLang = en_GB justifyBreaks = 0
justifyLastLine = 0 interlineado =
12 leftIndent = 0 linkUnderline = 0
rightIndent = 0
spaceAfter = 0 spaceBefore
= 0 spaceShrinkage = 0.05
splitLongWords = 1 strikeGap
= 1 strikeOffset = 0.25*F
strikeWidth =

textColor = Color(0,0,0,1) textTransform = Ninguno
underlineGap = 1 underlineOffset =
-0.125*F underlineWidth =

uriResiduosReduce = 0,3
ajuste de palabras = Ninguno
```

Figura 6-1: Estilo de párrafo predeterminado

Los dos atributos spaceBefore y spaceAfter hacen lo que dicen, excepto en la parte superior o inferior de un marco. En la parte superior de un marco, se ignora spaceBefore y en la parte inferior, se ignora spaceAfter. Esto significa que puede especificar que un estilo 'Encabezado2' tenía dos pulgadas de espacio antes cuando aparece en la mitad de la página, pero no obtendrá acres de espacios en blanco en la parte superior de la página. Estos dos atributos deben considerarse como "solicitudes" al Marco y no forman parte del espacio que ocupa el Párrafo en sí.

Las etiquetas fontSize y fontName son obvias, pero es importante establecer el interlineado. Este es el espacio entre líneas de texto adyacentes; una buena regla general es hacerlo un 20% más grande que el tamaño en puntos. Llegar

texto a doble espacio, utilice interlineado alto. Si configura autoLeading (predeterminado "desactivado") en "min" (use el interlineado observado incluso si es más pequeño que el especificado) o "max" (use el mayor entre observado y especificado), entonces se intenta determinar el interlineado línea por línea. Esto puede resultar útil si las líneas contienen diferentes tamaños de fuente, etc.

La siguiente figura muestra el espacio antes y después y un interlineado aumentado:

<pre> alineación = 0 permitirHuérfanos = 0 permitirViudas = 1 backColor = Ninguno borderColor = Ninguno borderPadding = 0 borderRadius = Ninguno borderWidth = 0 BulletAnchor = iniciar BulletFontName = Helvetica balaFontSize = 10 balaIndent = 0 incrustadoHyphenation = 1 endDots = Ninguno firstLineIndent = 0 </pre> <pre> fontName = Helvetica fontSize = 10 hyphenationLang = en_GB justifyBreaks = 0 justifyLastLine = 0 interlineado = 16 leftIndent = 0 linkUnderline = 0 </pre> <pre> rightIndent = 0 spaceAfter = 6 spaceBefore = 6 spaceShrinkage = 0,05 splitLongWords = 1 strikeGap = 1 strikeOffset = 0,25*F </pre> <pre> strikeWidth = textColor = Color(0,0,0,1) textTransform = Ninguno </pre> <pre> espacio de subrayado = 1 desplazamiento de subrayado = -0,125*F underlineWidth = uriWasteReduce = 0,3 wordWrap = Ninguno </pre>	<p>Por la presente se le acusa de que el 28 de mayo de 1970 usted publicó intencionalmente, ilegalmente y con malicia premeditada un supuesto libro de frases inglés-húngaro con la intención de causar una alteración del orden público. ¿Cómo se declara?</p>
---	---

Figura 6-2: Espacio antes y después y mayor avance

El atributo borderPadding ajusta el relleno entre el párrafo y el borde de su fondo.

Puede ser un valor único o una tupla que contenga de 2 a 4 valores. Estos valores se aplican de la misma manera que en las hojas de estilo en cascada (CSS). Si se da un solo valor, ese valor se aplica a los cuatro lados. Si se da más de un valor, se aplican en el sentido de las agujas del reloj a los lados comenzando desde arriba. Si se dan dos o tres valores, los valores faltantes se toman del lado opuesto.

Tenga en cuenta que en el siguiente ejemplo el cuadro amarillo está dibujado junto al párrafo mismo.

```
alineación = 0  
permitirOrphans = 0  
permitirWidows = 1 backColor  
= #FFFF00 borderColor = #000000  
borderPadding = (7, 2, 20) borderRadius  
= Ninguno borderWidth = 1
```

```
BulletAnchor = iniciar BulletFontName =  
Helvetica  
balaFontSize = 10 balaIndent = 0  
incrustadoHyphenation = 1  
endDots = Ninguno firstLineIndent = 0
```

```
fontName = Helvetica fontSize = 10  
hyphenationLang =  
en_GB justifyBreaks = 0 justifyLastLine = 0  
interlineado = 12 leftIndent = 0  
linkUnderline = 0
```

```
rightIndent = 0 spaceAfter = 0  
spaceBefore = 0  
spaceShrinkage = 0,05  
splitLongWords = 1 strikeGap = 1  
strikeOffset = 0,25*F
```

```
strikeWidth = textColor =  
Color(0,0,0,1) textTransform = Ninguno  
  
espacio de subrayado = 1  
desplazamiento de subrayado = -0,125*F  
underlineWidth = uriWasteReduce  
= 0,3 wordWrap = Ninguno
```

Por la presente se le acusa de que el 28 de mayo de 1970 usted publicó intencionalmente, ilegalmente y con malicia premeditada un supuesto libro de frases inglés-húngaro con la intención de causar una alteración del orden público. ¿Cómo se declara?

Figura 6-3: Relleno variable

Los atributos leftIndent y rightIndent hacen exactamente lo que cabría esperar; firstLineIndent se agrega a la sangría izquierda de la primera línea. Si desea un borde izquierdo recto, recuerde establecer firstLineIndent igual a 0.

```
alineación = 0  
permitirHuérfanos = 0  
permitirViudas = 1  
backColor = Ninguno borderColor  
= Ninguno  
borderPadding = 0 borderRadius  
= Ninguno borderWidth = 0  
  
BulletAnchor = iniciar BulletFontName =  
Helvetica  
balaFontSize = 10 balaIndent = 0  
incrustadoHyphenation = 1  
endDots = Ninguno firstLineIndent = 24
```

```
fontName = Helvetica fontSize = 10  
hyphenationLang =  
en_GB justifyBreaks = 0 justifyLastLine = 0  
interlineado = 12 leftIndent = 24  
linkUnderline = 0
```

```
rightIndent = 24 spaceAfter = 0  
spaceBefore = 0  
spaceShrinkage = 0,05  
splitLongWords = 1 strikeGap = 1  
strikeOffset = 0,25*F
```

```
strikeWidth = textColor =  
Color(0,0,0,1) textTransform = Ninguno  
  
espacio de subrayado = 1  
desplazamiento de subrayado = -0,125*F  
underlineWidth = uriWasteReduce  
= 0,3 wordWrap = Ninguno
```

Por la presente se le
acusá de que, el 28 de mayo de 1970,
usted, de forma deliberada, ilegal y
con malicia de previsión, publicó
un supuesto libro de frases inglés-
húngaro con la intención de provocar
una infracción del paz. ¿Cómo se
declara?

Figura 6-4: sangrías de un tercio de pulgada a izquierda y derecha, dos tercios en la primera línea

Configurar firstLineIndent igual a un número negativo, leftIndent mucho más alto y usar una fuente diferente (le mostraremos cómo más adelante!) puede brindarle una lista de definiciones:

```

alineación = 0
permitirHuérfanos = 0
permitirViudas = 1
backColor = Ninguno borderColor
= Ninguno
borderPadding = 0 borderRadius
= Ninguno borderWidth = 0

BulletAnchor = iniciar BulletFontName =
Helvetica
balaFontSize = 10 balaIndent = 0
incrustadoHyphenation = 1
endDots = Ninguno firstLineIndent = 0

fontName = Helvetica fontSize = 10
hyphenationLang =
en_GB justifyBreaks = 0 justifyLastLine = 0
interlineado = 12 leftIndent = 36
linkUnderline = 0

rightIndent = 0 spaceAfter = 0
spaceBefore = 0
spaceShrinkage = 0,05
splitLongWords = 1 strikeGap = 1
strikeOffset = 0,25*F

strikeWidth = textColor =
Color(0,0,0,1) textTransform = Ninguno

espacio de subrayado = 1
desplazamiento de subrayado = -0,125*F
underlineWidth = uriWasteReduce
= 0,3 wordWrap = Ninguno

```

Juez Pickles: Por la presente se le acusa de que el 28 de mayo de 1970 publicó de forma deliberada, ilegal y con malicia premeditada un supuesto libro de frases inglés-húngaro con la intención de alterar el orden público.

¿Cómo se declara?

Figura 6-5: Listas de definiciones

Hay cuatro valores posibles de alineación, definidos como constantes en el módulo reportlab.lib.enums.

Estos son TA_LEFT, TA_CENTER o TA_CENTRE, TA_RIGHT y TA_JUSTIFY, con valores de 0, 1, 2 y 4 respectivamente. Estos hacen exactamente lo que cabría esperar.

Establezca wordWrap en 'CJK' para obtener un ajuste de línea en idiomas asiáticos. Para texto occidental normal, puede cambiar la forma en que el algoritmo de salto de línea maneja las viudas y los huérfanos con los valores enableWidows y enableOrphans. Normalmente ambos deberían establecerse en 0, pero por razones históricas hemos permitido viudas. El color predeterminado del texto se puede configurar con textColor y el color de fondo del párrafo se puede configurar con backColor. Las propiedades del borde del párrafo se pueden cambiar usando borderWidth, borderPadding, borderColor y borderRadius.

El atributo textTransform puede ser Ninguno, "superior" o "inferior" para obtener el resultado obvio.

El atributo endDots puede ser Ninguno, una cadena o un objeto con atributos text y fontName, fontSize, textColor, backColor y dy(offset) opcionales para especificar el contenido final en la última línea de los párrafos justificados a izquierda/derecha.

El atributo splitLongWords se puede establecer en un valor falso para evitar dividir palabras muy largas.

El atributo BulletAnchor puede ser "inicio", "medio", "final" o "numérico" para controlar dónde está anclada la viñeta.

El atributo justifyBreaks controla si las líneas deliberadamente interrumpidas con una etiqueta
 deben justificarse

El atributo spaceShrinkage es un número fraccionario que especifica cuánto espacio hay en una línea de párrafo. puede encogerse para que encaje; normalmente es algo así como 0,05

Los atributos underlineWidth, underlineOffset, underlineGap y underlineColor controlan el comportamiento del subrayado cuando se utiliza <u> o una etiqueta de enlace. Esas etiquetas pueden tener valores de anulación de estos atributos. El valor del atributo para ancho y desplazamiento es una fracción * Letra donde la letra puede ser una de P, L, f o F que representan proporciones de tamaño de fuente. P usa el tamaño de fuente en la etiqueta, F es el tamaño de fuente máximo en la etiqueta, f es el tamaño de fuente inicial dentro de la etiqueta. L significa el tamaño de fuente global (estilo párrafo). ancho de huelga, Los atributos strikeOffset, strikeGap y strikeColor hacen lo mismo para las líneas tachadas.

El atributo linkUnderline controla si las etiquetas de enlace se subrayan automáticamente.

Si el módulo pyphen python está instalado, el atributo hyphenationLang controla qué idioma será Se utiliza para dividir palabras sin guiones incrustados explícitos.

Si se establece el guión incrustado, se intentará dividir las palabras con guiones incrustados.

El atributo uriWasteReduce controla cómo intentamos dividir los URI largos. Es la fracción de una recta que consideramos un desperdicio excesivo. El valor predeterminado en el módulo reportlab.rl_settings es 0,5 , lo que significa que Intentaremos dividir una palabra que parezca un uri si desperdiciamos al menos la mitad de la línea.

Actualmente, la separación de palabras y la división de uri están desactivadas de forma predeterminada. Necesitas modificar la configuración predeterminada. usando el archivo ~/.rl_settings o agregando un módulo reportlab_settings.py a la ruta de Python. Los valores adecuados son

```
hyphenationLanguage='en_GB'
guion incrustado=1
uriWasteReduce=0.3
```

6.2 Etiquetas de marcado XML de párrafo

El marcado XML se puede utilizar para modificar o especificar el estilo general del párrafo y también para especificar el marcado dentro del párrafo.

La etiqueta <para> más externa

Opcionalmente, el texto del párrafo puede estar rodeado por etiquetas <para atributos....> </para>. Los atributos, si alguno de ellos la etiqueta <para> de apertura afecta el estilo que se utiliza con el texto del párrafo y/o el texto de la viñeta.

Atributo	Sinónimos
alineación	alinear, alineación
permitir huérfanos	permitir huérfanos, permitir huérfanos
permitir viudas	permitir viudas, permitir viudas
autoliderazgo	autoliderazgo, autoliderazgo
color de espalda	color de fondo, color de fondo, bg, bgcolor
color del borde	color de borde, color de borde
fronteraRadio	borderRadius, borderradius
ancho del borde	ancho de borde, ancho de borde
relleno de borde	relleno de borde
balaAncla	banchor, balaAncla, balaAncla
viñetaColor	bcolor, color bala, color bala
balaFuenteNombre	bfont, nombrefuentebala, nombrefuentebala
tamaño de fuente de bala	bfontsize, balaFontSize, balafontsize
balaSangría	se une, balaIndent, boletindent
balaOffsetY	boffsety, balaOffsetY, balaoffsety
incrustadoSeparación de sílabas	Guión incrustado, Guión incrustado
puntos finales	puntos finales, puntos finales
incidente de primera linea	findt, primera líneaindent, primera líneaindent

nombre de la fuente	cara, fuente, nombre de fuente, nombre de fuente
tamaño de fuente	tamaño de fuente, tamaño de fuente, tamaño
separación de palabrasLang	separación de palabrasLang, separación de palabrasLanguage, separación de palabrasidioma
separación de palabrasMinWordLength	separación de palabrasMinWordLength, separación de palabrasminwordlength
separación de palabrasDesbordamiento	separación de palabrasDesbordamiento, separación de palabrasdesbordamiento
justificar descansos	justificar descansos, justificar descansos
justificarLastLine	justificarÚltimaLínea, justificarúltimalínea
principal	principal
sangría izquierda	sangría izquierda, sangría izquierda, lindent
sangría derecha	sangría derecha, sangría derecha, rindent
espaciodespués	espacioDespués, espacioa, espaciodespués
espacioAntes	espacioAntes, espaciob, espacioantes
espacioContracción	contracción del espacio, contracción del espacio
dividirPalabraslargas	dividir palabras largas, dividir palabras largas
huelgaColor	strikeColor, strikcolor
huelgaGap	huelgaGap, huelgaGap
huelgacompensación	strikeoffset, strikeoffset
ancho de huelga	ancho de huelga, ancho de huelga
color de texto	color, fg, color de texto, color de texto
transformación de texto	transformación de texto, transformación de texto
subrayadoColor	subrayadoColor, subrayadocolor
subrayadoGap	subrayadoGap, subrayadoGap
subrayadoDesplazamiento	subrayado compensado, subrayado compensado
subrayadoAncho	ancho de subrayado, ancho de subrayado
uriResiduosReducir	uriWasteReduce, uriwastereduce
convertirse en envoltura	ajuste de palabras, ajuste de palabras

Tabla 6-2 - Sinónimos de atributos de estilo

Se han proporcionado algunos sinónimos útiles para nuestros nombres de atributos de Python, incluidas versiones en minúsculas, y las propiedades equivalentes del estándar HTML donde existan. Estas adiciones hacen que sea mucho más fácil cree aplicaciones de impresión XML, ya que es posible que no sea necesario traducir gran parte del marcado dentro de los párrafos. La siguiente tabla muestra los atributos y sinónimos permitidos en la etiqueta de párrafo más externa.

6.3 Marcado dentro del párrafo

<p>Por la presente se le cobra que el día 28 de mayo, 1970, lo hizo intencionalmente, ilegalmente y <i>con malicia de previsión</i>, publicar una supuesta frase inglés-húngaro libro con la intención de causar un perturbación del orden público. <u>¿Cómo ¿Suplicas</u>?</p>	<p>Por la presente se le acusa de que en el 28 de mayo de 1970, lo hiciste voluntariamente, ilegítimamente y con malicia de premeditado, publicar un supuesto libro de frases inglés-húngaro con la intención provocar una alteración del orden público. <u>Como hacer</u> <u>¿suplicas?</u></p>
--	--

Figura 6-6: Etiquetas simples en negrita y cursiva

```

Este <a href="#MYANCHOR" color="blue">es
un enlace a</a> una etiqueta de anclaje, es decir, <a
name="MYANCHOR"/><font
color="green">aquí</font>. Este <link
href="#MYANCHOR"
color="blue"
fontName="Helvetica">es otro enlace al</link> el mismo
ancla
etiqueta.

```

Este **es un enlace a** una etiqueta de anclaje, es decir, **aquí**.
Este **es otro enlace a** la misma etiqueta ancla.

Figura 6-7: anclajes y enlaces

La etiqueta de enlace se puede utilizar como referencia, pero no como ancla. Las etiquetas de hipervínculo a y link tienen atributos adicionales fontName, fontSize, color y backColor . La referencia del hipervínculo puede tener un esquema de http:(página web externa), pdf:(diferente documento pdf) o documento:(mismo documento pdf); un esquema faltante se trata como documento como es el caso cuando la referencia comienza con # (en cuyo caso el ancla debe omitirlo). Cualquier otro esquema se trata como una especie de URI.

```

<strong>Por la presente se le acusa</
strong> de que, el día 28 de mayo de 1970, usted,
intencionalmente, ilegalmente y <strike>y con malicia de
previsión</strike>, <br/>publicó una supuesta -Libro de
frases en inglés-húngaro
con la intención de provocar una
alteración del orden público.

```

¿Cómo se declara?

Por la presente se le acusa de que el 28 de mayo de 1970 publicó intencionalmente, ilegalmente y con **malicia premeditada un supuesto libro** de frases inglés-húngaro con la intención de alterar el orden público. ¿Cómo se declara?

Figura 6-8: Etiquetas fuertes, de golpe y de rotura

La etiqueta <fuente>

La etiqueta se puede utilizar para cambiar el nombre de la fuente, el tamaño y el color del texto de cualquier subcadena dentro del párrafo. Los atributos legales son tamaño, rostro, nombre (que es lo mismo que rostro), color y fg (que es lo mismo que color). El nombre es el nombre de la familia de fuentes, sin sufijos en "negrita" o "cursiva". Los colores pueden ser nombres de colores HTML o una cadena hexadecimal codificada de diversas formas; consulte reportlab.lib.colors para conocer los formatos permitidos.

```

<font face="times" color="red">Por
la presente se le acusa</font> de que, el 28 de
mayo de 1970, actuó de forma deliberada, ilegal y <font
size=14>con malicia o previsión. </font>, publica un
supuesto libro de frases inglés-húngaro con la intención
de provocar una alteración del orden
público. ¿Cómo se declara?

```

Por la presente se le acusa de que el 28 de mayo de 1970 publicó de forma deliberada, ilegal y con **malicia premeditada** un supuesto libro de frases inglés-húngaro con la intención de provocar una alteración del orden público. ¿Cómo se declara?

Figura 6-9: La etiqueta de fuente

Superíndices y subíndices

Los superíndices y subíndices son compatibles con el

```
Ecuación ( $\alpha$ ): <greek>e</greek>
<super rise=9 size=6><greek>ip</greek></super> = -1
```

Ecuación (a): $e^{\frac{ip}{\alpha}} = -1$

Figura 6-10: Letras y superíndices griegos

Imágenes en línea

Podemos incrustar imágenes en un párrafo con la etiqueta `` que tiene atributos `src`, `ancho`, `alto` cuyos significados son obvios. El atributo `valign` se puede establecer en un valor similar a CSS de "baseline", "sub", "super", "top", "text-top", "middle", "bottom", "text-bottom"; el valor también puede ser un porcentaje numérico o un valor absoluto.

```
<para autoLeading="off" fontSize=12>Este
&lt;img/&gt; 
está alineado <b>top</b>. <br/><br/> Este &lt;img/&gt;  está
alineado <b>abajo</b>. <br/><br/> Este &lt;img/&gt;
 está alineado <b>en el medio</b>.<br/><br/> Este &lt;img/&gt;  está alineado <b>-4</b>.<br/><br/>
Este &lt;img/&gt;  está alineado <b>+4</b>.<br/><br/> Este &lt;img/&gt;  tiene un ancho <b>10</b>.<br/><br/></para>
```

Este `` está **alineado arriba**.

Este `` está **alineado**
Tomás.

Este `` está **alineado en el**
medio.

Este `` está **alineado -4**.

Este `` está **alineado +4**.

Este `` tiene **un ancho de 10**.

Figura 6-11: Imágenes en línea

Las etiquetas `<u>` y `<strike>`

Estas etiquetas se pueden utilizar para realizar subrayados explícitos o tachados. Estas etiquetas tienen atributos de ancho, desplazamiento, color, espacio y tipo. El atributo `tipo` controla cuántas líneas se dibujarán (tipo predeterminado = 1) y cuando `tipo>1` el atributo `espacio` controla la distancia entre líneas.

La etiqueta `<nobr>`

Si la separación de palabras está en funcionamiento, la etiqueta `<nobr>` la suprime para que `<nobr>` una palabra muy larga que no se rompa `</nobr>` no se rompa.

Numeración de párrafos y listas

La etiqueta `<seq>` proporciona soporte integral para listas de numeración, títulos de capítulos, etc. Actúa como una interfaz para la clase `Sequencer` en `reportlab.lib.sequencer`. Se utilizan para numerar títulos y figuras a lo largo de este documento. Puede crear tantos 'contadores' separados como desee, a los que se accede con el atributo `id`; estos se incrementarán en uno cada vez que se acceda a ellos. La etiqueta `seqreset` restablece un contador. Si desea que se reanude desde un número distinto de 1, utilice la sintaxis `<seqreset id="mycounter">`

base="42">. Vamos a intentarlo:

<pre><seq id="spam"/>, <seq id="spam"/>, <seq id="spam"/>, Restablecer<seqreset id="spam"/>. <seq id="spam"/>, <seq id="spam"/>, <seq id="spam"/>.</pre>	1, 2, 3. Reiniciar. 1, 2, 3.
--	------------------------------

Figura 6-12: Secuencias básicas

Puede guardar la especificación de una ID designando una ID de contador como predeterminada usando la etiqueta `<seqdefault id="Counter">`; luego se utilizará siempre que no se especifique una ID de contador. Esto ahorra algo de escritura, especialmente cuando se hacen listas de varios niveles; simplemente cambia la ID del contador al entrar o salir de un nivel.

<pre><seqdefault id="spam"/>Continúa... <seq>, <seq>, <seq>, <seq>, <seq>, <seq>, <seq>.</pre>	Continúa... 4, 5, 6, 7, 8, 9, 10.
--	-----------------------------------

Figura 6-13: La secuencia predeterminada

Finalmente, se puede acceder a secuencias de varios niveles utilizando una variación del formato de cadena de Python y el atributo de plantilla en etiquetas `<seq>`. Esto se utiliza para hacer los títulos de todas las figuras, así como los títulos del nivel dos. La subcadena `%(counter)s` extrae el valor actual de un contador sin incrementarlo; agregar un signo más como en `%(counter)s` incrementa el contador. Los títulos de las figuras utilizan un patrón como el siguiente:

Figura <seq template="%({Chapter}) s-%({FigureNo+s})/"> - Plantillas multinivel	Figura 6-1: Plantillas multinivel
---	-----------------------------------

Figura 6-14: Plantillas multinivel

Hicimos un poco de trampa: el documento real usaba 'Figura', pero el texto de arriba usa 'FigureNo'; de lo contrario, habríamos estropeado nuestra numeración.

6.4 Numeración de viñetas y párrafos

Además de las tres propiedades de sangría, se necesitan algunos otros parámetros para manejar correctamente las listas numeradas y con viñetas. Hablamos de esto aquí porque ya ha visto cómo manejar la numeración. Un párrafo puede tener un argumento BulletText opcional pasado a su constructor; alternativamente, el texto con viñetas se puede colocar en una etiqueta en el encabezado. Este texto se dibujará en la primera línea del párrafo, con su origen x determinado por el atributo BulletIndent del estilo, y en la fuente dada en el atributo BulletFontName. La "viñeta" puede ser un solo carácter como (¡doh!) una viñeta, o un fragmento de texto como un número en alguna secuencia numérica, o incluso un título corto como el que se usa en una lista de definiciones. Las fuentes pueden ofrecer varios caracteres de viñeta, pero sugerimos probar primero la viñeta Unicode (•), que puede escribirse como •; • o (en utf8) \xe2\x80\xaa;:

Atributo	Sinónimos
balaAncla	ancla, balaAncla, balaancla
viñetaColor	color bala, color bala, color, fg
BulletFontName	BulletFontName, BulletFontname, cara, fuente
balaFontSize	balaFontSize, balafontsize, tamaño de fuente, tamaño
balaSangría	balaIndent, boletindent, sangría
balaOffsetY	balaoffsety, offsety

Tabla 6-3 - Atributos y sinónimos de <bullet>

La etiqueta <bullet> solo se permite una vez en un párrafo determinado y su uso anula el estilo de viñeta implícito y el texto de viñeta especificado en la creación del párrafo.

```

alineación = 0
permitirHuérfanos = 0
permitirViudas = 1
backColor = Ninguno borderColor
= Ninguno
borderPadding = 0 borderRadius
= Ninguno borderWidth = 0

```

```

balaAnchor = inicio balaFontName =
Símbolo balaFontSize = 10 balaIndent = 18
incrustadoHyphenation = 1 endDots =
Ninguno firstLineIndent = 0

```

```

fontName = Times-Roman fontSize = 10
hyphenationLang =
en_GB justifyBreaks = 0 justifyLastLine = 0
interlineado = 12 leftIndent = 54
linkUnderline = 0

```

```

rightIndent = 0 spaceAfter = 0
spaceBefore = 0
spaceShrinkage = 0,05
splitLongWords = 1 strikeGap = 1
strikeOffset = 0,25*F

```

```

strikeWidth = textColor =
Color(0,0,0,1) textTransform = Ninguno

espacio de subrayado = 1
desplazamiento de subrayado = -0,125*F
underlineWidth = uriWasteReduce
= 0.3 wordWrap = Ninguno

```

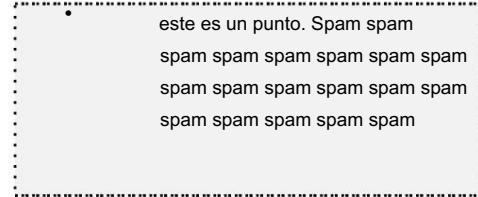


Figura 6-15: Uso básico de viñetas

Se utiliza exactamente la misma técnica para los números, excepto que se utiliza una etiqueta de secuencia. También es posible poner una cadena de varios caracteres en la viñeta; con una sangría profunda y una fuente con viñetas en negrita, puede crear una lista de definiciones compacta.

Capítulo 7 Tablas y estilos de tabla

Las clases Table y LongTable derivan de la clase Flowable y están pensadas como mecanismos de cuadrícula textual simples. La clase longTable utiliza un algoritmo codicioso al calcular el ancho de las columnas y está destinada a tablas largas donde la velocidad cuenta. Las celdas de la tabla pueden contener cualquier cosa que pueda convertirse en una cadena de Python o Flowables (o listas de Flowables).

Nuestras tablas actuales son una compensación entre dibujo eficiente y especificación y funcionalidad. Suponemos que el lector está familiarizado con las tablas HTML. De forma resumida, tienen las siguientes características:

- Pueden contener cualquier cosa convertible en cadena; objetos fluidos como otras mesas; o sub-historias completas.
- Pueden calcular las alturas de las filas para que se ajusten a los datos si no proporciona la altura de las filas. (También pueden calcular los anchos, pero generalmente es mejor que un diseñador establezca el ancho manualmente y se dibuja más rápido).
- Se pueden dividir en páginas si es necesario (consulte el atributo canSplit). Puede especificar que una cantidad de filas en la parte superior e inferior se repitan después de la división (por ejemplo, mostrar los encabezados nuevamente en las páginas 2,3,4...)
- Tienen una notación simple y poderosa para especificar sombreados y líneas de cuadrícula que funciona bien con tablas financieras o de bases de datos, donde no se sabe el número de filas por adelantado. Puedes decir fácilmente "poner la última fila en negrita y poner una línea encima".
- El estilo y los datos están separados, por lo que puede declarar varios estilos de tabla y utilizarlos para una familia de informes. Los orzuelos también pueden 'heredar', como ocurre con los párrafos.

Sin embargo, existe una limitación principal en comparación con una tabla HTML. Definen una cuadrícula rectangular simple. No hay una simple extensión de filas o columnas; Si necesita abarcar celdas, debe anidar tablas dentro de celdas de tabla o utilizar un esquema más complejo en el que la celda principal de un segmento contenga el contenido real.

Las tablas se crean pasando al constructor una secuencia opcional de anchos de columnas, una secuencia opcional de alturas de filas y los datos en orden de filas. El dibujo de la tabla se puede controlar utilizando una instancia TableStyle. Esto permite controlar el color y el grosor de las líneas (si las hay), y la fuente, alineación y relleno del texto. Se proporciona un mecanismo primitivo de cálculo automático de altura de fila o ancho de columna.

7.1 Métodos de usuario de tabla

Estos son los principales métodos que interesan al programador cliente.

```
Tabla(datos, colWidths=Ninguno, rowHeights=Ninguno, estilo=Ninguno, splitByRow=1, repetirRows=0, repetirCols=0,
filaSplitRange=Ninguno, spaceBefore=Ninguno, spaceAfter=Ninguno)
```

El argumento de datos es una secuencia de secuencias de valores de celda, cada una de las cuales debe poder convertirse en un valor de cadena usando la función str o debe ser una instancia de Flowable (como un párrafo) o una lista (o tupla) de dichas instancias. Si el valor de una celda es un Flowable o una lista de Flowables, estos deben tener un ancho determinado o la columna que los contiene debe tener un ancho fijo. La primera fila de valores de celda está en datos [0], es decir, la ésta. los valores están en orden de fila. El valor de la celda i, j está en los datos [i][j]. Los caracteres de nueva línea '\n' en los valores de celda se tratan como caracteres de división de línea y se utilizan en el momento del dibujo para formatear la celda en líneas.

Los otros argumentos son bastante obvios, el argumento colWidths es una secuencia de números o posiblemente Ninguno, que representa los anchos de las columnas. El número de elementos en colWidths determina el número de columnas de la tabla. Un valor de Ninguno significa que el ancho de columna correspondiente debe calcularse automáticamente.

El argumento RowHeights es una secuencia de números o posiblemente Ninguno, que representa las alturas de las filas. El número de elementos en rowHeights determina el número de filas de la tabla. Un valor de Ninguno significa que la altura de fila correspondiente debe calcularse automáticamente.

El argumento de estilo puede ser un estilo inicial para la tabla.

El argumento splitByRow solo es necesario para tablas demasiado altas y demasiado anchas para caber en el contexto actual. En este caso, debe decidir si desea colocar mosaicos hacia abajo y a lo ancho, o a lo ancho y luego hacia abajo. Este parámetro es un booleano que indica que la tabla debe dividirse por fila antes de intentar dividirse por columna cuando

Hay muy poco espacio disponible en el área de dibujo actual y la persona que llama quiere que la tabla se divida. Actualmente, la división de una tabla por columna no está implementada, por lo que establecer `splitByRow` en `False` generará un `NotImplementedError`.

El argumento `repetirRows` especifica el número o una tupla de filas principales que deben repetirse cuando se le pide a la tabla que se divida. Si es una tupla, debe especificar cuál de las filas principales debe repetirse; esto permite casos en los que la primera aparición de la tabla tiene más filas iniciales que partes divididas posteriores. Actualmente, el argumento `repetirCols` se ignora porque una tabla no se puede dividir por columnas.

Los argumentos `spaceBefore` y `spaceAfter` se pueden usar para poner espacio adicional antes o después de la tabla cuando se representa en una historia de ornitorrinco.

El argumento `rowSplitRange` se puede utilizar para controlar la división de la tabla en un subconjunto de sus filas; eso puede ser para evitar que se divida demasiado cerca del principio o del final de la tabla.

`Tabla.setStyle(tblStyle)`

Este método aplica una instancia particular de la clase `TableStyle` (que se analiza a continuación) a la instancia de `Table`.

Esta es la única manera de hacer que las tablas aparezcan con un formato agradable.

Los usos sucesivos del método `setStyle` aplican los estilos de forma aditiva. Es decir, las aplicaciones posteriores anulan las anteriores cuando se superponen.

7.2 Estilo de tabla

Esta clase se crea pasándole una secuencia de comandos, cada comando es una tupla identificada por su primer elemento que es una cadena; los elementos restantes de la tupla de comando representan las coordenadas de celda inicial y final del comando y posiblemente el grosor y los colores, etc.

7.3 Métodos de usuario de `TableStyle`

Estilo de tabla (secuencia de comando)

El método de creación inicializa `TableStyle` con la secuencia de comando de argumento como ejemplo:

```
LIST_STYLE = Estilo de tabla(
    [ ('LINEABOVE', (0,0), (-1,0), 2, colores.verde), ('LINEABOVE', (0,1), (-1,-1),
    0.25, colores.negro ), ('LINE BELOW', (0,-1), (-1,-1), 2, colores.verde), ('ALIGN',
    (1,1), (-1,-1), 'DERECHA ') ] )
```

`TableStyle.add(comandoSecuencia)`

Este método le permite agregar comandos a un `TableStyle` existente, es decir, puede crear `TableStyles` en múltiples declaraciones.

```
LIST_STYLE.add('FONDO', (0,0), (-1,0), colores.Color(0,0.7,0.7))
```

`TableStyle.getCommands()`

Este método devuelve la secuencia de comandos de la instancia.

```
cmds = LIST_STYLE.getCommands()
```

7.4 Comandos de estilo de tabla

Los comandos pasados a `TableStyles` se dividen en tres grupos principales que afectan el fondo de la tabla, dibujan líneas o establecen estilos de celda.

El primer elemento de cada comando es su identificador, el segundo y tercer argumento determinan las coordenadas de celda del cuadro de celdas que se ven afectadas con coordenadas negativas contando hacia atrás desde el límite.

valores como en la indexación de Python . Las coordenadas se dan como (columna, fila) que sigue a la hoja de cálculo 'A1' modelo, pero no el ordenamiento 'RC' más natural (para los matemáticos). La celda superior izquierda es (0, 0) la inferior derecha es (-1, -1). Dependiendo del comando, se producen varios extras (???) en los índices que comienzan en 3 en adelante.

Comandos de formato de celda TableStyle

Todos los comandos de formato de celda comienzan con un identificador, seguido de las definiciones de inicio y detención de celda y los quizás otros argumentos. Los comandos de formato de celda son:

FUENTE	- toma el nombre de la fuente, el tamaño de fuente opcional y el interlineado opcional.
NOMBRE DE FUENTE (o CARA)	- toma el nombre de la fuente.
TAMAÑO DE FUENTE (o TAMAÑO)	- toma el tamaño de fuente en puntos; el líder puede desincronizarse.
PRINCIPAL	- lidera en puntos.
COLOR DE TEXTO	- toma un nombre de color o tupla (R,G,B).
ALINEACIÓN (o ALINEAR)	- toma uno de IZQUIERDA, DERECHA y CENTRO (o CENTRO) o DECIMAL.
IZQUIERDO	- toma un número entero, por defecto es 6.
ACOLCHADO DERECHO	- toma un número entero, por defecto es 6.
ACOLCHADO INFERIOR	- toma un número entero, por defecto es 3.
AÑADIR SUPERIOR	- toma un número entero, por defecto es 3.
FONDO	- toma un color definido por un objeto, nombre de cadena o tupla/lista numérica, o toma una lista/tupla que describe un relleno degradado deseado que debería contener tres elementos de la forma [DIRECCIÓN, startColor, endColor] donde la DIRECCIÓN es VERTICAL u HORIZONTAL.
FONDOS DE FILA	- toma una lista de colores que se utilizarán cíclicamente.
COLANTES	- toma una lista de colores que se utilizarán cíclicamente.
VALIGAR	- toma uno de ARRIBA, MEDIO o ABAJO predeterminado

Esto establece el color de fondo de la celda en las celdas relevantes. El siguiente ejemplo muestra el ANTECEDENTES, y comandos TEXTCOLOR en acción:

```
datos = [['00', '01', '02', '03', '04'],
         ['10', '11', '12', '13', '14'],
         ['20', '21', '22', '23', '24'],
         ['30', '31', '32', '33', '34']]
t=Tabla(datos)
t.setStyle(TableStyle([('FONDO',(1,1),(-2,-2),colores.verde),
                      ('TEXTCOLOR',(0,0),(1,-1),colores.rojo)]))
```

produce

00	01	02	03	04
10	11	12	13	14
20	21	22	23	24
30	31	32	33	34

Para ver los efectos de los estilos de alineación necesitamos algunos anchos y una cuadrícula, pero debería ser fácil ver dónde los estilos provienen.

```
datos = [['00', '01', '02', '03', '04'],
         ['10', '11', '12', '13', '14'],
         ['20', '21', '22', '23', '24'],
         ['30', '31', '32', '33', '34']]
t=Tabla(datos,5*[0.4*pulgadas], 4*[0.4*pulgadas])
t.setStyle(TableStyle([('ALIGN',(1,1),(-2,-2),'DERECHA'),
                      ('TEXTCOLOR',(1,1),(-2,-2),colores.rojo),
                      ('VALIGN',(0,0),(0,-1),'ARRIBA'),
                      ('TEXTCOLOR',(0,0),(0,-1),colores.azul),
                      ('ALINEAR',(0,-1),(-1,-1),'CENTRO'),
                      ('VALIGN',(0,-1),(-1,-1),'MEDIO'),
                      ('TEXTCOLOR',(0,-1),(-1,-1),colores.verde),
                      ('INNERGRID', (0,0), (-1,-1), 0.25, colores.negro),
                      ('CAJA', (0,0), (-1,-1), 0.25, colores.negro),
                      ]))
```

produce

00				
01	02	03	04	

10		11	12	13	14
20		21	22	23	24
30	31	32	33	34	

Comandos de línea TableStyle

Los comandos de línea comienzan con el identificador, las coordenadas de inicio y finalización de la celda y siempre siguen con el grosor (en puntos) y el color de las líneas deseadas. Los colores pueden ser nombres o pueden especificarse como una tupla (R, G, B), donde R, G y B son flotantes y (0, 0, 0) es negro. Los nombres de los comandos de línea son: GRID, BOX, OUT-LINE, INNERGRID, LINEBELOW, LINEABOVE, LINEBEFORE y LINEAFTER. BOX y OUTLINE son equivalentes, y GRID es el equivalente de aplicar tanto BOX como INNERGRID.

Podemos ver algunos comandos de línea en acción con el siguiente ejemplo.

```
datos = [['00', '01', '02', '03', '04'], ['10', '11', '12', '13', '14'], ['20', '21', '22', '23', '24'], ['30', '31', '32', '33', '34']]

t=Tabla(datos,estilo=[('GRID',(1,1),(-2,-2),1,colores.verde), ('BOX',(0,0),(1,-1),2,colores.rojo),
                      ('LINEABOVE',(1,2),(-2,2),1,colores.azul),
                      ('LINEBEFORE',(2,1),(2,-2),1,colores.rosa),])
```

produce

00	01	02	03	04	
10	11	12	13	14	
20	21	22	23	24	
30	31	32	33	34	

Los comandos de línea causan problemas a las tablas cuando se dividen; El siguiente ejemplo muestra una tabla dividida en varias posiciones.

```
datos = [['00', '01', '02', '03', '04'], ['10', '11', '12', '13', '14'], ['20', '21', '22', '23', '24'], ['30', '31', '32', '33', '34']]

t=Tabla(datos,estilo=[('GRID',(0,0),(-1,-1),0.5,colores.gris), ('GRID',(1,1),
                      (-2,-2),1,colores.verde ), ('CAJA',(0,0),(1,-1),2,colores.rojo),
                      ('CAJA',(0,0),(-1,-1),2,colores.negro), ('LINEABOVE',
                      (1,2),(-2,2),1,colores.azul), ('LINEBEFORE',(2,1),
                      (2,-2),1,colores.rosa), ('FONDO', (0, 0), (0, 1), colores.rosa),
                      ('FONDO', (1, 1), (1, 2), colores.lavanda), ('FONDO ', (2, 2), (2, 3),
                      colores.naranja),])
```

produce

00	01	02	03	04	
10	11	12	13	14	
20	21	22	23	24	
30	31	32	33	34	

00	01	02	03	04	
----	----	----	----	----	--

10	11	12	13 14
20	21	22	23 24
30	31	32	33 34

00	01	02	03 04
10	11	12	13 14
20	21	22	23 24
30	31	32	33 34

Cuando se desdivide y se divide en la primera o segunda fila.

Valores de celda complejos

Como se mencionó anteriormente, podemos tener valores de celda complicados, incluidos párrafos, imágenes y otros elementos fluidos o listas de los mismos. Para ver esto en funcionamiento, considere el siguiente código y la tabla que produce. Tenga en cuenta que la imagen tiene un fondo blanco que oscurecerá cualquier fondo que elija para la celda. Para obtener mejores resultados debes utilizar un fondo transparente.

```
I = Imagen("../imagenes/relogo.gif")
I.drawHeight = 1,25*pulgada*I.drawHeight / I.drawWidth I.drawWidth = 1,25*pulgada P0 = Párrafo("")
```

```
<b>Un pa<font color=red>r</font>un<i>gráfico</i></b> <super><font
color=amarillo>1</font></super>" , hoja de estilo["TextoCuerpo"])
```

```
P = Párrafo("")
<para align=center spaceb=3>El <b>logotipo</font></b> izquierdo de
<b>ReportLab
Imagen</para>" , hoja de
estilo["BodyText"]) datos= [[{'A',
'ANTES DE CRISTO',
[P0, 'D'],
['00', '01', '02', [I,P], '04'],
['10', '11', '12', [P,I], '14'], ['20', '21', '22', '23', '24'],
['30', '31', '32', '33', '34']]
```

```
t=Tabla(datos,estilo=[('GRID',(1,1),(-2,-2),1,colores.verde), ('BOX',(0,0),(1,-1),2,colores.rojo),
('LINEABOVE',(1,2),(-2,2),1,colores.azul),
('LINEBEFORE',(2,1),(2,-2),1,colores.rosa), ('FONDO', (0, 0), (0,
1), colores.rosa), ('FONDO', (1, 1), (1, 2), colores.Javanda),
('FONDO', (2, 2), (2, 3), colores.naranja), ('CAJA',(0,0),
(-1,-1),2,colores.negro ), ('GRID',(0,0),(-1,-1),0.5,colors.black), ('VALIGN',(3,0),
(3,0),'BOTTOM'), ('FONDO',(3,0),(3,0),colores.verde_lima), ('FONDO',(3,1),
(3,1),colores.caqui), ('ALINEAR',( 3,1),(3,1),'CENTRO'),
('FONDO',(3,2),(3,2),colores.beige), ('ALINEAR',(3,2),
( 3,2),'IZQUIERDA'), ])
```

```
t._argW[3]=1,5*pulgada
```

```
produce
```

A	B	C	Un párrafo	D
---	---	---	------------	---

00	01	02	 La izquierda de ReportLab Logotipo	04
10	11	12	La izquierda de ReportLab Logotipo	14
20	21	22	23	24
30	31	32	33	34

Comandos de extensión de estilo de tabla

Nuestras clases Table admiten el concepto de expansión, pero no se especifica de la misma manera que html. El estilo especificación

SPAN, (sc,sr), (ec,er)

indica que las celdas en las columnas sc - ec y las filas sr - er deben combinarse en una súper celda con contenidos determinados por la celda (sc, sr). Las otras celdas deberían estar presentes, pero deberían contener cadenas vacías. o puede obtener resultados inesperados.

```
datos=[['Arriba\nIzquierda', "", '02', '03', '04'],
      [",", '12', '13', '14'],
      ['20', '21', '22', 'Abajo\nDerecha', ""],
      ['30', '31', '32', "", "]]
t=Tabla(datos,estilo=[

    ('GRID',(0,0),(-1,-1),0.5,colores.gris),
    ('FONDO',(0,0),(1,1),colores.verde_pálido),
    ('SPAN',(0,0),(1,1)),
    ('FONDO',(-2,-2),(-1,-1), colores.rosa),
    ('SPAN',(-2,-2),(-1,-1)),
])
]

```

produce

Arriba Izquierda	02	03	04	
	12	13	14	
20	21	22		Abajo
30	31	32	Derecha	

Observe que no necesitamos ser conservadores con nuestro comando GRID. Las celdas distribuidas no se dibujan. a través de.

Comandos varios de TableStyle

Para controlar la división de tablas se puede utilizar el comando NOSPLIT. La especificación de estilo.

NOSPLIT, (sc,sr), (ec,er)

exige que las celdas de las columnas sc - ec y las filas sr - er no se puedan dividir.

Índices de estilo de tabla especiales

En cualquier comando de estilo, el índice de la primera fila se puede establecer en una de las cadenas especiales 'splitlast' o 'splitfirst' para indicar que el estilo debe usarse solo para la última fila de una tabla dividida o la primera fila de una continuación. Esto permite dividir tablas con mejores efectos alrededor de la división.

Capítulo 8 Programación de flujos

Los siguientes flowables le permiten evaluar y ejecutar condicionalmente expresiones y declaraciones en el momento del cierre:

8.1 DocAssign(self, var, expr, vida='para siempre')

Asigna una variable de nombre var a la expresión expr. P.ej:

```
Asignación de documento('i',3)
```

8.2 DocExec(self, stmt, toda la vida='para siempre')

Ejecuta la sentencia stmt. P.ej:

```
DocExec('i=1')
```

8.3 DocPara(self, expr, format=Ninguno, estilo=Ninguno, klass=Ninguno, escape=True)

Crea un párrafo con el valor de expr como texto. Si se especifica el formato, debe usar %(__expr__)s para la interpolación de cadenas de la expresión expr (si corresponde). También puede utilizar interpolaciones de %(name)s para otras variables en el espacio de nombres. P.ej:

```
DocPara('i',format='El valor de i es %(__expr__d',style=normal)
```

8.4 DocAssert(self, cond, formato=Ninguno)

Genera un AssertionError que contiene la cadena de formato si cond se evalúa como False.

```
DocAssert(val, 'val es falso')
```

8.5 DocIf(self, cond, luegoBloquear, elseBloque=[])

Si cond se evalúa como Verdadero, este fluido se reemplaza por el bloque elsethe elseBlock.

```
DocIf('i>3',Paragraph('El valor de i es mayor que 3',normal),  
Párrafo('El valor de i no es mayor que 3',normal))
```

8.6 DocWhile(self, cond, whileBlock)

Ejecuta whileBlock mientras cond se evalúa como True. P.ej:

```
Asignación de documento('i',5)  
DocWhile('i',[DocPara('i',format='El valor de i es %(__expr__d',style=normal),DocExec('i=1'))])
```

Este ejemplo produce un conjunto de párrafos del formulario:

```
El valor de i es 5  
El valor de i es 4  
El valor de i es 3  
El valor de i es 2  
El valor de i es 1
```

Capítulo 9 Otros fluidos útiles

9.1 Preformato(texto, estilo, bulletText=Ninguno, dedent=0, maxLineLength=Ninguno, splitChars=Ninguno, newLineChars=Ninguno)

Crea un párrafo preformato que no se ajusta, divide líneas ni realiza otras manipulaciones. En el texto no se tienen en cuenta etiquetas de estilo XML. Si la sangría es distinta de cero, los espacios iniciales comunes se eliminarán del frente de cada línea.

Definición de una longitud máxima de línea

Puede utilizar la propiedad maxLineLength para definir una longitud máxima de línea. Si la longitud de una línea excede este valor máximo, la línea se dividirá automáticamente.

La línea se dividirá en cualquier carácter definido en splitChars. Si no se proporciona ningún valor para esta propiedad, la línea se dividirá en cualquiera de los siguientes caracteres estándar: espacio, dos puntos, punto, punto y coma, coma, guión, barra diagonal, barra diagonal inversa, paréntesis izquierdo, cuadrado izquierdo corchete y llave izquierda

Los caracteres se pueden insertar automáticamente al principio de cada línea que se haya creado. Puede configurar la propiedad newLineChars para los caracteres que desee utilizar.

```
de reportlab.lib.styles importar getSampleStyleSheet hoja de estilo=getSampleStyleSheet() normalStyle =
hoja de estilo["Código"] texto="" clase XPreformatted(Párrafo):
def __init__(self, text, style, bulletText = Ninguno, frags=Ninguno,
caseSensitive
=1):

    self.caseSensitive = caseSensitive si la longitud máxima de la línea
    y el texto:
        texto = self.stopLine(texto, longitud máxima de línea, caracteres divididos) limpiador = texto lambda, dedent=dedent:
        ".join(_dedenter(texto o ",dedent)) self._setup(texto, estilo, viñetaTexto, frags, limpiador)
    ...

    t=Preformato(texto,normalStyle,maxLineLength=60, newLineChars='')

produce la
clase XPreformatoada(Párrafo): def __init__(self,
    text, style, bulletText = Ninguno,
    > frags=Ninguno, caseSensitive=1):
    self.caseSensitive = caseSensitive si longitud de línea máxima y
    texto: texto = self.stopLine(texto, longitud de línea máxima,
    > splitCharacters) limpiador = texto lambda, dedent=dedent: ".join( > _dedenter(texto o
    ",dedent)) self._setup(texto, estilo,
    viñetaTexto, frags, limpiador)
```

9.2 XReformatted(texto, estilo, bulletText=Ninguno, dedent=0, frags=Ninguno)

Esta es una forma sin reorganización de la clase Paragraph; Las etiquetas XML están permitidas en el texto y tienen el mismo significado que para la clase Paragraph. En cuanto a Preformato, si la sangría es distinta de cero, los espacios iniciales comunes se eliminarán del frente de cada línea.

```
de reportlab.lib.styles importar getSampleStyleSheet hoja de estilo=getSampleStyleSheet() normalStyle =
hoja de estilo["Código"] texto=""
```

Esta es una forma sin reorganización de la clase Paragraph; Las etiquetas XML están permitidas en el <i>texto</i> y tienen el mismo

significados que para la clase Párrafo. En cuanto a Preformato, si la sangría es distinta de cero dedent

Los espacios iniciales comunes se eliminarán del frente de cada línea.

Puedes tener ' entidades de estilo también para < y > .

'''

```
t=XPreformatado(texto,estilonormal,dedent=3)
```

produce

Esta es una forma sin reorganización de la clase Paragraph ;
Las etiquetas XML están permitidas en el texto y tienen el mismo

significados que para la clase Párrafo .

En cuanto a Preformatado , si la sangría es distinta de cero, los espacios iniciales comunes se
eliminarán del frente de cada línea.

Puedes tener ' entidades de estilo también para < y > .

9.3 Imagen (nombre de archivo, ancho = Ninguno, alto = Ninguno)

Cree un fluido que contendrá la imagen definida por los datos en el nombre del archivo, que puede ser una ruta de archivo, un objeto similar a un archivo o una instancia de reportlab.graphics.shapes.Drawing. Se admite el tipo de imagen PDF predeterminado, jpeg , y si está instalada la extensión PIL para Python , también se pueden manejar otros tipos de imágenes. Si se especifica el ancho o el alto, determinan la dimensión de la imagen mostrada en puntos. Si alguna de las dimensiones no se especifica (o se especifica como Ninguna), se supone que la dimensión de píxeles correspondiente de la imagen está en puntos y se utiliza.

```
Imagen("lj8100.jpg")
```

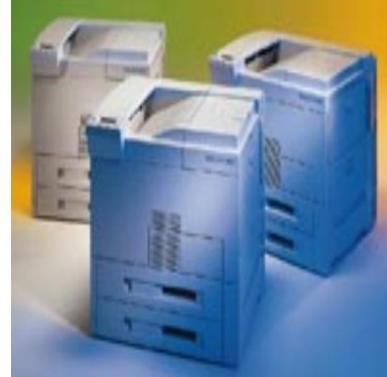
se mostrará como



mientras

```
im = Imagen("lj8100.jpg", ancho=2*pulgada, alto=2*pulgada) im.hAlign = 'CENTRO'
```

produce



9.4 Espaciador (ancho, alto)

Esto hace exactamente lo que se esperaría; agrega una cierta cantidad de espacio a la historia. Actualmente esto sólo funciona para espacios verticales.

9.5 Salto de página()

Este Flowable representa un salto de página. Funciona consumiendo efectivamente todo el espacio vertical que se le asigna.

Esto es suficiente para un solo documento de marco, pero solo sería un salto de marco para múltiples marcos, por lo que el mecanismo BaseDocTemplate detecta los saltos de página internamente y los maneja de manera especial.

9.6 CondPageBreak (altura)

Este Flowable intenta forzar una ruptura de marco si no queda suficiente espacio vertical en el marco actual.

Por lo tanto, probablemente tenga un nombre incorrecto y probablemente debería cambiarse el nombre a CondFrameBreak.

9.7 KeepTogether (fluidos)

Este Flowable compuesto toma una lista de Flowables e intenta mantenerlos en el mismo marco. Si la altura total de los Flowables en la lista de flowables excede el espacio disponible del marco actual, entonces se utiliza todo el espacio y se fuerza una ruptura de marco.

9.8 Tabla de contenido()

Se puede generar una tabla de contenidos utilizando el fluido TableOfContents. Se necesitan los siguientes pasos para agregar una tabla de contenido a su documento:

Cree una instancia de TableOfContents. Anule los estilos de nivel (opcional) y agregue el objeto a la historia:

```
toc = TablaDeContenido()
PS = ParagraphStyle
toc.levelStyles =
    [ PS(fontName='Times-Bold', fontSize=14, nombre='TOCHeading1',
        leftIndent=20, firstLineIndent=-20, spaceBefore=5, interlineado=16),
        PD(fontSize=12, nombre='TOCHeading2', leftIndent=40,
            firstLineIndent=-20, spaceBefore=0, líder=12),
        PD(fontSize=10, nombre='TOCHeading3', leftIndent=60,
            firstLineIndent=-20, spaceBefore=0, líder=12),
        PD(fontSize=10, nombre='TOCHeading4', leftIndent=100,
            firstLineIndent=-20, spaceBefore=0, líder=12),
    ]
historia.append(toc)
```

Las entradas a la tabla de contenido se pueden realizar manualmente llamando al método addEntry en el objeto TableOfContents o automáticamente enviando una notificación 'TOCEntry' en el método afterFlowable del DocTemplate que está utilizando. Los datos que se deben pasar para notificar son una lista de tres o cuatro elementos que contienen un número de nivel, el texto de la entrada, el número de página y una clave de destino opcional a la que debe apuntar la entrada. Esta lista generalmente se creará en un método de plantilla de documento como afterFlowable(), realizando llamadas de notificación usando el método notify() con datos apropiados como este:

```
def afterFlowable(self, flowable): """Detecta encabezados
de nivel 1 y 2, crea un esquema y realiza un seguimiento del título del
capítulo."""
if isinstance(flowable, Paragraph):
    txt = flowable.getText()
    if style == 'Heading1': # ...
        self.notify('TOCEntry', (1, txt, self.page))

estilo elif == 'Encabezado2': # ... clave =
    'h2-%s'
    % self.seq.nextf('encabezado2') self.canv.bookmarkPage(clave)
    self.notify('TOCEntry', (1, txt, self.page,
    clave)) # ...
```

De esta manera, siempre que se agregue a la historia un párrafo de estilo 'Encabezado1' o 'Encabezado2', aparecerá en la tabla de contenido. Se podrá hacer clic en las entradas del Título 2 porque se ha proporcionado una clave marcada.

Finalmente, debe utilizar el método multiBuild de DocTemplate porque las tablas de contenido necesitan varias pasadas para generarse:

```
doc.multiBuild(historia)
```

A continuación se muestra un ejemplo simple pero funcional de un documento con una tabla de contenido:

```
de reportlab.lib.styles importar ParagraphStyle como PS de reportlab.platypus importar
PageBreak de reportlab.platypus.paragraph importar párrafo de
reportlab.platypus doctemplate importar PageTemplate, BaseDocTemplate de
reportlab.platypus.tableofcontents importar TableOfContents de reportlab.platypus.frames importar marco desde reportlab.lib.units
importar cm

clase MiPlantillaDoc(PlantillaDocBase):

    def __init__(self, nombre de archivo, **kw): self.allowSplitting
        = 0 BaseDocTemplate.__init__(self,
            nombre de archivo, **kw) plantilla = PageTemplate('normal', [Frame(2.5*cm,
            2.5*cm, 15* cm, 25*cm, id='F1')]) self.addPageTemplates(plantilla)

    def afterFlowable(self, flowable): "Registra entradas TOC".
        if flowable._class_._name_ ==
            'Párrafo': texto = flowable.getText() estilo = flowable.style.name if estilo ==
                'Encabezado1':

            self.notify('TOCEntry', (0, texto, self.page))
        si estilo == 'Título2':
            self.notify('TOCEntry', (1, texto, self.page))

h1 = PS(nombre = 'Título1', tamaño de fuente
        = 14, interlineado = 16)

h2 = PS(nombre = 'Encabezado2',
        tamaño de fuente =
        12, interlineado =
        14, sangría izquierda = delta)

# Construir historia.
historia = [] toc =
TablaDeContenidos()
# Para ser concisos, utilizamos los mismos estilos para los títulos y las entradas TOC toc.levelStyles = [h1, h2]
story.append(toc) story.append(PageBreak())
story.append(Paragraph('First
header', h1)) story.append(Paragraph('Texto
en el primer subtítulo', PS('cuerpo'))) story.append(Paragraph('Primer
subtítulo', h2)) story.append(Paragraph('Texto en el primer subtítulo', PS('cuerpo'))) story.append(PageBreak())
story.append(Paragraph('Segundo subtítulo', h2)) story.append(Paragraph('Texto en
el segundo subtítulo', PS('cuerpo '))) story.append(Párrafo('Último título', h1))

doc = MyDocTemplate('mintoc.pdf') doc.multiBuild(historia)
```

9.9 Índice simple()

Se puede generar un índice utilizando el fluido SimpleIndex. Se necesitan los siguientes pasos para agregar un índice a su documento:

Utilice la etiqueta de índice en los párrafos para indexar términos:

```
historia = []
```

...

```
story.append('La tercera <index item="word" />palabra de este párrafo está indexada.')
```

Crea una instancia de SimpleIndex y agrégala a la historia donde quieras que aparezca:

```
índice = SimpleIndex(punto='.', encabezados=encabezados)
story.append(índice)
```

Los parámetros que puede pasar al constructor SimpleIndex se explican en la referencia de reportlab.

Ahora, cree el documento utilizando el creador de lienzos devuelto por SimpleIndex.getCanvasMaker():

```
doc.build(historia, canvasmaker=index.getCanvasMaker())
```

Para crear un índice con múltiples niveles, pase una lista de elementos separados por comas al atributo de elemento de una etiqueta de índice:

```
<índice de elemento="terma,termb,termc" />
<índice de elemento="terma,termd" />
```

terma representará el nivel más alto y termc el término más específico. termb y termc aparecerán en el mismo nivel dentro de terma.

Si necesita indexar un término que contiene una coma, deberá duplicarla para escapar de ella. Para evitar la ambigüedad de tres comas consecutivas (una coma con escape seguida de un separador de lista o un separador de lista seguido de una coma con escape?), introduzca un espacio en la posición correcta. Se eliminarán los espacios al principio o al final de los términos.

```
< elemento de índice="coma(,,),,,,...           " />
```

Esto indexa los términos "coma (,)", "," y "...".

9.10 ListaFlowable(), ListItem()

Utilice estas clases para hacer listas ordenadas y desordenadas. Las listas se pueden anidar.

ListFlowable() creará una lista ordenada, que puede contener cualquier fluido. La clase tiene una serie de parámetros para cambiar la fuente, el color, el tamaño, el estilo y la posición de los números de lista o de las viñetas en listas desordenadas. El tipo de numeración también se puede configurar para usar letras minúsculas o mayúsculas ('A,B,C', etc.) o números romanos (mayúsculas o minúsculas) usando la propiedad BulletType. Para cambiar la lista a un tipo desordenado, establezca bulletType='bullet'.

Los elementos dentro de una lista ListFlowable() se pueden cambiar desde su apariencia predeterminada envolviéndolos en una clase ListItem() y configurando sus propiedades.

Lo siguiente creará una lista ordenada y establecerá el tercer elemento en una sublista desordenada.

```
de reportlab.platypus importar ListFlowable, ListItem de reportlab.lib.styles importar
getSampleStyleSheet estilos = getSampleStyleSheet() estilo = estilos["Normal"] t =
ListFlowable( [ Paragraph("Item no.1", estilo),
ListItem(Paragraph( "Artículo n° 2",
estilo),bulletColor="verde",valor=7, ListFlowable(
[ Párrafo ("sublista elemento 1", estilo),
ListItem(Paragraph("sublista elemento 2", estilo),bulletColor='rojo',valor='cuadrado') ], bulletType='bullet', start='cuadrado', ),
Párrafo("Artículo n°4", estilo), ], tipo bala='l' )
```

produce i

Artículo no.1

vii También no. 2

viii ■ sublista elemento 1

■ elemento de sublista 2

ix Artículo nº4

Para hacer frente al anidamiento, el parámetro de inicio se puede configurar en una lista de posibles inicios; para ul los inicios aceptables son cualquier carácter Unicode o nombres específicos conocidos por flowables.py, por ejemplo, bala, círculo, cuadrado, disco, diamante, diamante wx, arrowhead, sparkle, squarelrs o blackstar. Para el viejo

El inicio puede ser cualquier carácter de '1iaAl' para indicar diferentes estilos de números.

9.11 Columnas equilibradas()

Utilice la clase BalancedColumns para crear un fluido que divida su contenido en dos o más columnas de tamaño aproximadamente igual. Efectivamente se sintetizan n fotogramas para tomar el contenido y los intentos fluidos. para equilibrar el contenido entre ellos. Los marcos creados se dividirán cuando la altura total sea demasiado grande y la división mantendrá el equilibrio.

```
desde reportlab.platypus importar BalancedColumns
desde reportlab.platypus.frames importar ShowBoundaryValue
F = [
    lista de fluidos.....
]
historia.append(
    Equilibrado(
        F,                      #los fluidos que estamos equilibrando
        nCols = 2, #el número de columnas
        necesario = 72,#el espacio mínimo que necesita el fluido
        spacAntes = 0,
        espacioDespués = 0,
        showBoundary = Ninguno,      #mostración de límites opcional
        leftPadding = Ninguno,       #estos anulan el marco creado
        rightPadding = Ninguno,      #paddings si se especifica lo contrario
        topPadding = Ninguno,        #rellenos de marco predeterminados
        bottomPadding = Ninguno,     #son usados
        InnerPadding = Ninguno,      #el espacio entre marcos si se especifica lo contrario
        nombre="",
        endSlack=0.1,)             #use max(relleno izquierdo,relleno derecho)
                                    #para fines de identificación cuando algo sale mal
                                    #asignación por disparidad de altura, es decir, 10% de la altura disponible
    )
)
```

Capítulo 10 Escribir tus propios objetos fluidos

Flowables pretende ser un estándar abierto para crear contenido de informes reutilizable y usted puede crear fácilmente sus propios objetos. Esperamos que con el tiempo construyamos una biblioteca de contribuciones, brindando a los usuarios de Reportlab una rica selección de tablas, gráficos y otros "widgets de informes" que puedan usar en sus propios informes. Esta sección le muestra cómo crear sus propios fluidos. Deberíamos poner la clase

Figura en la biblioteca estándar, ya que es una base muy útil.

10.1 Un fluido muy simple

Recuerde la función de mano de la sección pdfgen de esta guía del usuario que generó un dibujo de una mano como una figura cerrada compuesta de curvas de Bézier.

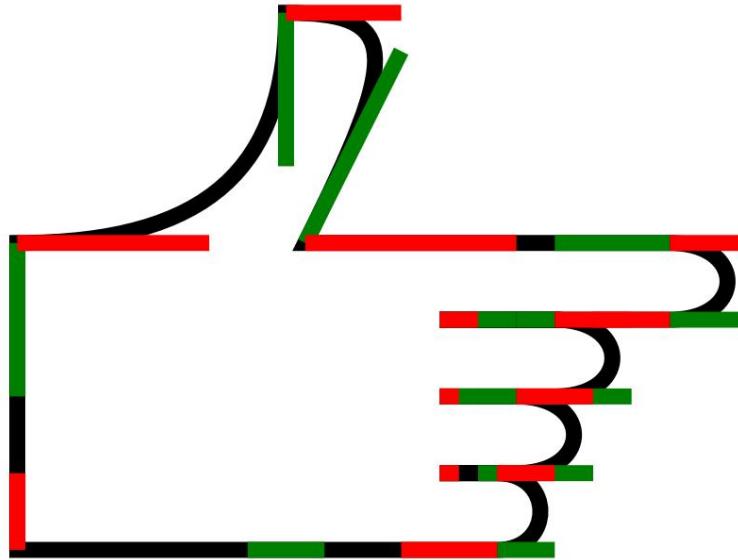


Figura 10-1: una mano

Para incrustar este o cualquier otro dibujo en un flowable Platypus debemos definir una subclase de Flowable con al menos un método wrap y un método draw.

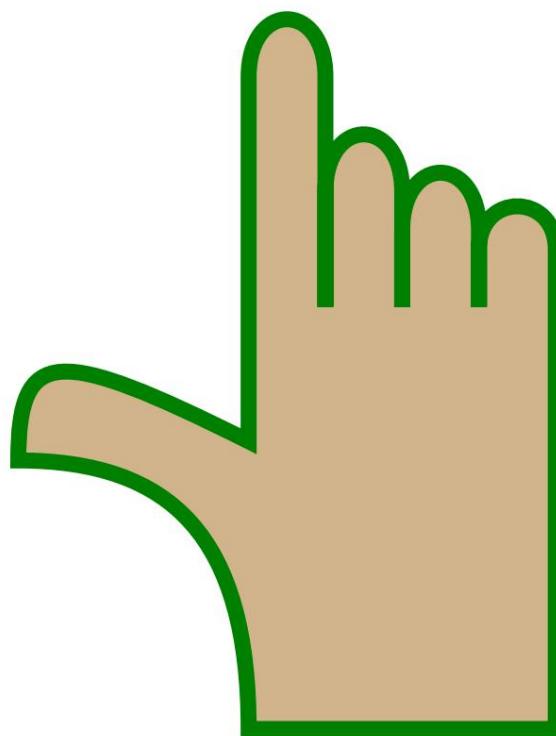
```
de reportlab.platypus.flowables importar Flowable de reportlab.lib.colors importar clase bronceado
y verde HandAnnotation(Flowable): ""Una mano fluida.""
```

```
def __init__(self, xoffset=0, tamaño=Ninguno, color de relleno=bronceado, color de trazo=verde):
    de reportlab.lib.units importe pulgadas si el tamaño es Ninguno: tamaño
        = 4*pulgadas self.fillcolor, self.strokecolor = fillcolor,
    trazocolor self.xoffset = xoffset self.size = tamaño # el tamaño normal es 4 pulgadas self.scale = tamaño /
    (4.0*pulgada) def wrap(self, *args): return
    (self.xoffset, self.size) def
    draw(self): lienzo = self.canv lienzo.setLineWidth(6)
    lienzo.setFillColor(self.fillcolor)
    canvas.setStrokeColor(self.strokecolor)
    canvas.translate(self.xoffset+self.size,0) canvas.rotate(90)
    canvas.scale(self.scale,
    self.scale) hand(canvas, debug=0,
    fill= 1)
```

El método de ajuste debe proporcionar el tamaño del dibujo; lo utiliza el bucle principal de Platypus para decidir si este elemento cabe en el espacio restante en el fotograma actual. El método de dibujo realiza el dibujo.

del objeto después de que el bucle principal de Platypus haya trasladado el origen (0,0) a una ubicación adecuada en un marco apropiado.

A continuación se muestran algunos ejemplos de usos del fluido HandAnnotation.



El valor por defecto.



Sólo una pulgada de alto.



De una pulgada de alto y desplazado hacia la izquierda con azul y cian.

10.2 Modificación de un fluido incorporado

Para modificar un fluido existente, debe crear una clase derivada y anular los métodos que necesita cambiar para obtener el comportamiento deseado.

Como ejemplo, para crear una imagen rotada, debe anular los métodos de ajuste y dibujo de la clase Im-age existente.

```
clase RotatedImage(Imagen): def
    wwrap(self.availWidth,availHeight): h, w =
        Image.wrap(self,availHeight,availWidth) return w, h def draw(self):
            self.canv.rotate(90)
```

```
Imagen.dibujar(yo)
I = ImagenRotada('../images/replogo.gif')
I.hAlign = 'CENTRO'
```

produce



Capítulo 11 Gráficos

11.1 Introducción

ReportLab Graphics es uno de los subpaquetes de la biblioteca ReportLab. Comenzó como un conjunto de programas independiente, pero ahora es una parte totalmente integrada del conjunto de herramientas ReportLab que le permite utilizar sus potentes funciones de gráficos y gráficos para mejorar sus formularios e informes PDF.

11.2 Conceptos generales

En esta sección presentaremos algunos de los principios más fundamentales de la biblioteca de gráficos, que aparecerán más adelante en varios lugares.

Dibujos y renderizados

Un dibujo es una descripción independiente de la plataforma de una colección de formas. No está asociado directamente con PDF, Postscript ni ningún otro formato de salida. Afortunadamente, la mayoría de los sistemas de gráficos vectoriales han seguido el modelo Postscript y es posible describir formas sin ambigüedades.

Un dibujo contiene varias formas primitivas. Las formas normales son aquellas ampliamente conocidas como rectángulos, círculos, líneas, etc. Una forma especial (lógica) es un Grupo, que puede contener otras formas y aplicarles una transformación. Los grupos representan compuestos de formas y permiten tratar el compuesto como si fuera una sola forma. Casi cualquier cosa se puede construir a partir de una pequeña cantidad de formas básicas.

El paquete proporciona varios renderizadores que saben cómo dibujar un dibujo en diferentes formatos. Estos incluyen PDF (renderPDF), Postscript (renderPS) y salida de mapa de bits (renderPM). El renderizador de mapas de bits utiliza el rasterizador libart de Raph Levien y la biblioteca de imágenes Python (PIL) de Fredrik Lundh. El renderizador SVG utiliza los módulos XML de la biblioteca estándar de Python, por lo que no es necesario instalar el paquete adicional de XML-SIG llamado PyXML. Si tiene instaladas las extensiones adecuadas, puede generar dibujos en forma de mapa de bits para la web, así como en forma vectorial para documentos PDF, y obtener un "resultado idéntico".

El renderizador de PDF tiene "privilegios" especiales: un objeto de dibujo también es un fluido y, por lo tanto, puede colocarse directamente en la historia de cualquier documento Platypus o dibujarse directamente en un lienzo con una línea de código. Además, el procesador de PDF tiene una función de utilidad para crear rápidamente un documento PDF de una página.

El renderizador SVG es especial ya que todavía es bastante experimental. El código SVG que genera no está realmente optimizado de ninguna manera y asigna solo las funciones disponibles en ReportLab Graphics (RLG) a SVG. Esto significa que no hay soporte para animación SVG, interactividad, secuencias de comandos o formas de recorte, enmascaramiento o graduación más sofisticadas. Así que tenga cuidado e informe cualquier error que encuentre.

Sistema coordinado

La dirección Y en nuestro sistema de coordenadas XY apunta de abajo hacia arriba. Esto es consistente con PDF, Post-script y notación matemática. También parece más natural para las personas, especialmente cuando trabajan con gráficos. Tenga en cuenta que en otros modelos gráficos (como SVG) la coordenada Y apunta hacia abajo. Para el renderizador SVG, esto en realidad no es un problema, ya que tomará sus dibujos y los volteará según sea necesario, para que su salida SVG se vea tal como se esperaba.

La coordenada X apunta, como siempre, de izquierda a derecha. Hasta ahora no parece haber ningún modelo que abogue por la dirección opuesta, al menos no todavía (con interesantes excepciones, al parecer, para los árabes que miran gráficos de series temporales...).

Empezando

Creemos un dibujo simple que contenga la cadena "Hola mundo" y algunos caracteres especiales, mostrados encima de un rectángulo coloreado. Después de crearlo, guardaremos el dibujo en un archivo PDF independiente.

```
desde reportlab.lib importar colores desde
reportlab.graphics.shapes importar *
```

```
d = Dibujo(400, 200)
d.add(Rect(50, 50, 300, 100, fillColor=colors.amarillo)) d.add(String(150,100, 'Hola
mundo', fontSize=18, fillColor=colors. rojo)) d.add(String(180,86, 'Caracteres especiales \
\ xc2\xc2\xc2\xae\xc2\xc2\xc1\xce\xb1\xce\xb2', fillColor=colores.rojo))

de reportlab.graphics importar renderPDF renderPDF.drawToFile(d,
'ejemplo1.pdf', 'Mi primer dibujo')
```

Esto producirá un archivo PDF que contiene el siguiente gráfico:



Figura 11-1: 'Hola mundo'

Cada renderizador puede hacer lo que sea apropiado para su formato y puede tener cualquier API necesaria. Si se refiere a un formato de archivo, normalmente tiene una función drawToFile, y eso es todo lo que necesitas saber sobre el renderizador. Guardemos el mismo dibujo en formato Postscript encapsulado:

```
de reportlab.graphics importar renderPS renderPS.drawToFile(d,
'ejemplo1.eps')
```

Esto producirá un archivo EPS con el dibujo idéntico, que se puede importar a herramientas de publicación como Quark Express. Si quisieramos generar el mismo dibujo como un archivo de mapa de bits para un sitio web, digamos, todo lo que tenemos que hacer es escribir un código como este:

```
de reportlab.graphics importar renderPM renderPM.drawToFile(d,
'ejemplo1.png', 'PNG')
```

Muchos otros formatos de mapas de bits, como GIF, JPG, TIFF, BMP y PPN, están realmente disponibles, por lo que es poco probable que necesites agregar pasos de posprocesamiento externos para convertir al formato final que necesitas.

Para producir un archivo SVG que contenga el dibujo idéntico, que pueda importarse a herramientas de edición gráfica como Illustrator, todo lo que necesitamos hacer es escribir un código como este:

```
desde reportlab.graphics importar renderSVG
renderSVG.drawToFile(d, 'ejemplo1.svg')
```

Verificación de atributos

Python es muy dinámico y nos permite ejecutar declaraciones en tiempo de ejecución que fácilmente pueden ser la fuente de comportamientos inesperados. Un 'error' sutil se produce al asignar un atributo que el marco no conoce porque el nombre del atributo utilizado contiene un error tipográfico. Python le permite salirse con la suya (agregando un nuevo atributo a un objeto, por ejemplo), pero el marco de gráficos no detectará este 'error tipográfico' sin tomar contramedidas especiales.

Existen dos técnicas de verificación para evitar esta situación. El valor predeterminado es que cada objeto verifique cada asignación en tiempo de ejecución, de modo que solo pueda asignar atributos "legales". Esto es lo que sucede por defecto. Como

esto impone una pequeña penalización de rendimiento; este comportamiento se puede desactivar cuando sea necesario.

```
>>> r = Rect(10,10,200,100, fillColor=colors.red) >>> r.fullColor = Colors.green # tenga en cuenta el error
tipográfico >>> rx = 'no es un número' >>> del r. ancho
# tipo de argumento ilegal # que
debería confundirlo
```

Estas declaraciones serían capturadas por el compilador en un lenguaje escrito estáticamente, pero Python te permite salirte con la tuya. El primer error podría dejarte mirando la imagen tratando de descubrir por qué los colores estaban mal. El segundo error probablemente se aclara sólo más tarde, cuando algún servidor intente dibujar el rectángulo. El tercero, aunque menos probable, resulta en un objeto inválido que no sabría dibujarse a sí mismo.

```
>>> r = formas.Rect(10,10,200,80) >>> r.fullColor =
colores.verde Rastreo (última llamada más
reciente):
Archivo "<entrada interactiva>", línea 1, en ?
Archivo "C:\code\users\landy\graphics\shapes.py", línea 254, en __setattr__ validarSetattr(self,attr,value) #from
reportlab.lib.attrmap Archivo "C:\code\users\landy\lib \attrmap.py", línea 74, en validarSetattr levanta
AttributeError, "Atributo ilegal '%s' en la clase %s" %(nombre, obj.__clase__.nombre__)

AttributeError: atributo ilegal 'fullColor' en la clase Rect
>>>
```

Esto impone una penalización de rendimiento, por lo que este comportamiento se puede desactivar cuando sea necesario. Para hacer esto, debe usar las siguientes líneas de código antes de importar reportlab.graphics.shapes por primera vez:

```
>>> importar reportlab.rl_config >>> reportlab.rl_config.shapeChecking
= 0 >>> desde reportlab.graphics importar formas >>>
```

Una vez que desactivas ShapeChecking, las clases en realidad se construyen sin el gancho de verificación; Entonces el código debería volverse más rápido. Actualmente, la penalización parece ser de alrededor del 25% en lotes de gráficos, por lo que no vale la pena desactivarla. Sin embargo, si trasladamos los renderizadores a C en el futuro (lo cual es muy posible), el 75% restante se reduciría a casi nada y el ahorro proveniente de la verificación sería significativo.

Cada objeto, incluido el dibujo mismo, tiene un método verificar(). Esto tiene éxito o genera una excepción. Si desactiva la verificación automática, entonces debería llamar explícitamente a verificar() en las pruebas al desarrollar el código, o quizás una vez en un proceso por lotes.

Edición de propiedades

Una piedra angular del reportlab/gráficos que cubriremos a continuación es que puede documentar automáticamente los widgets. Esto significa hacerse con todas sus propiedades editables, incluidas las de sus subcomponentes.

Otro objetivo es poder crear GUI y archivos de configuración para dibujos. Se puede crear una GUI genérica para mostrar todas las propiedades editables de un dibujo y permitirle modificarlas y ver los resultados. El entorno de desarrollo Visual Basic o Delphi son buenos ejemplos de este tipo de cosas. En una aplicación de gráficos por lotes, un archivo podría enumerar todas las propiedades de todos los componentes de un gráfico y fusionarse con una consulta de base de datos para crear un lote de gráficos.

Para soportar estas aplicaciones tenemos dos interfaces, getProperties y setProperties, así como un método conveniente dumpProperties. El primero devuelve un diccionario de las propiedades editables de un objeto; el segundo los pone en masa. Si un objeto tiene 'niños' expuestos públicamente, entonces también se pueden establecer y obtener sus propiedades de forma recursiva. Esto tendrá mucho más sentido cuando veamos los widgets más adelante, pero necesitamos poner el soporte en la base del marco.

```
>>> r = formas.Rect(0,0,200,100) >>> importar
pprint >>>
pprint pprint(r.getProperties()) { 'fillColor':
Color(0.00,0.00,0.00),
'altura': 100, 'rx': 0,
'ry': 0,
'strokeColor':
Color(0.00,0.00,0.00), 'strokeDashArray': Ninguno,
```

```
'strokeLineCap': 0, 'strokeLineJoin':
0, 'strokeMiterLimit': 0, 'strokeWidth':
1, 'ancho': 200, 'x': 0, 'y': 0}

>>> r.setProperties({'x':20, 'y':30, 'strokeColor': colores.rojo}) >>> r.dumpProperties() fillColor = Color(0.00,0.00,0.00) altura
= 100 rx = 0 ry = 0 trazoColor =
Color(1.00,0.00,0.00) trazoDashArray = Ninguno trazoLineCap
= 0 trazoLineJoin = 0

trazoMiterLimit = 0 trazoAncho = 1

ancho = 200 x = 20
y = 30

>>>
```

Nota: pprint es el módulo de biblioteca estándar de Python que le permite 'imprimir de forma bonita' la salida en varias líneas en lugar de tener una línea muy larga.

Estos tres métodos no parecen hacer mucho aquí, pero como veremos, hacen que nuestro marco de widgets sea mucho más poderoso cuando se trata de objetos no primitivos.

Nombrar a los niños

Puede agregar objetos al dibujo y al grupo de objetos. Estos normalmente van en una lista de contenidos. Sin embargo, también puedes darle un nombre a los objetos al agregarlos. Esto le permite hacer referencia y posiblemente cambiar cualquier elemento de un dibujo después de construirlo.

```
>>> d = formas.Drawing(400, 200) >>> s = formas.String(10,
10, 'Hola mundo') >>> d.add(s, 'caption') >>> s.caption .texto 'Hola mundo'
```

```
>>>
```

Tenga en cuenta que puede utilizar la misma instancia de forma en varios contextos de un dibujo; Si elige utilizar el mismo objeto Círculo en muchas ubicaciones (por ejemplo, un diagrama de dispersión) y utiliza nombres diferentes para acceder a él, seguirá siendo un objeto compartido y los cambios serán globales.

Esto proporciona un paradigma para crear y modificar dibujos interactivos.

11.3 Gráficos

La motivación para gran parte de esto es crear un paquete de gráficos flexible. Esta sección presenta un tratamiento de las ideas detrás de nuestro modelo de gráficos, cuáles son los objetivos del diseño y qué componentes del paquete de gráficos ya existen.

Objetivos de diseño

Estos son algunos de los objetivos de diseño:

Haga que el uso simple de alto nivel sea realmente simple

Debería ser posible crear un gráfico simple con un mínimo de líneas de código y, al mismo tiempo, hacer que "haga lo correcto" con configuraciones automáticas sensatas. Los fragmentos del gráfico circular de arriba hacen esto. Si un gráfico real tiene muchos subcomponentes, aún así no debería necesitar interactuar con ellos a menos que desee personalizar lo que hacen.

Permitir un posicionamiento preciso

Un requisito absoluto en edición y diseño gráfico es controlar la ubicación y el estilo de cada elemento. Intentaremos tener propiedades que especifiquen cosas en tamaños y proporciones fijas del dibujo, en lugar de tener un cambio de tamaño automático. Por lo tanto, el 'rectángulo de trazado interior' no cambiará mágicamente cuando aumente el tamaño de fuente de las etiquetas y, incluso si esto significa que sus etiquetas pueden salirse del borde izquierdo del rectángulo del gráfico. Es su trabajo obtener una vista previa del gráfico y elegir los tamaños y espacios que funcionarán.

Algunas cosas deben ser automáticas. Por ejemplo, si desea encajar N barras en un espacio de 200 puntos y no conoce N de antemano, especificamos la separación de barras como un porcentaje del ancho de una barra en lugar de un tamaño en puntos, y dejamos que el gráfico lo resuelva. . Esto sigue siendo determinista y controlable.

Controle los elementos secundarios individualmente o en grupo

Usamos clases de colección inteligentes que te permiten personalizar un grupo de cosas, o solo una de ellas. Por ejemplo, puedes hacer esto en nuestro gráfico circular experimental:

```
d = Dibujo(400,200) pc = Pie()
pc.x = 150
pc.y = 50
pc.data =
[10,20,30,40,50,60] pc.labels =
['a','b','c','d','e','f'] pc.slices.strokeWidth=0.5
pc.slices[3].popout = 20
pc.slices[3].strokeWidth = 2
pc.slices[3].strokeDashArray = [2,2]
pc.slices[3].labelRadius = 1.75 pc.slices[3].fontColor =
colores.red d.add(pc, "")
```

pc.slices[3] en realidad crea perezosamente un pequeño objeto que contiene información sobre el segmento en cuestión; esto se utilizará para formatear un cuarto segmento en el momento del sorteo, si lo hay.

Solo expone las cosas que deberías cambiar.

Desde un punto de vista estadístico, sería incorrecto permitir ajustar directamente el ángulo de una de las porciones del pastel en el ejemplo anterior, ya que eso está determinado por los datos. Entonces no todo quedará expuesto a través de las propiedades públicas. Puede haber 'puertas traseras' que le permitan violar esto cuando realmente lo necesite, o métodos para proporcionar funcionalidad avanzada, pero en general las propiedades serán ortogonales.

Composición y componentes basados.

Los gráficos se crean a partir de widgets secundarios reutilizables. Una leyenda es un ejemplo fácil de entender. Si necesita un tipo especializado de leyenda (por ejemplo, muestras de color circulares), debe subclasicar el widget Leyenda estándar. Entonces podrías hacer algo como...

```
c = MyChartWithLegend() c.legend
= MyNewLegendClass() # simplemente cámbialo c.legend.swatchRadius
= 5 # establece una propiedad solo relevante para la nueva c.data = [10,20,30] # y luego configura como de
costumbre ...
```

...o cree/modifique su propio gráfico o clase de dibujo que crea uno de estos de forma predeterminada. Esto también es muy relevante para los gráficos de series temporales, donde puede haber muchos estilos de eje x.

Las clases de gráficos de nivel superior crearán una serie de dichos componentes y luego llamarán a métodos o establecerán propiedades privadas para indicarles su altura y posición: todo lo que debe hacerse por usted y que no puede personalizar. Estamos trabajando en modelar cuáles deberían ser los componentes y publicaremos sus API aquí a medida que surja un consenso.

Múltiples

Un corolario del enfoque de componentes es que se pueden crear diagramas con varios cuadros o gráficos de datos personalizados. Nuestro ejemplo favorito de lo que buscamos es el informe meteorológico de nuestra galería aportado por un usuario; Nos gustaría facilitar la creación de dichos dibujos, conectar los bloques de construcción a sus leyendas y alimentar esos datos de manera consistente.

(Si quieres ver la imagen, está disponible en nuestra web [aquí](#))

Descripción general

Un gráfico o trazado es un objeto que se coloca en un dibujo; no es en sí mismo un dibujo. De este modo podrás controlar dónde va, poner varios en un mismo dibujo o añadir anotaciones.

Los gráficos tienen dos ejes; Los ejes pueden ser ejes de valor o categoría. Los ejes a su vez tienen una propiedad Etiquetas que le permite configurar todas las etiquetas de texto o cada una individualmente. La mayoría de los detalles de configuración que varían de un gráfico a otro se relacionan con las propiedades de los ejes o las etiquetas de los ejes.

Los objetos exponen propiedades a través de las interfaces analizadas en la sección anterior; Todos estos son opcionales y están ahí para permitir que el usuario final configure la apariencia. Las cosas que deben configurarse para que un gráfico funcione y la comunicación esencial entre un gráfico y sus componentes se manejan mediante métodos.

Puede subclasicar cualquier componente del gráfico y utilizar su reemplazo en lugar del original siempre que implemente los métodos y propiedades esenciales.

11.4 Etiquetas

Una etiqueta es una cadena de texto adjunta a algún elemento del gráfico. Se utilizan en ejes, para títulos o junto a ejes, o se adjuntan a puntos de datos individuales. Las etiquetas pueden contener caracteres de nueva línea, pero solo una fuente.

El texto y el "origen" de una etiqueta normalmente los establece su objeto principal. Se accede a ellos mediante métodos en lugar de propiedades. Por lo tanto, el eje X decide el 'punto de referencia' para cada etiqueta de marca de verificación y el texto numérico o de fecha para cada etiqueta. Sin embargo, el usuario final puede establecer propiedades de la etiqueta (o colección de etiquetas) directamente para afectar su posición relativa a este origen y todo su formato.

```
desde reportlab.graphics importar formas desde
reportlab.graphics.charts.textlabels importar etiqueta

d = Dibujo(200, 100)

# marcar el origen de la etiqueta d.add(Circle(100,90, 5,
fillColor=colors.green))

lab = Label()
lab.setOrigin(100,90) lab.boxAnchor =
'nw' lab.angle = 45 lab.dx = 0 lab.dy =
-20 lab.boxStrokeColor =
colores.verde
lab.setText('Some Multi
-Etiqueta de línea')

d.add(laboratorio)
```



Figura 11-2: Ejemplo de etiqueta

En el dibujo de arriba, la etiqueta se define en relación con la mancha verde. El cuadro de texto debe tener su esquina noreste diez puntos hacia abajo desde el origen y estar girado 45 grados alrededor de esa esquina.

Actualmente las etiquetas tienen las siguientes propiedades, que creemos que son suficientes para todos los gráficos que hemos visto hasta la fecha:

Propiedad	Significado
-----------	-------------

dx	El desplazamiento x de la etiqueta.
tú	El desplazamiento y de la etiqueta.
ángulo	El ángulo de rotación (en sentido antihorario) aplicado a la etiqueta.
cajaancla	El ancla del cuadro de la etiqueta, uno de 'n', 'e', 'w', 's', 'ne', 'nw', 'se', 'sw'.
textoAncla	El lugar donde anclar el texto de la etiqueta, uno de 'inicio', 'medio', 'fin'.
cajaRellenoColor	El color de relleno utilizado en el cuadro de la etiqueta.
cajaTrazoColor	El color del trazo utilizado en el cuadro de la etiqueta.
cajaTrazoAncho	El ancho de línea del cuadro de la etiqueta.
nombre de la fuente	El nombre de la fuente de la etiqueta.
tamaño de fuente	El tamaño de fuente de la etiqueta.
principal	El valor principal de las líneas de texto de la etiqueta.
x	La coordenada X del punto de referencia.
y	La coordenada Y del punto de referencia.
ancho	El ancho de la etiqueta.
altura	La altura de la etiqueta.

Tabla 11-4: Propiedades de la etiqueta

Para ver muchos más ejemplos de objetos Label con diferentes combinaciones de propiedades, eche un vistazo en el conjunto de pruebas de ReportLab en la carpeta pruebas, ejecute el script test_charts_textlabels.py y ¡Mira el documento PDF que genera!

11.5 ejes

Identificamos dos tipos básicos de ejes: los de Valor y los de Categoría . Ambos vienen en versiones horizontales y verticales. Ambos pueden subclasicarse para crear tipos de ejes muy específicos. Por ejemplo, si tiene reglas complejas para las cuales fechas para mostrar en una aplicación de serie temporal, o desea una escala irregular, anula el eje y crea una nueva uno.

Los ejes son responsables de determinar el mapeo de las coordenadas de datos a imágenes; transformar puntos a petición del mapa; dibujándose a sí mismos y sus marcas, líneas de cuadrícula y etiquetas de ejes.

Este dibujo muestra dos ejes, uno de cada tipo, que han sido creados directamente sin referencia a ningún cuadro:

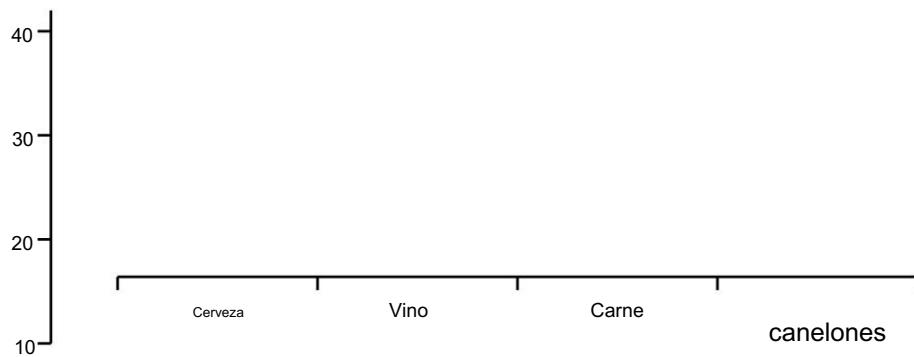


Figura 11-3: Dos ejes aislados

Aquí está el código que los creó:

```

desde reportlab.graphics importar formas desde
reportlab.graphics.charts.axes importar XCategoryAxis,YValueAxis

dibujo = Dibujo(400, 200)

datos = [(10, 20, 30, 40), (15, 22, 37, 42)]

xAxis = XCategoryAxis() xAxis.setPosition(75,
75, 300) xAxis.configure(datos) xAxis.categoryNames =
['Cerveza', 'Vino', 'Carne', 'Canelones']
xAxis.labels.boxAnchor = 'n' xAxis.labels[3].dy = -15 xAxis.labels[3].angle = 30 xAxis.labels[3].fontName = 'Times-Bold'

Eje y = EjeYValue() Eje y.setPosition(50,
50, 125) Eje y.configure(datos)

dibujo.añadir(ejex) dibujo.añadir(ejey)

```

Recuerda que, normalmente, no tendrás que crear ejes directamente; cuando se utiliza un gráfico estándar, viene con ejes ya preparados. Los métodos son los que utiliza el gráfico para configurarlo y cuidar la geometría. Sin embargo, hablaremos de ellos en detalle a continuación. Los ejes ortogonalmente duales a los que describimos tienen esencialmente las mismas propiedades, excepto las que se refieren a los ticks.

Clase XCategoryAxis

Un eje de categorías en realidad no tiene una escala; simplemente se divide en cubos del mismo tamaño. Es más simple que un eje de valores. El gráfico (o programador) establece su ubicación con el método `setPosition(x, y, length)`.

El siguiente paso es mostrarle los datos para que pueda configurarse. Esto es fácil para un eje de categorías: simplemente cuenta el número de puntos de datos en una de las series de datos. El atributo `invertido` (si es 1) indica que las categorías deben invertirse. Cuando se dibuja el dibujo, el eje puede proporcionar cierta ayuda al gráfico con su método `scale()`, que le indica al gráfico dónde comienza y termina una categoría determinada en la página. Todavía no hemos visto la necesidad de permitir que las personas anulen los anchos o posiciones de las categorías.

Un `XCategoryAxis` tiene las siguientes propiedades editables:

Propiedad	Significado
-----------	-------------

visible	¿Debería dibujarse el eje? A veces no desea mostrar uno o ambos ejes, pero aún así deben estar allí mientras administran la escala de puntos.
color del trazo	Color del eje
trazoDashArray	Si se debe dibujar el eje con un guión y, de ser así, de qué tipo. El valor predeterminado es Ninguno
anchura del trazo	Ancho del eje en puntos
marcar arriba	¿A qué distancia por encima del eje deben sobresalir las marcas? (Tenga en cuenta que hacer esto igual a la altura del gráfico le dará una línea de cuadrícula)
tickDown	¿A qué distancia por debajo del eje debe sobresalir la marca de verificación?
Nombres de categoría	Ninguno o una lista de cadenas. Debe tener la misma longitud que cada serie de datos.
etiquetas	Una colección de etiquetas para las marcas. De forma predeterminada, el 'norte' de cada etiqueta de texto (es decir, el centro superior) se coloca 5 puntos hacia abajo desde el centro de cada categoría en el eje. Puede redefinir cualquier propiedad de todo el grupo de etiquetas o de cualquier etiqueta. Si categoríaNames=Ninguno, no se dibujan etiquetas.
título	Aun no implementado. Esto debe ser como una etiqueta, pero también te permite configurar el texto directamente. Tendría una ubicación predeterminada debajo del eje.

Tabla 11-5: propiedades de XCategoryAxis

Eje de valor Y

El eje izquierdo del diagrama es YValueAxis. Un eje de valores se diferencia de un eje de categorías en que cada punto a lo largo de su longitud corresponde a un valor en el espacio del gráfico. Es trabajo del eje configurarse y convertir los valores Y del espacio del gráfico en puntos según demanda para ayudar al gráfico principal en el trazado.

setPosition(x, y, length) y configure(data) funcionan exactamente como para un eje de categorías. Si no ha especificado completamente el máximo, el mínimo y el intervalo de tick, entonces configure() da como resultado que el eje elija los valores adecuados. Una vez configurado, el eje de valores puede convertir los valores de datos y al espacio de dibujo con el método scale(). De este modo:

```
>>> Eje y = EjeYValue()
>>> Eje y.setPosition(50, 50, 125)
>>> datos = [(10, 20, 30, 40)(15, 22, 37, 42)]
>>> yAxis.configure(datos)
>>> yAxis.scale(10) # debe estar en la parte inferior
del gráfico 50.0
>>> yAxis.scale(40) # debe
estar cerca de la parte superior 167.1875
```

>>>

De forma predeterminada, el punto de datos más alto está alineado con la parte superior del eje, el más bajo con la parte inferior del eje y el eje elige 'bonitos números redondos' para sus puntos de marca. Puede anular estas configuraciones con las propiedades siguientes.

Propiedad	Significado
visible	¿Debería dibujarse el eje? A veces no desea mostrar uno o ambos ejes, pero aún así deben estar allí mientras administran la escala de puntos.
color del trazo	Color del eje
trazoDashArray	Si se debe dibujar el eje con un guión y, de ser así, de qué tipo. El valor predeterminado es Ninguno

anchura del trazo	Ancho del eje en puntos
tickIzquierda	¿A qué distancia a la izquierda del eje deben sobresalir las marcas? (Tenga en cuenta que hacer esto igual a la altura del gráfico le dará una línea de cuadrícula)
tickDerecha	¿Qué tan lejos a la derecha del eje debe sobresalir la marca de verificación?
valor mínimo	El valor de y al que debe corresponder la parte inferior del eje. El valor predeterminado es Ninguno, en cuyo caso el eje lo establece en el punto de datos real más bajo (por ejemplo, 10 en el ejemplo anterior). Es común establecerlo en cero para evitar engañar a la vista.
valorMax	El valor de y al que debe corresponder la parte superior del eje. El valor predeterminado es Ninguno, en cuyo caso el eje lo establece en el punto de datos real más alto (por ejemplo, 42 en el ejemplo anterior). Es común establecer esto en un "número redondo" para que las barras de datos no lleguen a la parte superior.
valorPaso	La y cambia entre intervalos de ticks. De forma predeterminada, esto es Ninguno y el gráfico intenta seleccionar "números redondos agradables" que sean ligeramente más anchos que el espacio mínimo de TickSpacing que aparece a continuación.
valorPasos	Una lista de números en los que colocar ticks.
espaciado mínimo de garrapatas	Esto se utiliza cuando valueStep se establece en Ninguno y se ignora en caso contrario. El diseñador especificó que las marcas de graduación no deberían estar a menos de X puntos de distancia (basado, presumiblemente, en consideraciones sobre el tamaño y el ángulo de la fuente de la etiqueta). El gráfico prueba valores del tipo 1,2,5,10,20,50,100... (bajando por debajo de 1 si es necesario) hasta que encuentra un intervalo que es mayor que el espaciado deseado y lo usa para el paso.
formato de texto de etiqueta	Esto determina lo que va en las etiquetas. A diferencia de un eje de categorías que acepta cadenas fijas, se supone que las etiquetas en ValueAxis son números. Puede proporcionar una 'cadena de formato' como '%0.2f' (muestra dos decimales) o una función arbitraria que acepte un número y devuelva una cadena. Un uso de este último es convertir una marca de tiempo a un formato legible de año, mes y día.
título	Aún no implementado. Esto debe ser como una etiqueta, pero también te permite configurar el texto directamente. Tendría una ubicación predeterminada debajo del eje.

Tabla 11-6: Propiedades del eje YValue

La propiedad valueSteps le permite especificar explícitamente las ubicaciones de las marcas de verificación, por lo que no tiene que seguir intervalos regulares. Por lo tanto, puede trazar fines de mes y fechas de fin de mes con un par de funciones auxiliares y sin necesidad de clases de gráficos de series temporales especiales. El siguiente código muestra cómo crear un XValueAxis simple con intervalos de tick especiales. ¡Asegúrese de configurar el atributo valueSteps antes de llamar al método de configuración!

```

desde reportlab.graphics.shapes importar dibujo desde
reportlab.graphics.charts.axes importar XValueAxis

dibujo = Dibujo(400, 100)

datos = [(10, 20, 30, 40)]

xAxis = XValueAxis()
xAxis.setPosition(75, 50, 300) xAxis.valueSteps = [10, 15,
20, 30, 35, 40] xAxis.configure(data) xAxis.labels.boxAnchor = 'n'

dibujo.add(xEje)

```

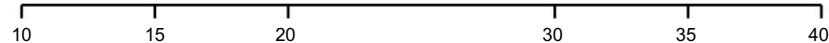


Figura 11-4: Un eje con marcas no equidistantes

Además de estas propiedades, todas las clases de ejes tienen tres propiedades que describen cómo unir dos de ellos para entre sí. Nuevamente, esto es interesante sólo si define sus propios gráficos o desea modificar la apariencia de un gráfico existente que utilice dichos ejes. Estas propiedades se enumeran aquí sólo de forma muy breve por ahora, pero puede encontrarlas una gran cantidad de funciones de muestra en el módulo reportlab/graphics/axes.py que puede examinar...

Un eje se une a otro, llamando al método `joinToAxis(otherAxis, mode, pos)` en el primer eje, siendo `mode` y `pos` las propiedades descritas por `joinAxisMode` y `joinAxisPos`, respectivamente. 'puntos' significa usar un valor absoluto y 'valor' usar un valor relativo (ambos indicados por la propiedad `joinAxisPos`) a lo largo del eje.

Propiedad	Significado
unirse al eje	Une ambos ejes si es cierto.
unirse al modo eje	Modo utilizado para conectar ejes ('inferior', 'superior', 'izquierda', 'derecha', 'valor', 'puntos', Ninguno).
unirseAxisPos	Posición en la que unirse con otro eje.

Tabla 11-7 - Propiedades de unión de ejes

11.6 Gráficos de barras

Esto describe nuestra clase `VerticalBarChart` actual, que utiliza los ejes y etiquetas anteriores. pensamos que es paso en la dirección correcta, pero está lejos de ser definitivo. Tenga en cuenta que las personas con las que hablamos están divididas aproximadamente 50/50 en si llamar a esto un gráfico de barras "vertical" u "horizontal". Elegimos este nombre porque aparece 'Vertical' junto a 'Barra', por lo que entendemos que las barras, en lugar del eje de categorías, son verticales.

Como es habitual, comenzaremos con un ejemplo:

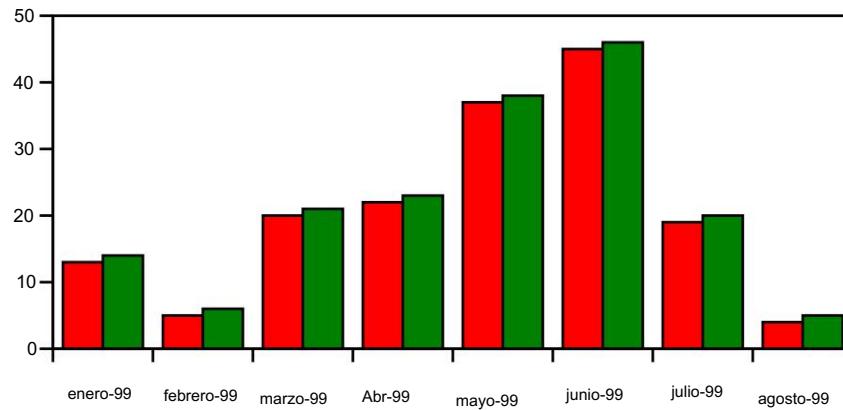


Figura 11-5: Gráfico de barras simple con dos series de datos

```
# código para producir el gráfico anterior

de reportlab.graphics.shapes importar Dibujo de reportlab.graphics.charts.barcharts
importar VerticalBarChart

dibujo = Dibujo(400, 200)

datos = [(13, 5,
          20, 22, 37, 45, 19, 4), (14, 6, 21, 23, 38, 46, 20, 5)]


bc = VerticalBarChart() bc.x = 50 bc.y = 50
bc.height = 125
bc.width = 300
bc.data = datos bc.strokeColor
= colores.negro

bc.valueAxis.valueMin = 0 bc.valueAxis.valueMax
= 50 bc.valueAxis.valueStep = 10

bc.categoryAxis.labels.boxAnchor = 'ne' bc.categoryAxis.labels.dx = 8
bc.categoryAxis.labels.dy = -2 bc.categoryAxis.labels.angle
= 30 bc.categoryAxis.categoryNames = ['Jan-99'
,'febrero-99','marzo-99', 'abril-99', 'mayo-99', 'junio-99', 'julio-99',
'agosto-99']

dibujo.add(bc)
```

La mayor parte de este código se ocupa de la configuración de los ejes y las etiquetas, que ya hemos cubierto. Estas son las propiedades de nivel superior de la clase `VerticalBarChart`:

Propiedad	Significado
datos	Debería ser una "lista de listas de números" o una "lista de tuplas de números". Si tiene solo una serie, escribala como <code>datos = [(10,20,30,42),]</code>
x, y, ancho, alto	Estos definen el 'rectángulo de trazado' interior. Resaltamos esto con un borde amarillo arriba. Tenga en cuenta que es su trabajo colocar el gráfico en el dibujo de manera que deje espacio para todas las etiquetas y marcas de los ejes. Especificamos este 'rectángulo interior' porque hace que sea muy fácil diseñar varios gráficos de manera coherente.
color del trazo	El valor predeterminado es Ninguno. Esto dibujará un borde alrededor del rectángulo del trazado, lo que puede resultar útil en la depuración. Los ejes sobrescribirán esto.
color de relleno	El valor predeterminado es Ninguno. Esto llenará el rectángulo del trazado con un color sólido. (Tenga en cuenta que podríamos implementar <code>dashArray</code> , etc. como para cualquier otra forma sólida)
usoAbsoluto	El valor predeterminado es 0. Si es 1, las tres propiedades siguientes son valores absolutos en puntos (lo que significa que puede crear un gráfico donde las barras sobresalgan del rectángulo del trazado); si 0, son cantidades relativas e indican los anchos proporcionales de los elementos involucrados.
ancho de barra	Como dice. El valor predeterminado es 10.
espaciado de grupo	El valor predeterminado es 5. Este es el espacio entre cada grupo de barras. Si solo tiene una serie, use <code>groupSpacing</code> y no <code>barSpacing</code> para dividirlas. La mitad del espacio del grupo se utiliza antes de la primera barra del gráfico y la otra mitad al final.

barraEspaciado	El valor predeterminado es 0. Este es el espacio entre barras en cada grupo. Si quisiera un pequeño espacio entre las barras verdes y rojas en el ejemplo anterior, haría que fuera distinto de cero.
barraEtiquetaFormato	El valor predeterminado es Ninguno. Al igual que con YValueAxis, si proporciona una función o una cadena de formato, se dibujarán etiquetas junto a cada barra que muestra el valor numérico. Se colocan automáticamente encima de la barra para los valores positivos y debajo para los negativos.
barraEtiquetas	Una colección de etiquetas utilizadas para dar formato a todas las etiquetas de barras. Dado que se trata de una matriz bidimensional, puede formatear explícitamente la tercera etiqueta de la segunda serie usando esta sintaxis: chart.barLabels[(1,2)].fontSize = 12
valorEje	El eje de valores, que puede formatearse como se describió anteriormente.
categoriaEje	El eje de categorías, que puede tener el formato descrito anteriormente.
título	Aun no implementado. Esto debe ser como una etiqueta, pero también te permite configurar el texto directamente. Tendría una ubicación predeterminada debajo del eje.

Tabla 11-8: Propiedades de VerticalBarChart

De esta tabla deducimos que agregar las siguientes líneas a nuestro código anterior debería duplicar el espacio entre grupos de barras (el atributo groupSpacing tiene un valor por defecto de cinco puntos) y también deberíamos ver un pequeño espacio entre barras del mismo grupo (barSpacing) .

```
bc.groupSpacing = 10 bc.barSpacing  
= 2,5
```

Y, de hecho, esto es exactamente lo que podemos ver después de agregar estas líneas al código anterior. Observe cómo también ha cambiado el ancho de las barras individuales. Esto se debe a que el espacio agregado entre las barras debe "quitarse" de algún lugar ya que el ancho total del gráfico permanece sin cambios.

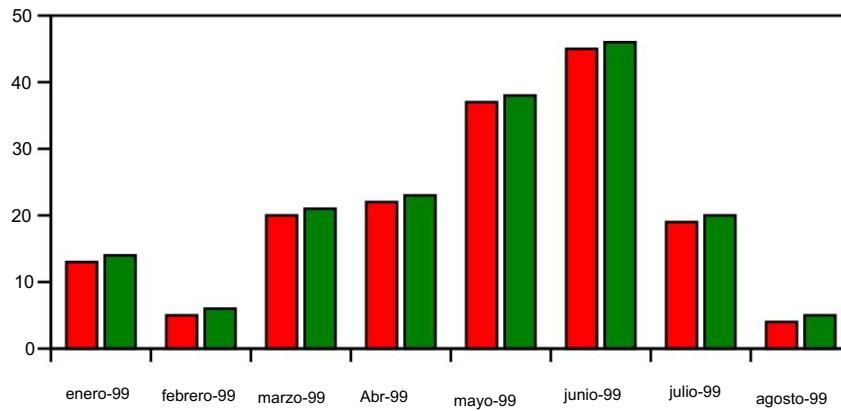


Figura 11-6: Como antes, pero con espaciado modificado

Las etiquetas de las barras se muestran automáticamente para los valores negativos debajo del extremo inferior de la barra y para los valores positivos encima del extremo superior de las demás.

Las barras apiladas también son compatibles con gráficos de barras verticales. Habilite este diseño para su gráfico configurando el atributo de estilo en "apilado" en el eje de categorías.

```
bc.categoryAxis.style = 'apilado'
```

A continuación se muestra un ejemplo de los valores del gráfico anterior organizados en estilo apilado.

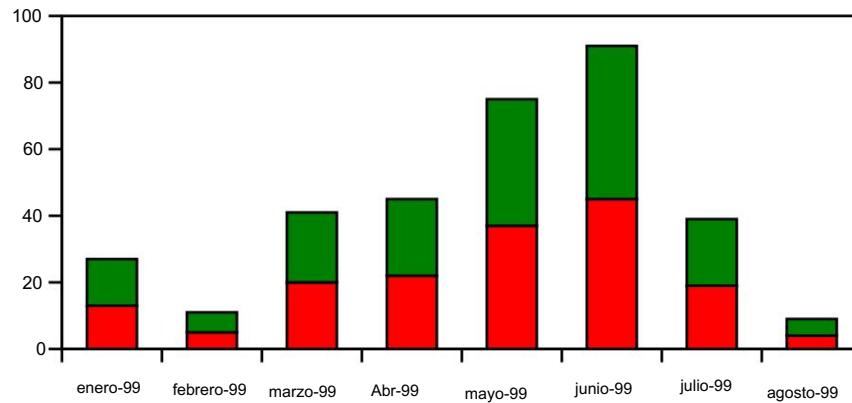


Figura 11-7: Apilamiento de barras una encima de la otra.

11.7 Gráficos de líneas

Consideramos que los "Gráficos de líneas" son esencialmente lo mismo que los "Gráficos de barras", pero con líneas en lugar de barras. Ambos comparten el mismo par de ejes Categoría/Valor. Esto contrasta con los "gráficos de líneas", donde ambos ejes son ejes de valor .

El siguiente código y su resultado servirán como un ejemplo sencillo. Seguirán más explicaciones. Por el momento, también puede estudiar el resultado de ejecutar la herramienta reportlab/lib/graphdocpy.py sin ningún argumento y buscar en el documento PDF generado ejemplos de gráficos de líneas.

```
desde reportlab.graphics.charts.linecharts importar HorizontalLineChart

dibujo = Dibujo(400, 200)

datos = [ (13,
           5, 20, 22, 37, 45, 19, 4), (5, 20, 46, 38, 23, 21, 6, 14)

        ]

lc = HorizontalLineChart() lc.x = 50 lc.y = 50
lc.height = 125
lc.width = 300
lc.data = datos lc.joinedLines
= 1 catNames = 'Ene Feb
Mar Abr May Jun Jul
Ago'.split(' ')
lc.categoryAxis.categoryNames = catNames lc.categoryAxis.labels.boxAnchor = 'n' lc.valueAxis.valueMin =
0 lc.valueAxis.valueMax = 60 lc.valueAxis.valueStep = 15 lc.lines[0].strokeWidth
= 2 lc.lines[1].strokeWidth = 1.5 dibujo.add(lc)
```

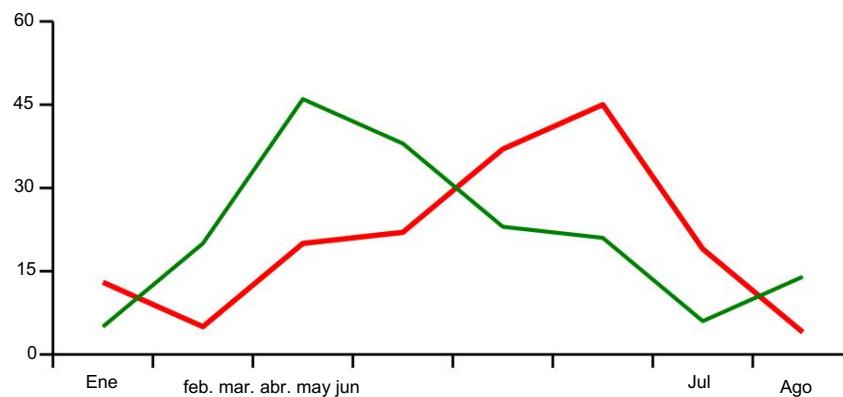


Figura 11-8: Ejemplo de gráfico de líneas horizontales

Propiedad	Significado
datos	Datos a trazar, lista de (listas de) números.
x, y, ancho, alto	Cuadro delimitador del gráfico de líneas. Tenga en cuenta que xey NO especifican el centro sino la esquina inferior izquierda
valorEje	El eje de valores, que puede formatearse como se describió anteriormente.
categoriaEje	El eje de categorías, que puede tener el formato descrito anteriormente.
color del trazo	El valor predeterminado es Ninguno. Esto dibujará un borde alrededor del rectángulo del trazado, lo que puede resultar útil en la depuración. Los ejes sobreescibirán esto.
color de relleno	El valor predeterminado es Ninguno. Esto llenará el rectángulo del trazado con un color sólido.
líneas.trazoColor	Color de la línea.
líneas.strokeWidth	Ancho de la línea.
líneaEtiquetas	Una colección de etiquetas utilizadas para formatear todas las etiquetas de línea. Dado que se trata de una matriz bidimensional, puede formatear explícitamente la tercera etiqueta de la segunda línea usando esta sintaxis: chart.lineLabels[[1,2]].fontSize = 12
formato de etiqueta de línea	El valor predeterminado es Ninguno. Al igual que con YValueAxis, si proporciona una función o una cadena de formato, se dibujarán etiquetas junto a cada línea que muestre el valor numérico. También puede configurarlo en 'valores' para mostrar los valores definidos explicitamente en lineLabelArray.
líneaEtiquetaArray	Matriz explícita de valores de etiquetas de línea; debe coincidir con el tamaño de los datos, si están presentes. Los valores de estas etiquetas se mostrarán solo si la propiedad lineLabelFormat anterior está configurada en 'valores'.

Tabla 11-9: Propiedades de HorizontalLineChart

11.8 Gráficos lineales

A continuación mostramos un ejemplo más complejo de un gráfico de líneas que también utiliza algunas características experimentales, como marcadores de líneas colocados en cada punto de datos.

```
desde reportlab.graphics.charts.lineplots importar LinePlot desde reportlab.graphics.widgets.markers importar
makeMarker
```

```

dibujo = Dibujo(400, 200)

datos = [ ((1,1),
           (2,2), (2.5,1), (3,3), (4,5)), ((1,2), (2,3), (2.5 ,2), (3.5,5), (4,6))

]

lp = LinePlot() lp.x = 50 lp.y
= 50 lp.height =
125 lp.width =
300 lp.data = datos
lp.joinedLines = 1
lp.lines[0].symbol =
makeMarker('FilledCircle')
lp.lines[1].symbol = makeMarker('Circle') lp.lineLabelFormat = '%2.0f' lp.strokeColor =
colores.negro lp.xValueAxis.valueMin = 0 lp.xValueAxis.valueMax = 5
lp.xValueAxis.valueSteps = [1, 2, 2.5, 3, 4, 5]
lp.xValueAxis.labelXFormat = '%2.1f'
lp.yValueAxis.valueMin = 0 lp.yValueAxis.valueMax
= 7 lp.yValueAxis.valueSteps = [1, 2, 3, 5, 6]

dibujo.add(lp)

```

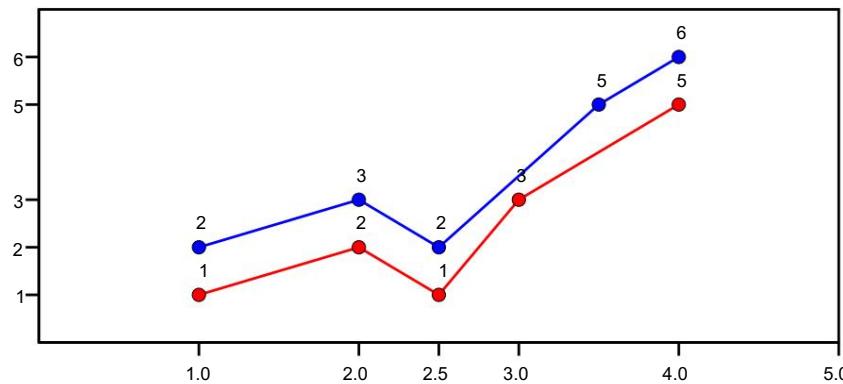


Figura 11-9: Ejemplo de trazado de líneas

Propiedad	Significado
datos	Datos a trazar, lista de (listas de) números.
x, y, ancho, alto	Cuadro delimitador del gráfico de líneas. Tenga en cuenta que xey NO especifican el centro sino la esquina inferior izquierda
xValorEje	El eje de valor vertical, que puede formatearse como se describió anteriormente.
yValorEje	El eje de valores horizontal, que puede formatearse como se describió anteriormente.
color del trazo	El valor predeterminado es Ninguno. Esto dibujará un borde alrededor del rectángulo del trazado, lo que puede resultar útil en la depuración. Los ejes sobreescibirán esto.
anchura del trazo	El valor predeterminado es Ninguno. Ancho del borde alrededor del rectángulo de trazado.
color de relleno	El valor predeterminado es Ninguno. Esto llenará el rectángulo del trazado con un color sólido.
líneas.trazoColor	Color de la línea.

líneas.strokeWidth Ancho de la línea.	
líneas.símbolo	Marcador utilizado para cada punto. Puedes crear un nuevo marcador usando la función makeMarker(). Por ejemplo, para usar un círculo, la llamada a la función sería makeMarker('Circle')
líneaEtiquetas	Una colección de etiquetas utilizadas para formatear todas las etiquetas de línea. Dado que se trata de una matriz bidimensional, puede formatear explícitamente la tercera etiqueta de la segunda línea usando esta sintaxis: chart.lineLabels[(1,2)].fontSize = 12
formato de etiqueta de línea	El valor predeterminado es Ninguno. Al igual que con YValueAxis, si proporciona una función o una cadena de formato, se dibujarán etiquetas junto a cada línea que muestre el valor numérico. También puede configurarlo en 'valores' para mostrar los valores definidos explícitamente en lineLabelArray.
líneaEtiquetaArray	Matriz explícita de valores de etiquetas de línea; debe coincidir con el tamaño de los datos, si están presentes. Los valores de estas etiquetas se mostrarán solo si la propiedad lineLabelFormat anterior está configurada en 'valores'.

Tabla 11-10 - Propiedades de LinePlot

11.9 Gráficos circulares

Como es habitual, comenzaremos con un ejemplo:

```
de reportlab.graphics.charts.piecharts importar Pie d = Dibujo(200, 100)
```

```
pc = Pie() pc.x =
65 pc.y = 15
pc.ancho = 70
pc.alto = 70 pc.data =
[10,20,30,40,50,60]
pc.labels = ['a', 'b', 'c', 'd', 'e', 'f']

pc.slices.strokeWidth=0.5 pc.slices[3].popout =
10 pc.slices[3].strokeWidth = 2
pc.slices[3].strokeDashArray = [2,2]
pc.slices[3].labelRadius = 1.75 pc.rebanadas[3].fontColor =
colores.rojo d.add(pc)
```

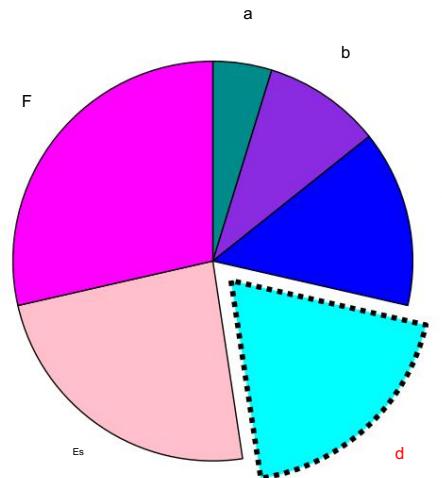


Figura 11-10: Un gráfico circular básico

Las propiedades se tratan a continuación. El pastel tiene una colección de "porciones" y documentamos las propiedades de las cuñas en la misma tabla.

Propiedad	Significado
datos	Una lista o tupla de números.
x, y, ancho, alto	Cuadro delimitador del pastel. Tenga en cuenta que x e y NO especifican el centro sino la esquina inferior izquierda, y que el ancho y el alto no tienen que ser iguales; Los pasteles pueden ser elípticos y las rebanadas se dibujarán correctamente.
etiquetas	Ninguno o una lista de cadenas. Hazlo Ninguno si no quieres etiquetas alrededor del borde del pastel. Como es imposible saber el tamaño de las rebanadas, generalmente desaconsejamos colocar etiquetas dentro o alrededor de los pasteles; es mucho mejor ponerlos en una leyenda al lado.
ángulo inicial	¿Dónde está el ángulo inicial del primer sector del pastel? El valor predeterminado es '90', que son las doce en punto.
dirección	¿En qué dirección avanzan los cortes? El valor predeterminado es "en el sentido de las agujas del reloj".
etiquetas laterales	Esto crea un gráfico con las etiquetas en dos columnas, una a cada lado.
etiquetas lateralesDesplazamiento	Esta es una fracción del ancho del pastel que define la distancia horizontal entre el pastel y las columnas de etiquetas.
etiquetas simples	El valor predeterminado es 1. Establezca en 0 para habilitar el uso de etiquetas personalizables y de propiedades con el prefijo label_ en los sectores de la colección.
rebanadas	Colección de rebanadas. Esto le permite personalizar cada cuña o individualmente. Vea abajo
rebanadas.trazoAncho	Ancho del borde de la cuña
rebanadas.trazoColor	Color del borde
rebanadas.strokeDashArray	Configuración de línea continua o discontinua
rebanadas.popout	¿A qué distancia deben sobresalir las rebanadas del centro del pastel? El valor predeterminado es cero.
rebanadas.fontSize	Tamaño de la fuente de la etiqueta
rebanadas.textColor	Color del texto de la etiqueta.
rebanadas.labelXRadius	Esto controla el punto de anclaje de una etiqueta de texto. Es una fracción del radio; 0.7 colocará el texto dentro del pastel, 1.2 lo colocará ligeramente afuera. (tenga en cuenta que si agregamos etiquetas, las mantendremos para especificar su punto de anclaje)

Tabla 11-11 - Propiedades circulares

Personalización de etiquetas

Cada etiqueta de diapositiva se puede personalizar individualmente cambiando las propiedades con el prefijo label_ en los sectores de la colección. Por ejemplo pc.slices[2].label_angle = 10 cambia el ángulo de la tercera etiqueta.

Antes de poder utilizar estas propiedades de personalización, debe deshabilitar las etiquetas simples con: pc.simpleLabels = 0

Propiedad	Significado
etiqueta_dx	X Desplazamiento de la etiqueta
etiqueta_tu	Y Desplazamiento de la etiqueta
ángulo_etiqueta	Ángulo de la etiqueta, el valor predeterminado (0) es horizontal, 90 es vertical, 180 está al revés
label_boxAncla	Punto de anclaje de la etiqueta
label_boxColor de trazo	Color del borde del cuadro de etiqueta
label_boxAncho de trazo	Ancho del borde del cuadro de etiqueta
label_boxFillColor	Color de relleno del cuadro de etiqueta.
etiqueta_trazoColor	Color del borde del texto de la etiqueta
label_strokeAncho	Ancho del borde para el texto de la etiqueta
texto_etiqueta	Texto de la etiqueta
ancho_etiqueta	Ancho de la etiqueta
label_maxWidth	Ancho máximo al que puede crecer la etiqueta
altura_etiqueta	Altura de la etiqueta
etiqueta_textoAncla	Altura máxima que puede alcanzar la etiqueta
etiqueta_visible	Verdadero si se va a dibujar la etiqueta
label_topPadding	Acolchado en la parte superior de la caja
label_leftPadding	Relleno a la izquierda del cuadro
label_rightPadding	Relleno a la derecha del cuadro
label_bottomPadding	Acolchado en la parte inferior de la caja.
etiqueta_puntero_simple	Establecer en 1 para punteros simples
label_pointer_strokeColor	Color de la línea indicadora
label_pointer_strokeWidth	Ancho de la línea del indicador

Tabla 11-12: Propiedades de personalización de etiquetas Pie.slices

Etiquetas laterales

Si el atributo sideLabels se establece en verdadero, entonces las etiquetas de los sectores se colocan en dos columnas, una en cada lado del pastel y el ángulo inicial del pastel se establecerán automáticamente. El ancla de la columna de la derecha es establecido en 'inicio' y el ancla de la columna de la izquierda está establecido en 'fin'. La distancia desde el borde del pastel desde El borde de cada columna lo decide el atributo sideLabelsOffset, que es una fracción del ancho de la columna. tarta. Si se cambia xradius, el pastel puede superponerse a las etiquetas, por lo que recomendamos dejar xradius como Ninguno. Hay un ejemplo a continuación.

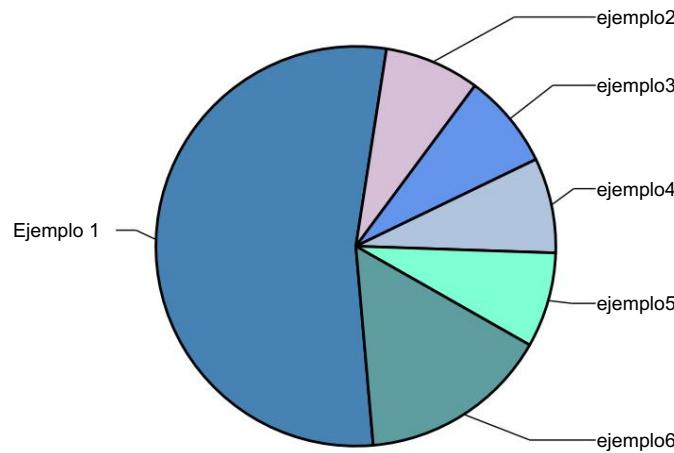


Figura 11-11: Un ejemplo de un gráfico circular con sideLabels =1

Si tiene sideLabels configurado en True, algunos de los atributos se vuelven redundantes, como pointerLabelMode.

Además, sideLabelsOffset solo cambia el gráfico circular si sideLabels está configurado en verdadero.

Algunos asuntos

Los punteros pueden cruzarse si hay demasiados sectores.

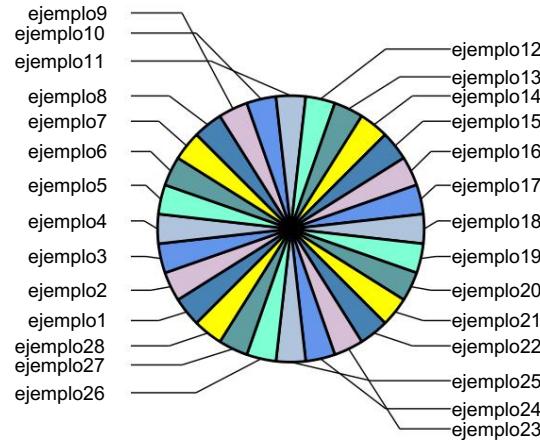


Figura 11-12: Un ejemplo de cruce de punteros

Además, las etiquetas pueden superponerse a pesar de checkLabelOverlap si corresponden a sectores que no son adyacentes.

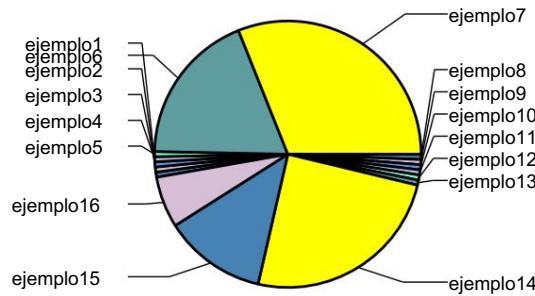


Figura 11-13: Un ejemplo de etiquetas superpuestas

11.10 Leyendas

Se pueden encontrar varias clases de leyendas preliminares, pero es necesario limpiarlas para que sean coherentes con el resto del modelo de gráficos. Las leyendas son el lugar natural para especificar los colores y estilos de línea de los gráficos; Proponemos que cada gráfico se cree con un atributo de leyenda que es invisible. Entonces se haría lo siguiente para especificar colores:

```
myChart.legend.defaultColors = [rojo, verde, azul]
```

También se podría definir un grupo de gráficos que comparten la misma leyenda:

```
myLegend = Legend()
myLegend.defaultColor = [rojo, verde,...] #¡qué asco!
myLegend.columns = 2
# etc. chart1.legend = myLegend

chart2.legend = myLegend chart3.legend
= myLegend
```

¿Esto funciona? ¿Es una complicación aceptable respecto a especificar directamente los colores del gráfico?

Cuestiones pendientes

Hay varios problemas que están casi resueltos, pero es demasiado pronto para empezar a hacerlos públicos. Sin embargo, aquí hay una lista de cosas que están en marcha:

- Especificación de color: en este momento el gráfico tiene una propiedad no documentada defaultColors, que proporciona una lista de colores para recorrer, de modo que cada serie de datos obtenga su propio color.
En este momento, si introduce una leyenda, debe asegurarse de que comparta la misma lista de colores.
Lo más probable es que esto sea reemplazado por un esquema para especificar una especie de leyenda que contenga atributos con diferentes valores para cada serie de datos. Esta leyenda también puede ser compartida por varios gráficos, pero no es necesario que sea visible.
- Tipos de gráficos adicionales: cuando el diseño actual se haya vuelto más estable, esperamos agregar variantes de gráficos de barras para tratar con barras de percentiles, así como la variante de lado a lado que se ve aquí.

panorama

Tomará algún tiempo abordar toda la gama de tipos de gráficos. Esperamos finalizar las barras y los pasteles primero y, posteriormente, producir implementaciones de prueba de tramas más generales.

Gráficos XY

La mayoría de los demás gráficos implican dos ejes de valores y trazan directamente datos xy de alguna forma. La serie se puede trazar como líneas, símbolos de marcador, ambos o gráficos personalizados, como gráficos de apertura, alto, bajo y cierre. Todos comparten los conceptos de escala y formato de eje/título. En cierto punto, una rutina recorrerá la serie de datos y 'hará' algo' con los puntos de datos en ubicaciones xy determinadas. Dado un gráfico lineal básico, debería ser muy fácil derivar un tipo de gráfico personalizado simplemente anulando un único método, por ejemplo, `drawSeries()`.

Personalización de marcadores y formas personalizadas.

Paquetes de trazado conocidos como Excel, Mathematica y Excel ofrecen una variedad de tipos de marcadores para agregar. gráficos. Podemos hacerlo mejor: puede escribir cualquier tipo de widget de gráfico que desee y simplemente decirle al gráfico que lo use como un ejemplo.

Parcelas combinadas

Combinar varios tipos de tramas es realmente fácil. Puedes simplemente dibujar varios gráficos (de barras, de líneas o lo que sea) en el mismo rectángulo, suprimiendo los ejes según sea necesario. Entonces, un gráfico podría correlacionar una línea con los casos de tifoidea escoceses a lo largo de un período de 15 años en el eje izquierdo con un conjunto de barras que muestran las tasas de inflación en el eje derecho. Si alguien puede recordarnos de dónde vino este ejemplo, lo atribuiremos y con gusto mostraremos el conocido gráfico como ejemplo.

editores interactivos

Un principio del paquete Graphics es hacer que todas las propiedades "interesantes" de sus componentes gráficos sean accesibles y modificables estableciendo valores apropiados de los atributos públicos correspondientes. Esto hace que sea muy tentador crear una herramienta como un editor GUI que le ayude a hacerlo de forma interactiva.

ReportLab ha creado una herramienta de este tipo utilizando el kit de herramientas Tkinter que carga código Python puro que describe un dibujo. y registra las operaciones de edición de su propiedad. Este "historial de cambios" se utiliza luego para crear código para una subclase de ese gráfico, por ejemplo, que se puede guardar y utilizar instantáneamente como cualquier otro gráfico o como un nuevo punto de partida para Otra sesión de edición interactiva.

Sin embargo, esto todavía es un trabajo en progreso y las condiciones para su liberación deben elaborarse más a fondo.

Varios.

Este no ha sido un análisis exhaustivo de todas las clases de gráficos. Se trabaja constantemente en esas clases. A Para ver exactamente qué hay en la distribución actual, utilice la utilidad `Graphdocpy.py`. De forma predeterminada, se ejecutará en `reportlab/graphics` y producir un informe completo. (Si desea ejecutarlo en otros módulos o paquetes, `graphdocpy.py -h` imprime un mensaje de ayuda que le indicará cómo hacerlo).

Esta es la herramienta que se mencionó en la sección sobre 'Widgets de documentación'.

11.11 Formas

Esta sección describe el concepto de formas y su importancia como componentes básicos de todos los resultados generados. por la biblioteca de gráficos. Se presentan algunas propiedades de las formas existentes y su relación con los diagramas. y se introduce brevemente la noción de tener diferentes renderizadores para diferentes formatos de salida.

Formas disponibles

Los dibujos se componen de formas. Absolutamente cualquier cosa puede construirse combinando el mismo conjunto de elementos primitivos. formas. El módulo `formas.py` proporciona una serie de formas y construcciones primitivas que se pueden agregar a un dibujo. Ellos son:

- recto
- Círculo
- Elipse
- Cuña (una rebanada de pastel)
- Polígono
- Línea

- Polilínea
- Cadena
- Grupo
- Ruta (aún no implementada, pero se agregará en el futuro)

El siguiente dibujo, tomado de nuestro conjunto de pruebas, muestra la mayoría de las formas básicas (excepto los grupos). Aquellos con una superficie rellena de color verde también se denominan formas sólidas (éstas son Rectángulo, Círculo, Elipse, Cuña y Polígono).

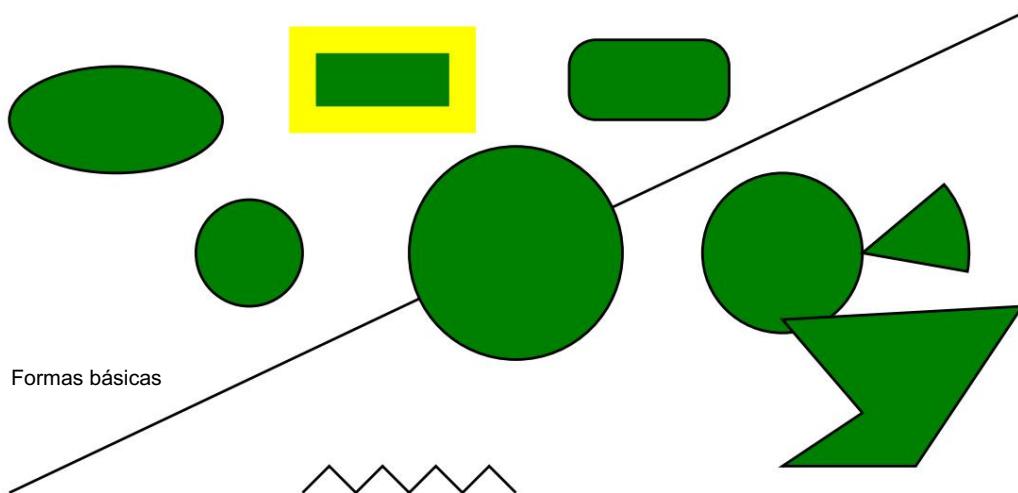


Figura 11-14: Formas básicas

Propiedades de forma

Las formas tienen dos tipos de propiedades: algunas para definir su geometría y otras para definir su estilo. Creemos un rectángulo rojo con bordes verdes de 3 puntos de espesor:

```
>>> desde reportlab.graphics.shapes importar Rect
>>> desde reportlab.lib.colors importar rojo, verde
>>> r = Rect(5, 5, 200, 100)
>>> r.fillColor = rojo
>>> r.strokeColor = verde
>>> r.strokeWidth = 3
>>>
```



Figura 11-15: rectángulo rojo con verde borde

Nota: En ejemplos futuros omitiremos las declaraciones de importación.

Todas las formas tienen una serie de propiedades que se pueden configurar. En un mensaje interactivo, podemos usar su método `dumpProperties()` para enumerarlos. Esto es lo que puedes usar para configurar un `Rect`:

```
>>> r.dumpProperties() fillColor =
Color(1.00,0.00,0.00) height = 100 rx = 0 ry = 0 trazoColor =
Color(0.00,0.50,0.00)
trazoDashArray
= Ninguno
trazoLineCap = 0 trazoLineJoin = 0 trazoMiterLimit = 0 trazoAncho = 3 ancho
= 200 x = 5 y = 5 >>>
```

Las formas generalmente tienen propiedades de estilo y propiedades de geometría. `x`, `y`, `ancho` y `alto` son parte de la geometría y deben proporcionarse al crear el rectángulo, ya que no tiene mucho sentido sin esas propiedades. Los demás son opcionales y vienen con valores predeterminados sensatos.

Puede establecer otras propiedades en líneas posteriores o pasándolas como argumentosopcionales a la construcción-`or`. También podríamos haber creado nuestro rectángulo de esta manera:

```
>>> r = Rect(5, 5, 200, 100,
            fillColor=rojo,
            trazoColor=verde,
            trazoAncho=3)
```

Repasemos las propiedades de estilo. `fillColor` es obvio. `trazo` es terminología publicada para el borde de una forma; el trazo tiene un color, ancho, posiblemente un patrón de guión y algunas características (rara vez utilizadas) de lo que sucede cuando una línea gira en una esquina. `rx` y `ry` son propiedades geométricas opcionales y se utilizan para definir el radio de la esquina de un rectángulo redondeado.

Todas las demás formas sólidas comparten las mismas propiedades de estilo.

Líneas

Proporcionamos líneas rectas individuales, PolyLines y curvas. Las líneas tienen todas las propiedades de trazo*, pero no `fillColor`. A continuación se muestran algunos ejemplos de líneas y polilíneas y la salida de gráficos correspondiente:

```
Línea(50,50, 300,100,
      colortrazo=colores.azul,anchotrazo=5)
Línea(50,100, 300,50,
      trazoColor=colores.rojo, trazoWidth=10,
      trazoDashArray=[10,
                     20])
Polilínea([120,110, 130,150, 140,110, 150,150, 160,110,
           170,150, 180,110, 190,150, 200,110], ancho de trazo = 2,
           color de trazo =
           colores.púrpura)
```

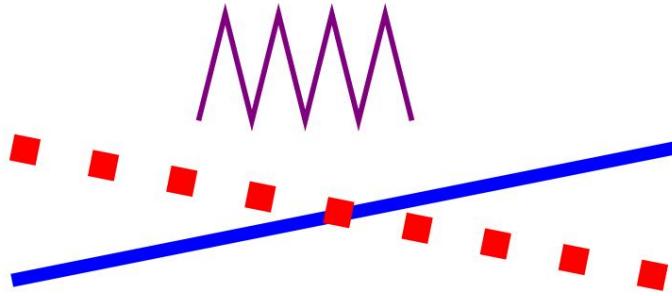


Figura 11-16: Ejemplos de líneas y polilíneas

Instrumentos de cuerda

El paquete ReportLab Graphics no está diseñado para un diseño de texto sofisticado, pero puede colocar cadenas en las ubicaciones deseadas y con alineación izquierda/derecha/centro. Especifiquemos un objeto String y observemos sus propiedades:

```
>>> s = String(10, 50, 'Hola Mundo') >>> s.dumpProperties()
Color(0.00,0.00,0.00) fontName = Times-
Roman fontSize = 10 texto = Hola Mundo textAnchor = inicio x =
10 años = 50
```

>>>

Las cadenas tienen una propiedad textAnchor, que puede tener uno de los valores "inicio", "medio", "fin". Si se establece en 'inicio', x e y se relacionan con el inicio de la cadena, y así sucesivamente. Esto proporciona una manera fácil de alinear el texto.

Las cadenas utilizan un estándar de fuente común: las fuentes Postscript Tipo 1 presentes en Acrobat Reader. De este modo podemos utilizar las 14 fuentes básicas en ReportLab y obtener métricas precisas para ellas. Recientemente también agregamos soporte para fuentes Tipo 1 adicionales y todos los renderizadores saben cómo renderizar fuentes Tipo 1.

Aquí hay un ejemplo más sofisticado usando el fragmento de código a continuación. Consulte la Guía del usuario de ReportLab para ver cómo se registran las fuentes no estándar como 'DarkGardenMK'.

```
d = Dibujo(400, 200) para el
tamaño en el rango(12, 36, 4):
    d.add(String(10+tamaño*2, 10+tamaño*2, 'Hola mundo', fontName='Times-
    Roman' , tamaño de fuente =
    tamaño))
    d.add(String(130, 120, 'Hola mundo', fontName='Mensajero',
    fontSize=36))

    d.add(String(150, 160, 'Hola mundo',
    fontName='DarkGardenMK',
    fontSize=36))
```

Hola Mundo

Hola Mundo



Figura 11-17: ejemplo de fuente elegante

Caminos

Las rutas Postscript son un concepto ampliamente comprendido en gráficos. Todavía no están implementados en reportlab/graphics, pero lo estarán pronto.

Grupos

Finalmente, tenemos objetos de grupo. Un grupo tiene una lista de contenidos, que son otros nodos. También puede aplicar una transformación: su contenido se puede rotar, escalar o desplazar. Si conoce las matemáticas, puede configurar la transformación directamente. De lo contrario, proporciona métodos para rotar, escalar, etc. Aquí hacemos un grupo que se rota y se traduce:

```
>>> g = Grupo(forma1, forma2, forma3) >>> g.rotate(30)
>>> g.translate(50, 200)
```

Los grupos proporcionan una herramienta para la reutilización. Puedes crear un montón de formas para representar algún componente (por ejemplo, un sistema de coordenadas) y colocarlas en un grupo llamado "Eje". Luego puedes poner ese grupo en otros grupos, cada uno con una traslación y rotación diferente, y obtendrás un montón de ejes. Sigue siendo el mismo grupo, sorteado en diferentes lugares.

Hagamos esto con un poco más de código:

```
d = Dibujo(400, 200)

Eje = Grupo (Línea
    (0,0,100,0), # Eje x Línea (0,0,0,50), #
    Eje y Línea (0,10,10,10), # ticks en el eje
    y Línea (0, 20,10,20), Línea(0,30,10,30),
    Línea(0,40,10,40),
    Línea(10,0,10,10), # ticks
    en el eje x Línea(20,
    ,20,10), Línea(30,0,30,10), Línea(40,0,40,10),
    Línea(50,0,50,10),
    Línea(60,0,60,10),
    Línea(70,0,70,10),
    Línea(80,0,80,10),
    Línea(90,0,90,10),
    Cadena(20, 35, 'Ejes',
    relleno=colores.negro))
```

```

firstAxisGroup = Grupo(Eje)
firstAxisGroup.translate(10,10) d.add(firstAxisGroup)

secondAxisGroup = Grupo(Eje)
secondAxisGroup.translate(150,10)
secondAxisGroup.rotate(15)

d.add (segundo grupo de ejes)

tercerGrupoEje = Grupo(Eje,
                        transformar = mmult (traducir (300,10), rotar (30)))

d.add(tercer grupo de ejes)

```

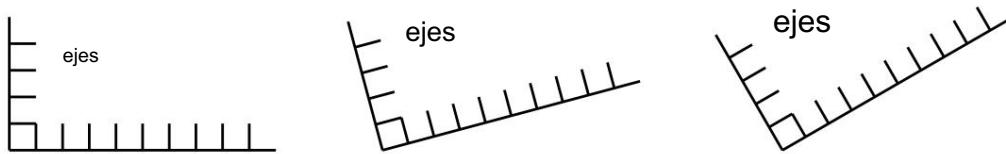


Figura 11-18: Ejemplos de grupos

11.12 Aparatos

Ahora describimos los widgets y cómo se relacionan con las formas. Utilizando muchos ejemplos, se muestra cómo los widgets crean componentes gráficos reutilizables.

Formas versus widgets

Hasta ahora, los dibujos han sido "puros datos". No hay ningún código en ellos para hacer nada, excepto ayudar al programador a verificar e inspeccionar el dibujo. De hecho, esa es la piedra angular de todo el concepto y es lo que nos permite lograr la portabilidad: un renderizador sólo necesita implementar las formas primitivas.

Queremos crear objetos gráficos reutilizables, incluida una potente biblioteca de gráficos. Para ello necesitamos reutilizar cosas más tangibles que rectángulos y círculos. Deberíamos poder escribir objetos para que otros los reutilicen: flechas, engranajes, cuadros de texto, nodos de diagramas UML e incluso gráficos completos.

El estándar de widgets es un estándar integrado sobre el módulo de formas. Cualquiera puede escribir nuevos widgets y nosotros podemos crear bibliotecas a partir de ellos. Los widgets admiten los métodos `getProperties()` y `setProperties()`, por lo que puede inspeccionarlos, modificarlos y documentarlos de manera uniforme.

- Un widget es una forma
- reutilizable que se puede inicializar sin argumentos cuando se llama a su método `draw()`. Crea una forma primitiva o un grupo para representarse
- a sí mismo. Puede tener los parámetros que desee y pueden controlar la forma en que se dibuja. tiene un método `demo()` que debería devolver un ejemplo atractivo de sí mismo en un rectángulo de 200x100. Esta es la piedra angular de las herramientas de documentación automática. El método `demo()` también debe tener una cadena de documentación bien escrita, ¡ya que también está impresa!

Los widgets van en contra de la idea de que un dibujo es sólo un conjunto de formas; ¿Seguramente tienen su propio código? La forma en que funcionan es que un widget puede convertirse en un grupo de formas primitivas. Si algunos de sus componentes son en sí mismos widgets, también se convertirán. Esto sucede automáticamente durante el renderizado; el renderizador no verá el widget de su gráfico, sino solo una colección de rectángulos, líneas y cadenas. También puedes 'aplanar' explícitamente un dibujo, haciendo que todos los widgets se conviertan en primitivos.

Usando un widget

Imaginemos un nuevo widget simple. Usaremos un widget para dibujar una cara y luego mostraremos cómo se implementó.

```
>>> desde reportlab.lib importar colores >>> desde reportlab.graphics importar
formas >>> desde reportlab.graphics importar widgetbase >>> desde reportlab.graphics
importar renderPDF >>> d = formas.Dibujo(200, 100) >>> f = widgetbase.Face() >>> f.skinColor =
colores.amarillo >>> f.mood = "triste" >>> d.add(f) >>> renderPDF.drawToFile(d, "face.pdf", "Una
cara")
```

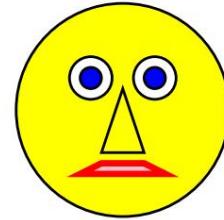


Figura 11-19: Un widget de muestra

Veamos qué propiedades tiene disponibles, usando el método `setProperties()` que hemos visto anteriormente:

```
>>> f.dumpProperties() colorojo =
Color(0.00,0.00,1.00) humor = triste

tamaño = 80
skinColor = Color(1.00,1.00,0.00) x = 10 y = 10 >>>
```

Una cosa que parece extraña sobre el código anterior es que no establecimos el tamaño o la posición cuando hicimos la cara. Esta es una compensación necesaria para permitir una interfaz uniforme para construir widgets y documentarlos; no pueden requerir argumentos en su método `__init__()`. En cambio, generalmente están diseñados para caber en una ventana de 200 x 100, y puedes moverlos o cambiar su tamaño estableciendo propiedades como `x`, `y`, `ancho`, etc. después de su creación.

Además, un widget siempre proporciona un método `demo()`. Los simples como este siempre hacen algo sensato antes de configurar las propiedades, pero los más complejos como un gráfico no tendrían datos para trazar. La herramienta de documentación llama a `demo()` para que su nueva y elegante clase de gráfico pueda crear un dibujo que muestre lo que puede hacer.

Aquí hay algunos widgets simples disponibles en el módulo `signos y símbolos.py`:

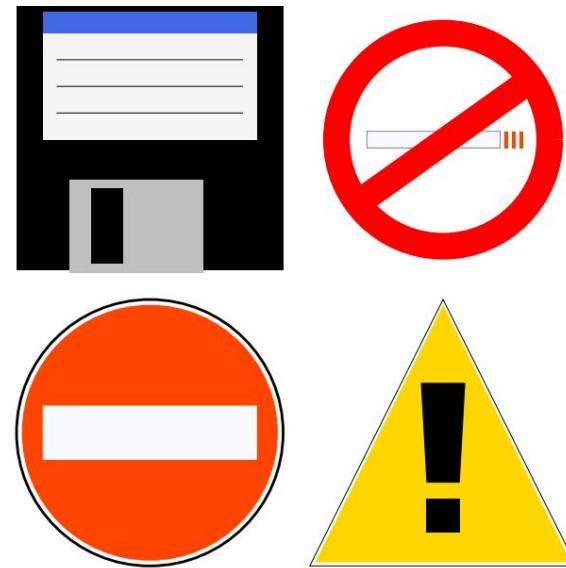


Figura 11-20: Algunas muestras de signos y símbolos.py

Y este es el código necesario para generarlos como se ve en el dibujo de arriba:

```
desde reportlab.graphics.shapes importar dibujo desde reportlab.graphics.widgets
importar signos y símbolos

d = Dibujo(230, 230)

ne = signos y símbolos.NoEntry() ds = signos y
símbolos.DangerSign() fd = signos y símbolos.FloppyDisk() ns
= signos y símbolos.No fumar()

ne.x, ne.y = 10, 10 ds.x, ds.y = 120,
10 fd.x, fd.y = 10, 120 ns.x, ns.y = 120,
120

d.añadir(ne)
d.añadir(ds)
d.añadir(fd)
d.añadir(ns)
```

Widgets compuestos

Imaginemos un widget compuesto que dibuja dos caras una al lado de la otra. Esto es fácil de crear cuando tienes el widget Cara.

```
>>> tf = widgetbase.TwoFaces() >>> tf.faceOne.mood

'feliz' >>>
tf.faceTwo.mood 'triste'

>>> tf.dumpProperties() faceOne.eyeColor
= Color(0.00,0.00,1.00) faceOne.mood = cara felizOne.size = 80
faceOne.skinColor = Ninguno faceOne.x
= 10 faceOne.y = 10
faceTwo.eyeColor = Color (0.00,0.00,1.00)
faceTwo.mood = triste
faceTwo.size = 80
faceTwo.skinColor = Ninguno
```

```
caraDos.x = 100 caraDos.y
= 10
>>>
```

Los atributos 'faceOne' y 'faceTwo' están expuestos deliberadamente para que puedas acceder a ellos directamente. También podría haber atributos de nivel superior, pero en este caso no los hay.

Verificación de widgets

El diseñador del widget decide la política de verificación, pero de forma predeterminada funcionan como formas (verificando cada asignación) si el diseñador ha proporcionado la información de verificación.

Implementación de widgets

Intentamos que la implementación de widgets fuera lo más fácil posible. Aquí está el código para un widget Face que no realiza ninguna verificación de tipo:

```
class Cara(Widget):
    """Esto dibuja
    una cara con dos ojos, boca y nariz."""

    def __init__(self):
        self.x = 10
        self.y = 10
        self.size = 80
        self.skinColor = Ninguno
        self.eyeColor = colores.azul
        self.mood = 'feliz'

    def dibujar(self):
        s = self.size
        # abreviar ya que usaremos esto mucho
        g = formas.Grupo()
        g.transform = [1, 0, 0, 1, self.x, self.y] # fondo
        g.add(shapes.Circle(s * 0.5, s * 0.5, s * 0.5, fillColor=self.skinColor))

# CÓDIGO OMITIDO PARA HACER MÁS FORMAS regresar g
```

Omitimos todo el código para dibujar las formas en este documento, pero puedes encontrarlo en la distribución en `widgetbase.py`.

De forma predeterminada, `setProperties` devuelve cualquier atributo sin un guión bajo inicial. Esta es una política deliberada para fomentar convenciones de codificación consistentes.

Una vez que su widget funcione, probablemente desee agregar soporte para verificación. Esto implica agregar un diccionario a la clase llamada `_verifyMap`, que asigna desde nombres de atributos a 'funciones de verificación'. El módulo `widgetbase.py` define un montón de funciones de verificación con nombres como `isNumber`, `isListOfShapes`, etc.

También puedes simplemente usar Ninguno, lo que significa que el atributo debe estar presente pero puede ser de cualquier tipo. Y puedes y debes escribir tus propias funciones de verificación. Queremos restringir el atributo personalizado "estado de ánimo" a los valores "feliz", "triste" u "bien". Entonces hacemos esto:

```
class Cara(Widget):
    """Esto dibuja
    una cara con dos ojos. Expone un par de propiedades para configurarse y oculta todos los
    demás detalles"""
    def checkMood(moodName):
        return (moodName in ('happy', 'triste', 'ok'))
    _verifyMap = {
        'x': formas.isNumber,
        'y': formas.isNumber,
        'tamaño': formas.isNumber,
        'skinColor': shapes.isColorOrNone,
        'eyeColor': formas.isColorOrNone,
        'estado de ánimo': checkMood
    }
```

Esta verificación se realizará en cada asignación de atributo; o, si `config.shapeChecking` está desactivado, cada vez que llame a `myFace.verify()`.

Documentación de widgets

Estamos trabajando en una herramienta genérica para documentar cualquier paquete o módulo de Python; esto ya está registrado en ReportLab y se utilizará para generar una referencia para el paquete ReportLab. Cuando encuentra widgets, agrega secciones adicionales al manual que

- incluyen: la cadena de documentación
- para su clase de widget el fragmento de código de su método demo() , para que la gente pueda ver cómo usarlo el dibujo producido
- por el método demo() la propiedad volcado para el widget en el dibujo.

Esta herramienta permitirá que podamos tener garantizada documentación actualizada en nuestros widgets y gráficos, tanto en el sitio web como impreso; ¡Y que también puedes hacer lo mismo con tus propios widgets!

Estrategias de diseño de widgets

No pudimos encontrar una arquitectura consistente para diseñar widgets, ¡así que dejamos ese problema a los autores! Si no le gusta la estrategia de verificación predeterminada o la forma en que funciona setProperties/getProperties, puede anularlas usted mismo.

Para widgets simples, se recomienda hacer lo que hicimos anteriormente: seleccionar propiedades que no se superpongan, inicializar cada propiedad en __init__ y construir todo cuando se llame a draw(). En su lugar, puede tener ganchos __setattr__ y actualizar las cosas cuando se configuran ciertos atributos. Considere un gráfico circular. Si desea exponer los sectores individuales, puede escribir un código como este:

```
de reportlab.graphics.charts importar gráficos circulares pc =
piecharts.Pie() pc.defaultColors
= [navy, blue, skyblue] #usado en rotación pc.data = [10,30,50,25] pc.slices[7].ancho de
trazo = 5
```

La última línea es problemática ya que solo hemos creado cuatro sectores; de hecho, es posible que aún no los hayamos creado. ¿PC.slices[7] genera un error? ¿Es una prescripción de lo que debería suceder si se define una séptima cuña, utilizada para anular la configuración predeterminada? Por ahora, dejamos este problema directamente en el autor del widget y recomendamos que haga funcionar uno simple antes de exponer los 'objetos secundarios' cuya existencia depende de los valores de otras propiedades :-)

También discutimos las reglas mediante las cuales los widgets principales podrían pasar propiedades a sus hijos. Parece haber un deseo general de una forma global de decir que "todos los cortes obtienen su ancho de línea del ancho de línea de su padre" sin mucha codificación repetitiva. No tenemos una solución universal, así que dejemos eso a los autores de widgets. Esperamos que la gente experimente con enfoques push-down, pull-down y coincidencia de patrones y se les ocurra algo agradable. Mientras tanto, ciertamente podemos escribir widgets de gráficos monolíticos que funcionen como los de, por ejemplo, Visual Basic y Delphi.

Por ahora, eche un vistazo al siguiente código de muestra que utiliza una versión anterior de un widget de gráfico circular y el resultado que genera:

```
desde reportlab.lib.colors importar * desde
reportlab.graphics importar formas, renderizarPDF desde
reportlab.graphics.charts.piecharts importar Pie

d = Dibujo(400,200)
d.add(String(100,175,"Sin etiquetas", textAnchor="medio")) d.add(String(300,175,"Con
etiquetas", textAnchor="medio"))

pc = Pastel()
pc.x = 25 pc.y
= 50 pc.data
= [10,20,30,40,50,60] pc.slices[0].popout =
5 d.add(pc, 'pie1')

pc2 = Pastel()
pc2.x = 150
pc2.y = 50
pc2.data = [10,20,30,40,50,60] pc2.labels =
['a','b','c','d','e','f'] d.add(pc2, 'pie2')
```

```
pc3 = Pastel()  
pc3.x = 275  
pc3.y = 50  
pc3.data = [10,20,30,40,50,60] pc3.labels =  
['a','b','c','d','e','f'] pc3.slices.labelRadius = 0.65  
pc3.slices.fontSize = "Helvetica-Bold"  
pc3.slices.fontSize = 16 pc3.slices.fontColor =  
colores.amarillo d.add(pc3 , 'pastel3')
```

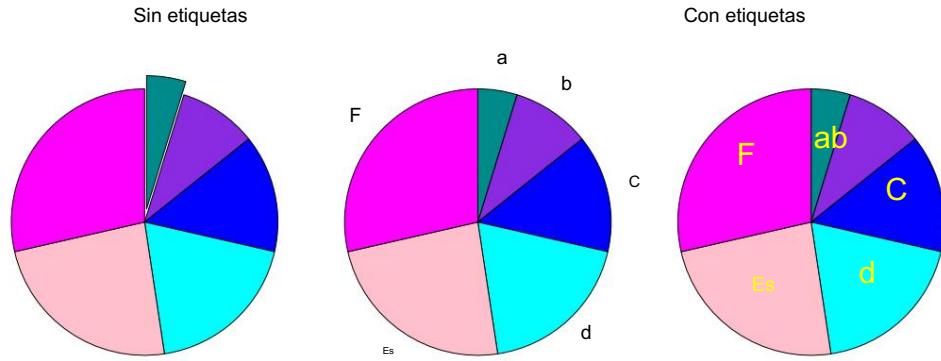


Figura 11-21: Algunas tartas de muestra

Apéndice A Demostraciones de ReportLab

En los subdirectorios de reportlab/demos hay una serie de ejemplos de trabajo que muestran casi todos los aspectos de reportlab en uso.

A.1 Odisea

Los tres scripts `odyssey.py`, `dodyssey.py` y `fodyssey.py` toman el archivo `odyssey.txt` y producen documentos PDF. El `odyssey.txt` incluido es breve; Puede encontrar una versión más larga y de prueba en <ftp://ftp.reportlab.com/odyssey.full.zip>.

```
Windows cd
reportlab\demos\odyssey python odyssey.py iniciar
odyssey.pdf
```

```
Linux cd
reportlab/demos/odyssey python odyssey.py acord
odyssey.pdf
```

El formato simple se muestra en el script `odyssey.py`. Funciona bastante rápido, pero todo lo que hace es reunir el texto y colocarlo en las páginas del lienzo. No manipula ningún párrafo en absoluto, por lo que puedes ver las etiquetas XML < >.

Los scripts `fodyssey.py` y `dodyssey.py` manejan el formato de párrafo para que puedas ver los cambios de color, etc.

Ambos scripts utilizan la clase de plantilla de documento y el script `dodyssey.py` muestra la capacidad de realizar un diseño de dos columnas y utiliza varias plantillas de página.

A.2 Fuentes y colores estándar

En `reportlab/demos/stdfonts`, el script `stdfonts.py` se puede utilizar para ilustrar las fuentes estándar de ReportLab. Ejecute el script usando

```
cd reportlab\demos\stdfonts python stdfonts.py
```

para producir dos documentos PDF, `StandardFonts_MacRoman.pdf` y `StandardFonts_WinAnsi.pdf` que muestran las dos codificaciones de fuentes integradas más comunes.

El script `colortest.py` en `reportlab/demos/colors` demuestra las diferentes formas en que `reportlab` puede configurar y usar colores.

Intenta ejecutar el script y ver el documento de salida, `colortest.pdf`. Esto muestra diferentes espacios de color y una gran selección de colores nombrados en el módulo `reportlab.lib.colors`.

A.3 Py2pdf

Dinu Gherman contribuyó con este útil script que utiliza `reportlab` para producir documentos PDF muy coloreados a partir de scripts de Python, incluidos marcadores para clases, métodos y funciones. Para obtener una buena versión del script principal, intenta

```
cd reportlab\demos\py2pdf python py2pdf.py
py2pdf.py acord py2pdf.pdf
```

es decir, utilizamos `py2pdf` para producir una buena versión de `py2pdf.py` en el documento con el mismo nombre raíz y una extensión `.pdf`.

El script `py2pdf.py` tiene muchas opciones que están más allá del alcance de esta sencilla introducción; Consulta los comentarios al inicio del guión.

A.4 Papel tábano

El script Python, gfe.py, en reportlab\demos\gadflypaper utiliza un estilo en línea de preparación de documentos. El guión producido casi en su totalidad por Aaron Watters produce un documento que describe el tábano de Aaron en la base de datos de memoria para Python. Para generar el documento utilice

```
cd reportlab\gadflypaper python  
gfe.py iniciar gfe.pdf
```

todo lo que hay en el documento PDF fue producido por el script, por lo que este es un estilo de producción de documentos en línea. Entonces, para producir un encabezado seguido de texto, el script usa las funciones header y p que toman algo de texto y lo agregan a una lista global de historias.

```
encabezado("Conclusión")  
  
p("""El diseño renovado del motor de consultas en Gadfly 2 admite  
.....  
e integración.""")
```

A.5 Punto de Python

Andy Robinson ha refinado el script pythonpoint.py (en reportlab\demos\pythonpoint) hasta convertirlo en un script realmente útil. Toma un archivo de entrada que contiene un marcado XML y utiliza un analizador de estilo xmllib para asignar las etiquetas a diapositivas PDF. Cuando se ejecuta en su propio directorio, pythonpoint.py toma como entrada predeterminada el archivo pythonpoint.xml y produce pythonpoint.pdf, que es la documentación para Pythonpoint. También puedes verlo en acción con un documento más antiguo.

```
cd reportlab\demos\pythonpoint python  
pythonpoint.py monterey.xml iniciar monterey.pdf
```

Pythonpoint no solo se autodocumenta, sino que también demuestra reportlab y PDF. Utiliza muchas características de reportlab (plantillas de documentos, tablas, etc.). Las características exóticas del PDF, como los desvanecimientos y los marcadores, también se muestran con buenos resultados. El uso de un documento XML se puede contrastar con el estilo en línea de la demostración de gadflypaper; el contenido está completamente separado del formato