

---

Documentation for package "reportlab.graphics"  
Generated by: graphdocpy.py version 0.8  
Date generated: 2013-05-07 20:19  
Format: PDF

---

reportlab.graphics	11
qr	11
Classes	11
QrCodeWidget(PlotArea)	11
Public Attributes	11
eanbc	13
Classes	13
Ean13BarcodeWidget(PlotArea)	13
Public Attributes	13
Ean8BarcodeWidget(Ean13BarcodeWidget)	14
Public Attributes	14
UPCA(Ean13BarcodeWidget)	15
Public Attributes	15
lineplots	16
Classes	16
AreaLinePlot(LinePlot)	16
Public Attributes	16
GridLinePlot(SimpleTimeSeriesPlot)	20
Public Attributes	20
LinePlot(AbstractLineChart)	25
Public Attributes	25
LinePlot3D(LinePlot)	29
Public Attributes	29
ScatterPlot(LinePlot)	33
Public Attributes	33
ShadedPolyFiller(Filler, ShadedPolygon)	37
Public Attributes	37
SimpleTimeSeriesPlot(LinePlot)	38
Public Attributes	38
SplitLinePlot(AreaLinePlot)	43
Public Attributes	43
Functions	50
sample1a( ... )	50
sample1b( ... )	51
sample1c( ... )	52
sample2( ... )	53
textlabels	55
Classes	55
BarChartLabel(PMVLabel)	55
Public Attributes	55

---

	Label(Widget) . . . . .	57
	Public Attributes . . . . .	57
	NA_Label(BarChartLabel) . . . . .	59
	Public Attributes . . . . .	59
	PMVLabel(Label) . . . . .	61
	Public Attributes . . . . .	61
spider . . . . .		63
Classes . . . . .		63
SpiderChart(PlotArea) . . . . .		63
Public Attributes . . . . .		63
SpokeLabel(WedgeLabel) . . . . .		65
Public Attributes . . . . .		65
StrandLabel(SpokeLabel) . . . . .		67
Public Attributes . . . . .		67
Functions . . . . .		69
sample1( ... ) . . . . .		69
sample2( ... ) . . . . .		71
dotbox . . . . .		73
Classes . . . . .		73
DotBox(Widget) . . . . .		73
Public Attributes . . . . .		73
axes . . . . .		75
Classes . . . . .		75
AdjYValueAxis(YValueAxis) . . . . .		75
Public Attributes . . . . .		75
CALabel(PMVLabel) . . . . .		79
Public Attributes . . . . .		79
CategoryAxis(_AxisG) . . . . .		81
Public Attributes . . . . .		81
NormalDateXValueAxis(XValueAxis) . . . . .		82
Public Attributes . . . . .		82
ValueAxis(_AxisG) . . . . .		86
Public Attributes . . . . .		86
XCategoryAxis(_XTicks, CategoryAxis) . . . . .		87
Public Attributes . . . . .		87
XValueAxis(_XTicks, ValueAxis) . . . . .		88
Public Attributes . . . . .		88
YCategoryAxis(_YTicks, CategoryAxis) . . . . .		92
Public Attributes . . . . .		92
YValueAxis(_YTicks, ValueAxis) . . . . .		93

---

	Public Attributes . . . . .	93
	_AxisG(Widget) . . . . .	97
	Public Attributes . . . . .	97
Functions . . . . .		98
	sample0a( ... ) . . . . .	98
	sample0b( ... ) . . . . .	99
	sample1( ... ) . . . . .	100
	sample4a( ... ) . . . . .	101
	sample4b( ... ) . . . . .	102
	sample4c( ... ) . . . . .	103
	sample4c1( ... ) . . . . .	104
	sample4d( ... ) . . . . .	105
	sample5a( ... ) . . . . .	106
	sample5b( ... ) . . . . .	107
	sample5c( ... ) . . . . .	108
	sample5d( ... ) . . . . .	109
	sample6a( ... ) . . . . .	110
	sample6b( ... ) . . . . .	111
	sample6c( ... ) . . . . .	112
	sample6d( ... ) . . . . .	113
	sample7a( ... ) . . . . .	114
	sample7b( ... ) . . . . .	115
	sample7c( ... ) . . . . .	116
	sample7d( ... ) . . . . .	117
areas . . . . .		118
	Classes . . . . .	118
	PlotArea(Widget) . . . . .	118
	Public Attributes . . . . .	118
slidebox . . . . .		119
	Classes . . . . .	119
	SlideBox(Widget) . . . . .	119
	Public Attributes . . . . .	119
linecharts . . . . .		121
	Classes . . . . .	121
	AbstractLineChart(PlotArea) . . . . .	121
	Public Attributes . . . . .	121
	HorizontalLineChart(LineChart) . . . . .	122
	Public Attributes . . . . .	122
	HorizontalLineChart3D(HorizontalLineChart) . . . . .	126
	Public Attributes . . . . .	126

---

	LineChart(AbstractLineChart) . . . . .	129
	Public Attributes . . . . .	129
	SampleHorizontalLineChart(HorizontalLineChart) <sup>130</sup>	
	Public Attributes . . . . .	130
	VerticalLineChart(LineChart) . . . . .	133
	Public Attributes . . . . .	133
Functions . . . . .		134
	sample1( ... ) . . . . .	134
	sample1a( ... ) . . . . .	135
	sample2( ... ) . . . . .	136
	sample3( ... ) . . . . .	137
piecharts . . . . .		138
Classes . . . . .		138
	AbstractPieChart(PlotArea) . . . . .	138
	Public Attributes . . . . .	138
	LegendedPie(Pie) . . . . .	139
	Public Attributes . . . . .	139
	Pie(AbstractPieChart) . . . . .	142
	Public Attributes . . . . .	142
	Pie3d(Pie) . . . . .	144
	Public Attributes . . . . .	144
	WedgeLabel(Label) . . . . .	146
	Public Attributes . . . . .	146
Functions . . . . .		148
	sample0a( ... ) . . . . .	148
	sample0b( ... ) . . . . .	149
	sample1( ... ) . . . . .	150
	sample2( ... ) . . . . .	151
	sample3( ... ) . . . . .	152
	sample4( ... ) . . . . .	153
	sample5( ... ) . . . . .	154
	sample6( ... ) . . . . .	155
	sample7( ... ) . . . . .	156
	sample8( ... ) . . . . .	157
	sample9( ... ) . . . . .	158
barcharts . . . . .		159
Classes . . . . .		159
	BarChart(PlotArea) . . . . .	159
	Public Attributes . . . . .	159
	BarChart3D(BarChart) . . . . .	160

Public Attributes . . . . .	160
HorizontalBarChart(BarChart) . . . . .	161
Public Attributes . . . . .	161
HorizontalBarChart3D(BarChart3D, HorizontalBarChart) . . . . .	164
Public Attributes . . . . .	164
SampleH5c4(Drawing) . . . . .	167
VerticalBarChart(BarChart) . . . . .	167
Public Attributes . . . . .	167
VerticalBarChart3D(BarChart3D, VerticalBarChart) <sub>on</sub> Public Attributes . . . . .	171
Functions . . . . .	174
sampleH0a( ... ) . . . . .	174
sampleH0b( ... ) . . . . .	175
sampleH0c( ... ) . . . . .	176
sampleH1( ... ) . . . . .	177
sampleH2a( ... ) . . . . .	178
sampleH2b( ... ) . . . . .	179
sampleH2c( ... ) . . . . .	180
sampleH3( ... ) . . . . .	182
sampleH4a( ... ) . . . . .	184
sampleH4b( ... ) . . . . .	185
sampleH4c( ... ) . . . . .	186
sampleH4d( ... ) . . . . .	187
sampleH5a( ... ) . . . . .	188
sampleH5b( ... ) . . . . .	189
sampleH5c1( ... ) . . . . .	190
sampleH5c2( ... ) . . . . .	191
sampleH5c3( ... ) . . . . .	192
sampleH5c4( ... ) . . . . .	193
sampleStacked1( ... ) . . . . .	194
sampleSymbol1( ... ) . . . . .	196
sampleV0a( ... ) . . . . .	198
sampleV0b( ... ) . . . . .	199
sampleV0c( ... ) . . . . .	200
sampleV1( ... ) . . . . .	201
sampleV2a( ... ) . . . . .	202
sampleV2b( ... ) . . . . .	203
sampleV2c( ... ) . . . . .	204
sampleV3( ... ) . . . . .	206
sampleV4a( ... ) . . . . .	208

	sampleV4b( ... ) . . . . .	209
	sampleV4c( ... ) . . . . .	210
	sampleV4d( ... ) . . . . .	211
	sampleV5a( ... ) . . . . .	212
	sampleV5b( ... ) . . . . .	213
	sampleV5c1( ... ) . . . . .	214
	sampleV5c2( ... ) . . . . .	215
	sampleV5c3( ... ) . . . . .	216
	sampleV5c4( ... ) . . . . .	217
legends . . . . .		218
Classes . . . . .		218
Legend(Widget) . . . . .		218
Public Attributes . . . . .		218
LineLegend(Legend) . . . . .		221
Public Attributes . . . . .		221
LineSwatch(Widget) . . . . .		223
Public Attributes . . . . .		223
Functions . . . . .		224
sample1c( ... ) . . . . .		224
sample2c( ... ) . . . . .		225
sample3( ... ) . . . . .		226
sample3a( ... ) . . . . .		227
doughnut . . . . .		228
Classes . . . . .		228
Doughnut(AbstractPieChart) . . . . .		228
Public Attributes . . . . .		228
Functions . . . . .		230
sample1( ... ) . . . . .		230
sample2( ... ) . . . . .		232
sample3( ... ) . . . . .		233
sample4( ... ) . . . . .		234
linechart_with_markers . . . . .		235
Classes . . . . .		235
LineChartWithMarkers(_DrawingEditorMixin, Drawing) . . . . .		235
line_chart . . . . .		236
Classes . . . . .		236
LineChart(_DrawingEditorMixin, Drawing) . . . . .		236
exploded_pie . . . . .		237
Classes . . . . .		238
ExplodedPie(_DrawingEditorMixin, Drawing) . . . . .		238

radar . . . . .	239
Classes . . . . .	239
RadarChart(_DrawingEditorMixin, Drawing) . . . . .	239
stacked_column . . . . .	240
Classes . . . . .	240
StackedColumn(_DrawingEditorMixin, Drawing) . . . . .	240
bubble . . . . .	241
Classes . . . . .	241
Bubble(_DrawingEditorMixin, Drawing) . . . . .	241
scatter_lines_markers . . . . .	243
Classes . . . . .	243
ScatterLinesMarkers(_DrawingEditorMixin, Drawing) . . . . .	243
scatter_lines . . . . .	244
Classes . . . . .	244
ScatterLines(_DrawingEditorMixin, Drawing) . . . . .	244
stacked_bar . . . . .	245
Classes . . . . .	246
StackedBar(_DrawingEditorMixin, Drawing) . . . . .	246
clustered_bar . . . . .	247
Classes . . . . .	247
ClusteredBar(_DrawingEditorMixin, Drawing) . . . . .	247
simple_pie . . . . .	248
Classes . . . . .	248
SimplePie(_DrawingEditorMixin, Drawing) . . . . .	248
filled_radar . . . . .	249
Classes . . . . .	249
FilledRadarChart(_DrawingEditorMixin, Drawing) . . . . .	250
scatter . . . . .	250
Classes . . . . .	251
Scatter(_DrawingEditorMixin, Drawing) . . . . .	251
clustered_column . . . . .	252
Classes . . . . .	252
ClusteredColumn(_DrawingEditorMixin, Drawing) . . . . .	252
table . . . . .	253
Classes . . . . .	253
TableWidget(Widget) . . . . .	253
Public Attributes . . . . .	253
grids . . . . .	255
Classes . . . . .	255
DoubleGrid(Widget) . . . . .	255



---

	Public Attributes	255
	Grid(Widget)	257
	Public Attributes	257
	ShadedPolygon(Widget, LineShape)	259
	Public Attributes	259
	ShadedRect(Widget)	260
	Public Attributes	260
flags		261
	Classes	261
	Flag(_Symbol)	261
	Public Attributes	261
	Star(_Symbol)	262
	Public Attributes	262
eventcal		262
	Classes	262
	EventCalendar(Widget)	262
	Public Attributes	262
signsandsymbols		263
	Classes	263
	ArrowOne(_Symbol)	263
	Public Attributes	263
	ArrowTwo(ArrowOne)	263
	Public Attributes	263
	Crossbox(_Symbol)	264
	Public Attributes	264
	DangerSign(_Symbol)	264
	Public Attributes	264
	ETriangle(_Symbol)	265
	Public Attributes	265
	FloppyDisk(_Symbol)	265
	Public Attributes	265
	NoEntry(_Symbol)	265
	Public Attributes	265
	NoSmoking(NotAllowed)	266
	Public Attributes	266
	NotAllowed(_Symbol)	266
	Public Attributes	266
	Octagon(_Symbol)	266
	Public Attributes	267
	RTriangle(_Symbol)	267

---

Public Attributes . . . . .	267
SmileyFace(_Symbol) . . . . .	267
Public Attributes . . . . .	267
StopSign(_Symbol) . . . . .	268
Public Attributes . . . . .	268
Tickbox(_Symbol) . . . . .	268
Public Attributes . . . . .	268
YesNo(_Symbol) . . . . .	268
Public Attributes . . . . .	269
_Symbol(Widget) . . . . .	269
Public Attributes . . . . .	269

---

## reportlab.graphics

### qr

```
# QRCode for Python
#
# Ported from the Javascript library by Sam Curren
# ReportLab module by German M. Bravo
#
# QRCode for Javascript
# http://d-project.googlecode.com/svn/trunk/misc/qrcode/js/qrcode.js
#
# Copyright (c) 2009 Kazuhiko Arase
#
# URL: http://www.d-project.com/
#
# Licensed under the MIT license:
# http://www.opensource.org/licenses/mit-license.php
#
# The word "QR Code" is registered trademark of
# DENSO WAVE INCORPORATED
# http://www.denso-wave.com/qrcode/faqpatent-e.html
```

### Classes

#### QrCodeWidget (PlotArea)

##### Public Attributes

**background** Handle to background object e.g. Rect(0,0,width,height).

**barBorder** Width of QR border.

**barFillColor** bar color

**barHeight** Height of bars.

**barLevel** QR Code level.

**barStrokeColor** Color of bar borders.

**barStrokeWidth** Width of bar borders.

**barWidth** Width of bars.

**debug** Used only for debugging.

**fillColor** Color of the plot area interior.

**height** Height of the chart.

**strokeColor** Color of the plot area border.

**strokeWidth** Width plot area border.

**value** the text

**width** Width of the chart.

**x** x-coord

**y** y-coord

*Example*

```
def demo(self):  
    msg = "demo() must be implemented for each Widget!"  
    raise shapes.NotImplementedError, msg
```

*Properties of Example Widget*

```
value = 'Hello World'
```

## eanbc

### Classes

#### Ean13BarcodeWidget (PlotArea)

##### Public Attributes

**background** Handle to background object e.g. Rect(0,0,width,height).

**barFillColor** bar color

**barHeight** Height of bars.

**barStrokeColor** Color of bar borders.

**barStrokeWidth** Width of bar borders.

**barWidth** Width of bars.

**debug** Used only for debugging.

**fillColor** Color of the plot area interior.

**fontName** fontName

**fontSize** font size

**height** Height of the chart.

**humanReadable** if human readable

**lquiet** left quiet zone length

**quiet** if quiet zone to be used

**rquiet** right quiet zone length

**strokeColor** Color of the plot area border.

**strokeWidth** Width plot area border.

**textColor** human readable text color

**value** the number

**width** Width of the chart.

**x** x-coord

**y** y-coord

##### Example

```
def demo(self):
    msg = "demo() must be implemented for each Widget!"
    raise shapes.NotImplementedError, msg
```

##### Properties of Example Widget

```
value = '123456789012'
```

## **Ean8BarcodeWidget ( Ean13BarcodeWidget )**

### **Public Attributes**

**background** Handle to background object e.g. Rect(0,0,width,height).

**barFillColor** bar color

**barHeight** Height of bars.

**barStrokeColor** Color of bar borders.

**barStrokeWidth** Width of bar borders.

**barWidth** Width of bars.

**debug** Used only for debugging.

**fillColor** Color of the plot area interior.

**fontName** fontName

**fontSize** font size

**height** Height of the chart.

**humanReadable** if human readable

**lquiet** left quiet zone length

**quiet** if quiet zone to be used

**rquiet** right quiet zone length

**strokeColor** Color of the plot area border.

**strokeWidth** Width plot area border.

**textColor** human readable text color

**value** the number

**width** Width of the chart.

**x** x-coord

**y** y-coord

### *Example*

```
def demo(self):
    msg = "demo() must be implemented for each Widget!"
    raise shapes.NotImplementedError, msg
```

### *Properties of Example Widget*

```
value = '1234567'
```

## UPCA ( Ean13BarcodeWidget )

### Public Attributes

**background** Handle to background object e.g. Rect(0,0,width,height).

**barFillColor** bar color

**barHeight** Height of bars.

**barStrokeColor** Color of bar borders.

**barStrokeWidth** Width of bar borders.

**barWidth** Width of bars.

**debug** Used only for debugging.

**fillColor** Color of the plot area interior.

**fontName** fontName

**fontSize** font size

**height** Height of the chart.

**humanReadable** if human readable

**lquiet** left quiet zone length

**quiet** if quiet zone to be used

**rquiet** right quiet zone length

**strokeColor** Color of the plot area border.

**strokeWidth** Width plot area border.

**textColor** human readable text color

**value** the number

**width** Width of the chart.

**x** x-coord

**y** y-coord

### *Example*

```
def demo(self):
    msg = "demo() must be implemented for each Widget!"
    raise shapes.NotImplementedError, msg
```

### *Properties of Example Widget*

```
value = '12345678901'
```

## lineplots

This module defines a very preliminary Line Plot example.

### Classes

#### AreaLinePlot(LinePlot)

we're given data in the form [(X1,Y11,...Y1M)....(Xn,Yn1,...YnM)]

#### Public Attributes

**annotations** list of callables, will be called with self, xscale, yscale.

**background** Handle to background object e.g. Rect(0,0,width,height).

**behindAxes** If true use separate line group.

**data** Data to be plotted, list of (lists of) x/y tuples.

**debug** Used only for debugging.

**fillColor** Color used for background interior of plot area.

**gridFirst** If true use draw grids before axes.

**height** Height of the chart.

**joinedLines** Display data points joined with lines if true.

**lineLabelArray** explicit array of line label values, must match size of data if present.

**lineLabelFormat** Formatting string or function used for data point labels.

**lineLabelNudge** Distance between a data point and its label.

**lineLabels** Handle to the list of data point labels.

**lines** Handle of the lines.

**reversePlotOrder** If true reverse plot order.

**strokeColor** Color used for background border of plot area.

**strokeWidth** Width plot area border.

**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**xValueAxis** Handle of the x axis.

**y** Y position of the lower-left corner of the chart.

**yValueAxis** Handle of the y axis.

#### Example

```
def demo(self):
    """Shows basic use of a line chart."""

    drawing = Drawing(400, 200)

    data = [
        ((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 = data
lp.joinedLines = 1
lp.lineLabelFormat = '%2.0f'
lp.strokeColor = colors.black

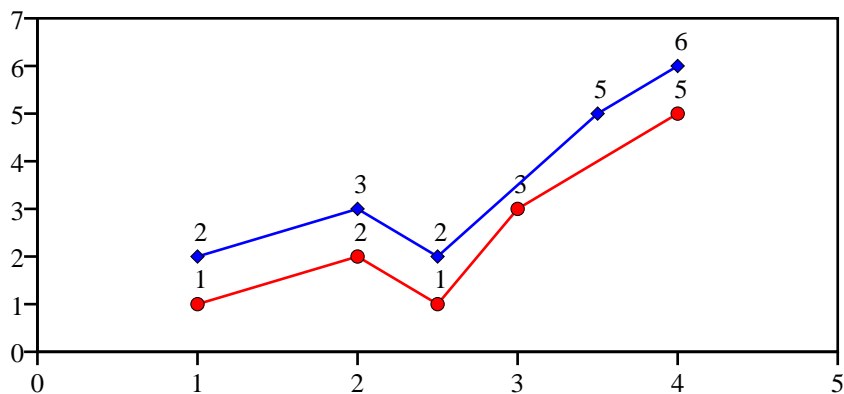
lp.lines[0].strokeColor = colors.red
lp.lines[0].symbol = makeMarker('FilledCircle')
lp.lines[1].strokeColor = colors.blue
lp.lines[1].symbol = makeMarker('FilledDiamond')

lp.xValueAxis.valueMin = 0
lp.xValueAxis.valueMax = 5
lp.xValueAxis.valueStep = 1

lp.yValueAxis.valueMin = 0
lp.yValueAxis.valueMax = 7
lp.yValueAxis.valueStep = 1

drawing.add(lp)

return drawing
```



### *Properties of Example Widget*

```
annotations = []
background = None
behindAxes = 0
data = [(1, 20, 100, 30), (2, 11, 50, 15), (3, 15, 70, 40)]
debug = 0
fillColor = None
gridFirst = 0
height = 85
joinedLines = 1
lineLabelArray = None
lineLabelFormat = None
lineLabelNudge = 10
lineLabels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1d3acf8>
lines = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1d3ac68>
reversePlotOrder = 1
strokeColor = None
strokeWidth = 1
width = 180
x = 20
xValueAxis.abf_ignore_zero = False
xValueAxis.avoidBoundFrac = None
xValueAxis.avoidBoundSpace = None
xValueAxis.drawGridLast = False
xValueAxis.forceZero = 0
xValueAxis.gridEnd = None
```

```
xValueAxis.gridStart = None
xValueAxis.gridStrokeColor = Color(0,0,0,1)
xValueAxis.gridStrokeDashArray = None
xValueAxis.gridStrokeLineCap = 0
xValueAxis.gridStrokeLineJoin = 0
xValueAxis.gridStrokeMiterLimit = 10
xValueAxis.gridStrokeWidth = 0.25
xValueAxis.hiLen = 0
xValueAxis.joinAxis = None
xValueAxis.joinAxisMode = None
xValueAxis.joinAxisPos = None
xValueAxis.keepTickLabelsInside = 0
xValueAxis.labelAxisMode = 'axis'
xValueAxis.labelTextFormat = None
xValueAxis.labelTextPostFormat = None
xValueAxis.labelTextScale = None
xValueAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1d3a950>
xValueAxis.loLen = 0
xValueAxis.maximumTicks = 7
xValueAxis.minimumTickSpacing = 10
xValueAxis.origShiftIPC = None
xValueAxis.origShiftMin = None
xValueAxis.origShiftSpecialValue = None
xValueAxis.rangeRound = 'none'
xValueAxis.requiredRange = 0
xValueAxis.reverseDirection = 0
xValueAxis.skipEndL = 'none'
xValueAxis.strokeColor = Color(0,0,0,1)
xValueAxis.strokeDashArray = None
xValueAxis.strokeLineCap = 0
xValueAxis.strokeLineJoin = 0
xValueAxis.strokeMiterLimit = 10
xValueAxis.strokeWidth = 1
xValueAxis.style = 'normal'
xValueAxis.subGridEnd = None
xValueAxis.subGridStart = None
xValueAxis.subGridStrokeColor = Color(0,0,0,1)
xValueAxis.subGridStrokeDashArray = None
xValueAxis.subGridStrokeLineCap = 0
xValueAxis.subGridStrokeLineJoin = 0
xValueAxis.subGridStrokeMiterLimit = 10
xValueAxis.subGridStrokeWidth = 0.25
xValueAxis.subTickHi = 0
xValueAxis.subTickLo = 0
xValueAxis.subTickNum = 0
xValueAxis.tickAxisMode = 'axis'
xValueAxis.tickDown = 5
xValueAxis.tickUp = 0
xValueAxis.valueMax = None
xValueAxis.valueMin = None
xValueAxis.valueStep = None
xValueAxis.visible = 1
xValueAxis.visibleAxis = 1
xValueAxis.visibleGrid = 0
xValueAxis.visibleLabels = 1
xValueAxis.visibleSubGrid = 0
xValueAxis.visibleSubTicks = 0
xValueAxis.visibleTicks = 1
xValueAxis.zrangePref = 0
y = 10
yValueAxis.abf_ignore_zero = False
yValueAxis.avoidBoundFrac = None
yValueAxis.avoidBoundSpace = None
yValueAxis.drawGridLast = False
yValueAxis.forceZero = 0
yValueAxis.gridEnd = None
yValueAxis.gridStart = None
yValueAxis.gridStrokeColor = Color(0,0,0,1)
yValueAxis.gridStrokeDashArray = None
yValueAxis.gridStrokeLineCap = 0
yValueAxis.gridStrokeLineJoin = 0
yValueAxis.gridStrokeMiterLimit = 10
yValueAxis.gridStrokeWidth = 0.25
yValueAxis.hiLen = 0
yValueAxis.joinAxis = None
yValueAxis.joinAxisMode = None
yValueAxis.joinAxisPos = None
yValueAxis.keepTickLabelsInside = 0
yValueAxis.labelAxisMode = 'axis'
```

```
yValueAxis.labelTextFormat = None
yValueAxis.labelTextPostFormat = None
yValueAxis.labelTextScale = None
yValueAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1d3abd8>
yValueAxis.loLLen = 0
yValueAxis.maximumTicks = 7
yValueAxis.minimumTickSpacing = 10
yValueAxis.origShiftIPC = None
yValueAxis.origShiftMin = None
yValueAxis.origShiftSpecialValue = None
yValueAxis.rangeRound = 'none'
yValueAxis.requiredRange = 0
yValueAxis.reverseDirection = 0
yValueAxis.skipEndL = 'none'
yValueAxis.strokeColor = Color(0,0,0,1)
yValueAxis.strokeDashArray = None
yValueAxis.strokeLineCap = 0
yValueAxis.strokeLineJoin = 0
yValueAxis.strokeMiterLimit = 10
yValueAxis.strokeWidth = 1
yValueAxis.style = 'normal'
yValueAxis.subGridEnd = None
yValueAxis.subGridStart = None
yValueAxis.subGridStrokeColor = Color(0,0,0,1)
yValueAxis.subGridStrokeDashArray = None
yValueAxis.subGridStrokeLineCap = 0
yValueAxis.subGridStrokeLineJoin = 0
yValueAxis.subGridStrokeMiterLimit = 10
yValueAxis.subGridStrokeWidth = 0.25
yValueAxis.subTickHi = 0
yValueAxis.subTickLo = 0
yValueAxis.subTickNum = 0
yValueAxis.tickAxisMode = 'axis'
yValueAxis.tickLeft = 5
yValueAxis.tickRight = 0
yValueAxis.valueMax = None
yValueAxis.valueMin = None
yValueAxis.valueStep = None
yValueAxis.visible = 1
yValueAxis.visibleAxis = 1
yValueAxis.visibleGrid = 0
yValueAxis.visibleLabels = 1
yValueAxis.visibleSubGrid = 0
yValueAxis.visibleSubTicks = 0
yValueAxis.visibleTicks = 1
yValueAxis.zrangePref = 0
```

## **GridLinePlot(SimpleTimeSeriesPlot)**

A customized version of SimpleTimeSeriesSPlot.  
It uses NormalDateXValueAxis() and AdjYValueAxis() for the X and Y axes.  
The chart has a default grid background with thin horizontal lines aligned with the tickmarks (and labels). You can change the background to be any Grid or ShadedRect, or scale the whole chart.  
If you do provide a background, you can specify the colours of the stripes with 'background.stripeColors'.

### **Public Attributes**

**annotations** list of callables, will be called with self, xscale, yscale.  
**background** Background for chart area (now Grid or ShadedRect).  
**behindAxes** If true use separate line group.  
**data** Data to be plotted, list of (lists of) x/y tuples.  
**debug** Used only for debugging.  
**fillColor** Color used for background interior of plot area.  
**gridFirst** If true use draw grids before axes.  
**height** Height of the chart.  
**joinedLines** Display data points joined with lines if true.  
**lineLabelArray** explicit array of line label values, must match size of data if present.  
**lineLabelFormat** Formatting string or function used for data point labels.  
**lineLabelNudge** Distance between a data point and its label.  
**lineLabels** Handle to the list of data point labels.  
**lines** Handle of the lines.  
**reversePlotOrder** If true reverse plot order.  
**scaleFactor** Scalefactor to apply to whole drawing.  
**strokeColor** Color used for background border of plot area.  
**strokeWidth** Width plot area border.  
**width** Width of the chart.  
**x** X position of the lower-left corner of the chart.  
**xValueAxis** Handle of the x axis.  
**y** Y position of the lower-left corner of the chart.  
**yValueAxis** Handle of the y axis.

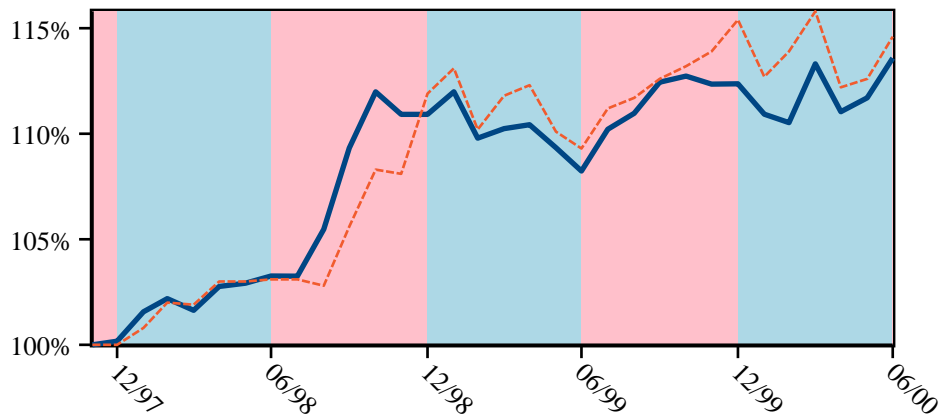
### *Example*

```
def demo(self,drawing=None):
    from reportlab.lib import colors
    if not drawing:
        drawing = Drawing(400, 200)
    lp = GridLinePlot()
    lp.x = 50
    lp.y = 50
    lp.height = 125
    lp.width = 300
    lp.data = _monthlyIndexData
    lp.joinedLines = 1
```

```

lp.strokeColor = colors.black
c0 = colors.PCMYKColor(100,65,0,30, spotName='PANTONE 288 CV', density=100)
lp.lines[0].strokeColor = c0
lp.lines[0].strokeWidth = 2
lp.lines[0].strokeDashArray = None
c1 = colors.PCMYKColor(0,79,91,0, spotName='PANTONE Wm Red CV', density=100)
lp.lines[1].strokeColor = c1
lp.lines[1].strokeWidth = 1
lp.lines[1].strokeDashArray = [3,1]
lp.xValueAxis.labels.fontSize = 10
lp.xValueAxis.labels.textAnchor = 'start'
lp.xValueAxis.labels.boxAnchor = 'w'
lp.xValueAxis.labels.angle = -45
lp.xValueAxis.labels.dx = 0
lp.xValueAxis.labels.dy = -8
lp.xValueAxis.xLabelFormat = '{mm}/{yy}'
lp.yValueAxis.labelTextFormat = '%5d%% '
lp.yValueAxis.tickLeft = 5
lp.yValueAxis.labels.fontSize = 10
lp.background = Grid()
lp.background.stripeColors = [colors.pink, colors.lightblue]
lp.background.orientation = 'vertical'
drawing.add(lp,'plot')
return drawing

```



### Properties of Example Widget

```

annotations = []
background.delta = 20
background.delta0 = 0
background.deltaSteps = []
background.fillColor = Color(1,1,1,1)
background.height = 100
background.orientation = 'horizontal'
background.stripeColors = [Color(1,0,0,1), Color(0,.501961,0,1), Color(0,0,1,1)]
background.strokeColor = Color(0,0,0,1)
background.strokeWidth = 0.5
background.useLines = 1
background.useRects = 0
background.width = 100
background.x = 0
background.y = 0
behindAxes = 0
data = [(19971202, 100.0),
        (19971231, 100.1704367),
        (19980131, 101.5639577),
        (19980228, 102.1879927),
        (19980331, 101.6337257),
        (19980430, 102.7640446),
        (19980531, 102.9198038),
        (19980630, 103.2593879),
        (19980731, 103.2516421),

```

```
(19980831, 105.4744329),
(19980930, 109.3242705),
(19981031, 111.9859291),
(19981130, 110.9184642),
(19981231, 110.9184642),
(19990131, 111.9882532),
(19990228, 109.7912614),
(19990331, 110.2418963),
(19990430, 110.4279321),
(19990531, 109.3395547),
(19990630, 108.2341748),
(19990731, 110.2129447),
(19990831, 110.9683062),
(19990930, 112.4425371),
(19991031, 112.7314032),
(19991130, 112.3509645),
(19991231, 112.3660659),
(20000131, 110.9255248),
(20000229, 110.5266306),
(20000331, 113.3116101),
(20000430, 111.0449133),
(20000531, 111.702717),
(20000630, 113.5832178)],
[(19971202, 100.0),
(19971231, 100.0),
(19980131, 100.8),
(19980228, 102.0),
(19980331, 101.9),
(19980430, 103.0),
(19980531, 103.0),
(19980630, 103.1),
(19980731, 103.1),
(19980831, 102.8),
(19980930, 105.6),
(19981031, 108.3),
(19981130, 108.1),
(19981231, 111.9),
(19990131, 113.1),
(19990228, 110.2),
(19990331, 111.8),
(19990430, 112.3),
(19990531, 110.1),
(19990630, 109.3),
(19990731, 111.2),
(19990831, 111.7),
(19990930, 112.6),
(19991031, 113.2),
(19991130, 113.9),
(19991231, 115.4),
(20000131, 112.7),
(20000229, 113.9),
(20000331, 115.8),
(20000430, 112.2),
(20000531, 112.6),
(20000630, 114.6)]]
debug = 0
fillColor = None
gridFirst = 0
height = 85
joinedLines = 1
lineLabelArray = None
lineLabelFormat = None
lineLabelNudge = 10
lineLabels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1d4c290>
lines = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1d4c200>
reversePlotOrder = 0
scaleFactor = None
strokeColor = None
strokeWidth = 1
width = 180
x = 20
xValueAxis.abf_ignore_zero = False
xValueAxis.avoidBoundFrac = None
xValueAxis.avoidBoundSpace = None
xValueAxis.bottomAxisLabelSlack = 0.1
xValueAxis.dailyFreq = 0
xValueAxis.dayOfWeekName = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
xValueAxis.drawGridLast = False
xValueAxis.forceDatesEachYear = []
```

```
xValueAxis.forceEndDate = 0
xValueAxis.forceFirstDate = 0
xValueAxis.forceZero = 0
xValueAxis.gridEnd = None
xValueAxis.gridStart = None
xValueAxis.gridStrokeColor = Color(0,0,0,1)
xValueAxis.gridStrokeDashArray = None
xValueAxis.gridStrokeLineCap = 0
xValueAxis.gridStrokeLineJoin = 0
xValueAxis.gridStrokeMiterLimit = 10
xValueAxis.gridStrokeWidth = 0.25
xValueAxis.hiLLen = 0
xValueAxis.joinAxis = None
xValueAxis.joinAxisMode = None
xValueAxis.joinAxisPos = None
xValueAxis.keepTickLabelsInside = 0
xValueAxis.labelAxisMode = 'axis'
xValueAxis.labelTextFormat = None
xValueAxis.labelTextPostFormat = None
xValueAxis.labelTextScale = None
xValueAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1d4c3f8>
xValueAxis.loLLen = 0
xValueAxis.maximumTicks = 7
xValueAxis.minimumTickSpacing = 10
xValueAxis.monthName = ['January',
                        'February',
                        'March',
                        'April',
                        'May',
                        'June',
                        'July',
                        'August',
                        'September',
                        'October',
                        'November',
                        'December']
xValueAxis.niceMonth = 1
xValueAxis.origShiftIPC = None
xValueAxis.origShiftMin = None
xValueAxis.origShiftSpecialValue = None
xValueAxis.rangeRound = 'none'
xValueAxis.requiredRange = 0
xValueAxis.reverseDirection = 0
xValueAxis.skipEndL = 'none'
xValueAxis.specialTickClear = 0
xValueAxis.specifiedTickDates = None
xValueAxis.strokeColor = Color(0,0,0,1)
xValueAxis.strokeDashArray = None
xValueAxis.strokeLineCap = 0
xValueAxis.strokeLineJoin = 0
xValueAxis.strokeMiterLimit = 10
xValueAxis.strokeWidth = 1
xValueAxis.style = 'normal'
xValueAxis.subGridEnd = None
xValueAxis.subGridStart = None
xValueAxis.subGridStrokeColor = Color(0,0,0,1)
xValueAxis.subGridStrokeDashArray = None
xValueAxis.subGridStrokeLineCap = 0
xValueAxis.subGridStrokeLineJoin = 0
xValueAxis.subGridStrokeMiterLimit = 10
xValueAxis.subGridStrokeWidth = 0.25
xValueAxis.subTickHi = 0
xValueAxis.subTickLo = 0
xValueAxis.subTickNum = 0
xValueAxis.tickAxisMode = 'axis'
xValueAxis.tickDown = 5
xValueAxis.tickUp = 0
xValueAxis.valueMax = None
xValueAxis.valueMin = None
xValueAxis.valueStep = None
xValueAxis.valueSteps = None
xValueAxis.visible = 1
xValueAxis.visibleAxis = 1
xValueAxis.visibleGrid = 0
xValueAxis.visibleLabels = 1
xValueAxis.visibleSubGrid = 0
xValueAxis.visibleSubTicks = 0
xValueAxis.visibleTicks = 1
xValueAxis.xLabelFormat = '{mm}/{yy}'
```

```
xValueAxis.zrangePref = 0
y = 10
yValueAxis.abf_ignore_zero = False
yValueAxis.avoidBoundFrac = None
yValueAxis.avoidBoundSpace = None
yValueAxis.drawGridLast = False
yValueAxis.forceZero = 0
yValueAxis.gridEnd = None
yValueAxis.gridStart = None
yValueAxis.gridStrokeColor = Color(0,0,0,1)
yValueAxis.gridStrokeDashArray = None
yValueAxis.gridStrokeLineCap = 0
yValueAxis.gridStrokeLineJoin = 0
yValueAxis.gridStrokeMiterLimit = 10
yValueAxis.gridStrokeWidth = 0.25
yValueAxis.hiLLen = 0
yValueAxis.joinAxis = None
yValueAxis.joinAxisMode = None
yValueAxis.joinAxisPos = None
yValueAxis.keepTickLabelsInside = 0
yValueAxis.labelAxisMode = 'axis'
yValueAxis.labelTextFormat = None
yValueAxis.labelTextPostFormat = None
yValueAxis.labelTextScale = None
yValueAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1d4c098>
yValueAxis.loLLen = 0
yValueAxis.maximumTicks = 7
yValueAxis.minimumTickSpacing = 10
yValueAxis.origShiftIPC = None
yValueAxis.origShiftMin = None
yValueAxis.origShiftSpecialValue = None
yValueAxis.rangeRound = 'none'
yValueAxis.requiredRange = 0
yValueAxis.reverseDirection = 0
yValueAxis.skipEndL = 'none'
yValueAxis.strokeColor = Color(0,0,0,1)
yValueAxis.strokeDashArray = None
yValueAxis.strokeLineCap = 0
yValueAxis.strokeLineJoin = 0
yValueAxis.strokeMiterLimit = 10
yValueAxis.strokeWidth = 1
yValueAxis.style = 'normal'
yValueAxis.subGridEnd = None
yValueAxis.subGridStart = None
yValueAxis.subGridStrokeColor = Color(0,0,0,1)
yValueAxis.subGridStrokeDashArray = None
yValueAxis.subGridStrokeLineCap = 0
yValueAxis.subGridStrokeLineJoin = 0
yValueAxis.subGridStrokeMiterLimit = 10
yValueAxis.subGridStrokeWidth = 0.25
yValueAxis.subTickHi = 0
yValueAxis.subTickLo = 0
yValueAxis.subTickNum = 0
yValueAxis.tickAxisMode = 'axis'
yValueAxis.tickLeft = 5
yValueAxis.tickRight = 0
yValueAxis.valueMax = None
yValueAxis.valueMin = None
yValueAxis.valueStep = None
yValueAxis.visible = 1
yValueAxis.visibleAxis = 1
yValueAxis.visibleGrid = 0
yValueAxis.visibleLabels = 1
yValueAxis.visibleSubGrid = 0
yValueAxis.visibleSubTicks = 0
yValueAxis.visibleTicks = 1
yValueAxis.zrangePref = 0
```



## LinePlot(AbstractLineChart)

Line plot with multiple lines.

Both x- and y-axis are value axis (so there are no separate X and Y versions of this class).

### Public Attributes

**annotations** list of callables, will be called with self, xscale, yscale.

**background** Handle to background object e.g. Rect(0,0,width,height).

**behindAxes** If true use separate line group.

**data** Data to be plotted, list of (lists of) x/y tuples.

**debug** Used only for debugging.

**fillColor** Color used for background interior of plot area.

**gridFirst** If true use draw grids before axes.

**height** Height of the chart.

**joinedLines** Display data points joined with lines if true.

**lineLabelArray** explicit array of line label values, must match size of data if present.

**lineLabelFormat** Formatting string or function used for data point labels.

**lineLabelNudge** Distance between a data point and its label.

**lineLabels** Handle to the list of data point labels.

**lines** Handle of the lines.

**reversePlotOrder** If true reverse plot order.

**strokeColor** Color used for background border of plot area.

**strokeWidth** Width plot area border.

**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**xValueAxis** Handle of the x axis.

**y** Y position of the lower-left corner of the chart.

**yValueAxis** Handle of the y axis.

### Example

```
def demo(self):
    """Shows basic use of a line chart."""

    drawing = Drawing(400, 200)

    data = [
        ((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 = data
```

```
lp.joinedLines = 1
lp.lineLabelFormat = '%2.0f'
lp.strokeColor = colors.black

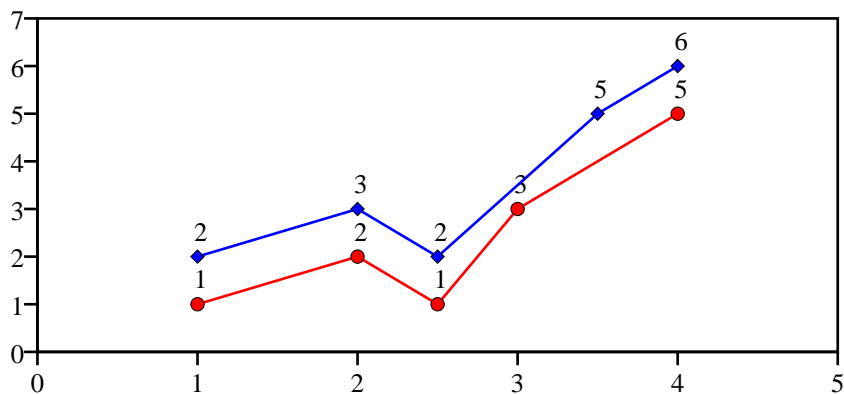
lp.lines[0].strokeColor = colors.red
lp.lines[0].symbol = makeMarker('FilledCircle')
lp.lines[1].strokeColor = colors.blue
lp.lines[1].symbol = makeMarker('FilledDiamond')

lp.xValueAxis.valueMin = 0
lp.xValueAxis.valueMax = 5
lp.xValueAxis.valueStep = 1

lp.yValueAxis.valueMin = 0
lp.yValueAxis.valueMax = 7
lp.yValueAxis.valueStep = 1

drawing.add(lp)

return drawing
```



### *Properties of Example Widget*

```
annotations = []
background = None
behindAxes = 0
data = [((1, 1), (2, 2), (2.5, 1), (3, 3), (4, 5)),
        ((1, 2), (2, 3), (2.5, 2), (3, 4), (4, 6))]
debug = 0
fillColor = None
gridFirst = 0
height = 85
joinedLines = 1
lineLabelArray = None
lineLabelFormat = None
lineLabelNudge = 10
lineLabels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1e596c8>
lines = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1e59638>
reversePlotOrder = 0
strokeColor = None
strokeWidth = 1
width = 180
x = 20
xValueAxis.abf_ignore_zero = False
xValueAxis.avoidBoundFrac = None
xValueAxis.avoidBoundSpace = None
xValueAxis.drawGridLast = False
xValueAxis.forceZero = 0
xValueAxis.gridEnd = None
xValueAxis.gridStart = None
xValueAxis.gridStrokeColor = Color(0,0,0,1)
xValueAxis.gridStrokeDashArray = None
```

```
xValueAxis.gridStrokeLineCap = 0
xValueAxis.gridStrokeLineJoin = 0
xValueAxis.gridStrokeMiterLimit = 10
xValueAxis.gridStrokeWidth = 0.25
xValueAxis.hiLen = 0
xValueAxis.joinAxis = None
xValueAxis.joinAxisMode = None
xValueAxis.joinAxisPos = None
xValueAxis.keepTickLabelsInside = 0
xValueAxis.labelAxisMode = 'axis'
xValueAxis.labelTextFormat = None
xValueAxis.labelTextPostFormat = None
xValueAxis.labelTextScale = None
xValueAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1e594d0>
xValueAxis.loLen = 0
xValueAxis.maximumTicks = 7
xValueAxis.minimumTickSpacing = 10
xValueAxis.origShiftIPC = None
xValueAxis.origShiftMin = None
xValueAxis.origShiftSpecialValue = None
xValueAxis.rangeRound = 'none'
xValueAxis.requiredRange = 0
xValueAxis.reverseDirection = 0
xValueAxis.skipEndL = 'none'
xValueAxis.strokeColor = Color(0,0,0,1)
xValueAxis.strokeDashArray = None
xValueAxis.strokeLineCap = 0
xValueAxis.strokeLineJoin = 0
xValueAxis.strokeMiterLimit = 10
xValueAxis.strokeWidth = 1
xValueAxis.style = 'normal'
xValueAxis.subGridEnd = None
xValueAxis.subGridStart = None
xValueAxis.subGridStrokeColor = Color(0,0,0,1)
xValueAxis.subGridStrokeDashArray = None
xValueAxis.subGridStrokeLineCap = 0
xValueAxis.subGridStrokeLineJoin = 0
xValueAxis.subGridStrokeMiterLimit = 10
xValueAxis.subGridStrokeWidth = 0.25
xValueAxis.subTickHi = 0
xValueAxis.subTickLo = 0
xValueAxis.subTickNum = 0
xValueAxis.tickAxisMode = 'axis'
xValueAxis.tickDown = 5
xValueAxis.tickUp = 0
xValueAxis.valueMax = None
xValueAxis.valueMin = None
xValueAxis.valueStep = None
xValueAxis.visible = 1
xValueAxis.visibleAxis = 1
xValueAxis.visibleGrid = 0
xValueAxis.visibleLabels = 1
xValueAxis.visibleSubGrid = 0
xValueAxis.visibleSubTicks = 0
xValueAxis.visibleTicks = 1
xValueAxis.zrangePref = 0
y = 10
yValueAxis.abf_ignore_zero = False
yValueAxis.avoidBoundFrac = None
yValueAxis.avoidBoundSpace = None
yValueAxis.drawGridLast = False
yValueAxis.forceZero = 0
yValueAxis.gridEnd = None
yValueAxis.gridStart = None
yValueAxis.gridStrokeColor = Color(0,0,0,1)
yValueAxis.gridStrokeDashArray = None
yValueAxis.gridStrokeLineCap = 0
yValueAxis.gridStrokeLineJoin = 0
yValueAxis.gridStrokeMiterLimit = 10
yValueAxis.gridStrokeWidth = 0.25
yValueAxis.hiLen = 0
yValueAxis.joinAxis = None
yValueAxis.joinAxisMode = None
yValueAxis.joinAxisPos = None
yValueAxis.keepTickLabelsInside = 0
yValueAxis.labelAxisMode = 'axis'
yValueAxis.labelTextFormat = None
yValueAxis.labelTextPostFormat = None
yValueAxis.labelTextScale = None
```

```
yValueAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1e595a8>
yValueAxis.loLLen = 0
yValueAxis.maximumTicks = 7
yValueAxis.minimumTickSpacing = 10
yValueAxis.origShiftIPC = None
yValueAxis.origShiftMin = None
yValueAxis.origShiftSpecialValue = None
yValueAxis.rangeRound = 'none'
yValueAxis.requiredRange = 0
yValueAxis.reverseDirection = 0
yValueAxis.skipEndL = 'none'
yValueAxis.strokeColor = Color(0,0,0,1)
yValueAxis.strokeDashArray = None
yValueAxis.strokeLineCap = 0
yValueAxis.strokeLineJoin = 0
yValueAxis.strokeMiterLimit = 10
yValueAxis.strokeWidth = 1
yValueAxis.style = 'normal'
yValueAxis.subGridEnd = None
yValueAxis.subGridStart = None
yValueAxis.subGridStrokeColor = Color(0,0,0,1)
yValueAxis.subGridStrokeDashArray = None
yValueAxis.subGridStrokeLineCap = 0
yValueAxis.subGridStrokeLineJoin = 0
yValueAxis.subGridStrokeMiterLimit = 10
yValueAxis.subGridStrokeWidth = 0.25
yValueAxis.subTickHi = 0
yValueAxis.subTickLo = 0
yValueAxis.subTickNum = 0
yValueAxis.tickAxisMode = 'axis'
yValueAxis.tickLeft = 5
yValueAxis.tickRight = 0
yValueAxis.valueMax = None
yValueAxis.valueMin = None
yValueAxis.valueStep = None
yValueAxis.visible = 1
yValueAxis.visibleAxis = 1
yValueAxis.visibleGrid = 0
yValueAxis.visibleLabels = 1
yValueAxis.visibleSubGrid = 0
yValueAxis.visibleSubTicks = 0
yValueAxis.visibleTicks = 1
yValueAxis.zrangePref = 0
```

## LinePlot3D(LinePlot)

### Public Attributes

**annotations** list of callables, will be called with self, xscale, yscale.

**background** Handle to background object e.g. Rect(0,0,width,height).

**behindAxes** If true use separate line group.

**data** Data to be plotted, list of (lists of) x/y tuples.

**debug** Used only for debugging.

**fillColor** Color used for background interior of plot area.

**gridFirst** If true use draw grids before axes.

**height** Height of the chart.

**joinedLines** Display data points joined with lines if true.

**lineLabelArray** explicit array of line label values, must match size of data if present.

**lineLabelFormat** Formatting string or function used for data point labels.

**lineLabelNudge** Distance between a data point and its label.

**lineLabels** Handle to the list of data point labels.

**lines** Handle of the lines.

**reversePlotOrder** If true reverse plot order.

**strokeColor** Color used for background border of plot area.

**strokeWidth** Width plot area border.

**theta\_x**  $dx/dz$

**theta\_y**  $dy/dz$

**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**xValueAxis** Handle of the x axis.

**y** Y position of the lower-left corner of the chart.

**yValueAxis** Handle of the y axis.

**zDepth** depth of an individual series

**zSpace** z gap around series

### Example

```
def demo(self):
    """Shows basic use of a line chart."""

    drawing = Drawing(400, 200)

    data = [
        ((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 = data
lp.joinedLines = 1
lp.lineLabelFormat = '%2.0f'
lp.strokeColor = colors.black

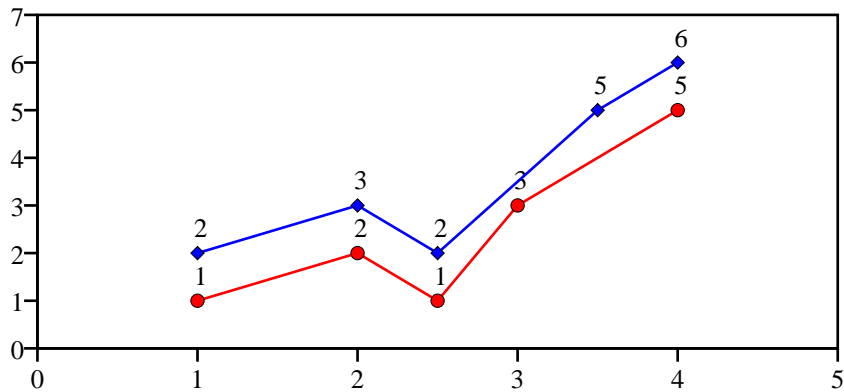
lp.lines[0].strokeColor = colors.red
lp.lines[0].symbol = makeMarker('FilledCircle')
lp.lines[1].strokeColor = colors.blue
lp.lines[1].symbol = makeMarker('FilledDiamond')

lp.xValueAxis.valueMin = 0
lp.xValueAxis.valueMax = 5
lp.xValueAxis.valueStep = 1

lp.yValueAxis.valueMin = 0
lp.yValueAxis.valueMax = 7
lp.yValueAxis.valueStep = 1

drawing.add(lp)

return drawing
```



### *Properties of Example Widget*

```
annotations = []
background = None
behindAxes = 0
data = [((1, 1), (2, 2), (2.5, 1), (3, 3), (4, 5)),
        ((1, 2), (2, 3), (2.5, 2), (3, 4), (4, 6))]
debug = 0
fillColor = None
gridFirst = 0
height = 85
joinedLines = 1
lineLabelArray = None
lineLabelFormat = None
lineLabelNudge = 10
lineLabels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1e68f80>
lines = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1e68ef0>
reversePlotOrder = 0
strokeColor = None
strokeWidth = 1
width = 180
x = 20
xValueAxis.abf_ignore_zero = False
xValueAxis.avoidBoundFrac = None
xValueAxis.avoidBoundSpace = None
xValueAxis.drawGridLast = False
xValueAxis.forceZero = 0
xValueAxis.gridEnd = None
xValueAxis.gridStart = None
xValueAxis.gridStrokeColor = Color(0,0,0,1)
```

```
xValueAxis.gridStrokeDashArray = None
xValueAxis.gridStrokeLineCap = 0
xValueAxis.gridStrokeLineJoin = 0
xValueAxis.gridStrokeMiterLimit = 10
xValueAxis.gridStrokeWidth = 0.25
xValueAxis.hiLLen = 0
xValueAxis.joinAxis = None
xValueAxis.joinAxisMode = None
xValueAxis.joinAxisPos = None
xValueAxis.keepTickLabelsInside = 0
xValueAxis.labelAxisMode = 'axis'
xValueAxis.labelTextFormat = None
xValueAxis.labelTextPostFormat = None
xValueAxis.labelTextScale = None
xValueAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1e68d88>
xValueAxis.loLLen = 0
xValueAxis.maximumTicks = 7
xValueAxis.minimumTickSpacing = 10
xValueAxis.origShiftIPC = None
xValueAxis.origShiftMin = None
xValueAxis.origShiftSpecialValue = None
xValueAxis.rangeRound = 'none'
xValueAxis.requiredRange = 0
xValueAxis.reverseDirection = 0
xValueAxis.skipEndL = 'none'
xValueAxis.strokeColor = Color(0,0,0,1)
xValueAxis.strokeDashArray = None
xValueAxis.strokeLineCap = 0
xValueAxis.strokeLineJoin = 0
xValueAxis.strokeMiterLimit = 10
xValueAxis.strokeWidth = 1
xValueAxis.style = 'normal'
xValueAxis.subGridEnd = None
xValueAxis.subGridStart = None
xValueAxis.subGridStrokeColor = Color(0,0,0,1)
xValueAxis.subGridStrokeDashArray = None
xValueAxis.subGridStrokeLineCap = 0
xValueAxis.subGridStrokeLineJoin = 0
xValueAxis.subGridStrokeMiterLimit = 10
xValueAxis.subGridStrokeWidth = 0.25
xValueAxis.subTickHi = 0
xValueAxis.subTickLo = 0
xValueAxis.subTickNum = 0
xValueAxis.tickAxisMode = 'axis'
xValueAxis.tickDown = 5
xValueAxis.tickUp = 0
xValueAxis.valueMax = None
xValueAxis.valueMin = None
xValueAxis.valueStep = None
xValueAxis.visible = 1
xValueAxis.visibleAxis = 1
xValueAxis.visibleGrid = 0
xValueAxis.visibleLabels = 1
xValueAxis.visibleSubGrid = 0
xValueAxis.visibleSubTicks = 0
xValueAxis.visibleTicks = 1
xValueAxis.zrangePref = 0
y = 10
yValueAxis.abf_ignore_zero = False
yValueAxis.avoidBoundFrac = None
yValueAxis.avoidBoundSpace = None
yValueAxis.drawGridLast = False
yValueAxis.forceZero = 0
yValueAxis.gridEnd = None
yValueAxis.gridStart = None
yValueAxis.gridStrokeColor = Color(0,0,0,1)
yValueAxis.gridStrokeDashArray = None
yValueAxis.gridStrokeLineCap = 0
yValueAxis.gridStrokeLineJoin = 0
yValueAxis.gridStrokeMiterLimit = 10
yValueAxis.gridStrokeWidth = 0.25
yValueAxis.hiLLen = 0
yValueAxis.joinAxis = None
yValueAxis.joinAxisMode = None
yValueAxis.joinAxisPos = None
yValueAxis.keepTickLabelsInside = 0
yValueAxis.labelAxisMode = 'axis'
yValueAxis.labelTextFormat = None
yValueAxis.labelTextPostFormat = None
```

```
yValueAxis.labelTextScale = None
yValueAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1e68e60>
yValueAxis.loLLen = 0
yValueAxis.maximumTicks = 7
yValueAxis.minimumTickSpacing = 10
yValueAxis.origShiftIPC = None
yValueAxis.origShiftMin = None
yValueAxis.origShiftSpecialValue = None
yValueAxis.rangeRound = 'none'
yValueAxis.requiredRange = 0
yValueAxis.reverseDirection = 0
yValueAxis.skipEndL = 'none'
yValueAxis.strokeColor = Color(0,0,0,1)
yValueAxis.strokeDashArray = None
yValueAxis.strokeLineCap = 0
yValueAxis.strokeLineJoin = 0
yValueAxis.strokeMiterLimit = 10
yValueAxis.strokeWidth = 1
yValueAxis.style = 'normal'
yValueAxis.subGridEnd = None
yValueAxis.subGridStart = None
yValueAxis.subGridStrokeColor = Color(0,0,0,1)
yValueAxis.subGridStrokeDashArray = None
yValueAxis.subGridStrokeLineCap = 0
yValueAxis.subGridStrokeLineJoin = 0
yValueAxis.subGridStrokeMiterLimit = 10
yValueAxis.subGridStrokeWidth = 0.25
yValueAxis.subTickHi = 0
yValueAxis.subTickLo = 0
yValueAxis.subTickNum = 0
yValueAxis.tickAxisMode = 'axis'
yValueAxis.tickLeft = 5
yValueAxis.tickRight = 0
yValueAxis.valueMax = None
yValueAxis.valueMin = None
yValueAxis.valueStep = None
yValueAxis.visible = 1
yValueAxis.visibleAxis = 1
yValueAxis.visibleGrid = 0
yValueAxis.visibleLabels = 1
yValueAxis.visibleSubGrid = 0
yValueAxis.visibleSubTicks = 0
yValueAxis.visibleTicks = 1
yValueAxis.zrangePref = 0
```



## ScatterPlot(LinePlot)

A scatter plot widget

### Public Attributes

**annotations** list of callables, will be called with self, xscale, yscale.

**axisStrokeWidth** Stroke width for both axes

**axisTickLengths** Length of the ticks on both axes

**background** Handle to background object e.g. Rect(0,0,width,height).

**behindAxes** If true use separate line group.

**bottomPadding** Padding at bottom of drawing

**data** Data points - a list of x/y tuples.

**debug** Used only for debugging.

**fillColor** Color used for background interior of plot area.

**gridFirst** If true use draw grids before axes.

**height** Height of the area inside the axes

**joinedLines** Display data points joined with lines if true.

**labelOffset** Space between label and Axis (or other labels)

**leftPadding** Padding on left of drawing

**lineLabelArray** explicit array of line label values, must match size of data if present.

**lineLabelFormat** Formatting string or function used for data point labels.

**lineLabelNudge** Distance between a data point and its label.

**lineLabels** Handle to the list of data point labels.

**lines** Handle of the lines.

**outerBorderColor** Color of outer border (if any)

**outerBorderOn** Is there an outer border (continuation of axes)

**reversePlotOrder** If true reverse plot order.

**rightPadding** Padding on right of drawing

**strokeColor** Color used for border of plot area.

**strokeWidth** Width plot area border.

**topPadding** Padding at top of drawing

**width** Width of the area inside the axes

**x** X position of the lower-left corner of the chart.

**xLabel** Label for the whole X-Axis

**xValueAxis** Handle of the x axis.

**y** Y position of the lower-left corner of the chart.

**yLabel** Label for the whole Y-Axis

**yValueAxis** Handle of the y axis.

*Example*

```
def demo(self,drawing=None):
    if not drawing:
        tx,ty=self._getDrawingDimensions()
        drawing = Drawing(tx,ty)
    drawing.add(self.draw())
    return drawing
```

*Properties of Example Widget*

```
annotations = []
background = None
behindAxes = 0
bottomPadding = 5
data = [(0.03, 62.73), (0.074, 54.363), (1.216, 17.964)),
        ((1.36, 11.621), (1.387, 50.011), (1.428, 68.953)),
        ((1.444, 86.888), (1.754, 35.58), (1.766, 36.05))]
debug = 0
fillColor = None
gridFirst = 0
height = 77
joinedLines = 0
leftPadding = 5
lineLabelArray = None
lineLabelFormat = '%.2f'
lineLabelNudge = 0
lineLabels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1e80320>
lines = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1e80290>
outerBorderColor = Color(0,0,0,1)
outerBorderOn = 1
reversePlotOrder = 0
rightPadding = 10
strokeColor = None
strokeWidth = 1
topPadding = 5
width = 142
x = 25.996
xLabel = 'X Lable'
xValueAxis.abf_ignore_zero = False
xValueAxis.avoidBoundFrac = None
xValueAxis.avoidBoundSpace = None
xValueAxis.drawGridLast = False
xValueAxis.forceZero = 0
xValueAxis.gridEnd = None
xValueAxis.gridStart = None
xValueAxis.gridStrokeColor = Color(0,0,0,1)
xValueAxis.gridStrokeDashArray = None
xValueAxis.gridStrokeLineCap = 0
xValueAxis.gridStrokeLineJoin = 0
xValueAxis.gridStrokeMiterLimit = 10
xValueAxis.gridStrokeWidth = 0.25
xValueAxis.hiLLen = 0
xValueAxis.joinAxis = None
xValueAxis.joinAxisMode = None
xValueAxis.joinAxisPos = None
xValueAxis.keepTickLabelsInside = 0
xValueAxis.labelAxisMode = 'axis'
xValueAxis.labelTextFormat = None
xValueAxis.labelTextPostFormat = None
xValueAxis.labelTextScale = None
xValueAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1e80128>
xValueAxis.loLLen = 0
xValueAxis.maximumTicks = 7
xValueAxis.minimumTickSpacing = 10
xValueAxis.origShiftIPC = None
xValueAxis.origShiftMin = None
xValueAxis.origShiftSpecialValue = None
xValueAxis.rangeRound = 'both'
xValueAxis.requiredRange = 0
xValueAxis.reverseDirection = 0
xValueAxis.skipEndL = 'none'
xValueAxis.strokeColor = Color(0,0,0,1)
xValueAxis.strokeDashArray = None
xValueAxis.strokeLineCap = 0
xValueAxis.strokeLineJoin = 0
```

```
xValueAxis.strokeMiterLimit = 10
xValueAxis.strokeWidth = 0.5
xValueAxis.style = 'normal'
xValueAxis.subGridEnd = None
xValueAxis.subGridStart = None
xValueAxis.subGridStrokeColor = Color(0,0,0,1)
xValueAxis.subGridStrokeDashArray = None
xValueAxis.subGridStrokeLineCap = 0
xValueAxis.subGridStrokeLineJoin = 0
xValueAxis.subGridStrokeMiterLimit = 10
xValueAxis.subGridStrokeWidth = 0.25
xValueAxis.subTickHi = 0
xValueAxis.subTickLo = 0
xValueAxis.subTickNum = 0
xValueAxis.tickAxisMode = 'axis'
xValueAxis.tickDown = 2
xValueAxis.tickUp = 0
xValueAxis.valueMax = None
xValueAxis.valueMin = None
xValueAxis.valueStep = None
xValueAxis.visible = 1
xValueAxis.visibleAxis = 1
xValueAxis.visibleGrid = 0
xValueAxis.visibleLabels = 1
xValueAxis.visibleSubGrid = 0
xValueAxis.visibleSubTicks = 0
xValueAxis.visibleTicks = 1
xValueAxis.zrangePref = 0
y = 16
yLabel = 'Y Lable'
yValueAxis.abf_ignore_zero = False
yValueAxis.avoidBoundFrac = None
yValueAxis.avoidBoundSpace = None
yValueAxis.drawGridLast = False
yValueAxis.forceZero = 0
yValueAxis.gridEnd = None
yValueAxis.gridStart = None
yValueAxis.gridStrokeColor = Color(0,0,0,1)
yValueAxis.gridStrokeDashArray = None
yValueAxis.gridStrokeLineCap = 0
yValueAxis.gridStrokeLineJoin = 0
yValueAxis.gridStrokeMiterLimit = 10
yValueAxis.gridStrokeWidth = 0.25
yValueAxis.hiLLen = 0
yValueAxis.joinAxis = None
yValueAxis.joinAxisMode = None
yValueAxis.joinAxisPos = None
yValueAxis.keepTickLabelsInside = 0
yValueAxis.labelAxisMode = 'axis'
yValueAxis.labelTextFormat = '%s'
yValueAxis.labelTextPostFormat = None
yValueAxis.labelTextScale = None
yValueAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1e80200>
yValueAxis.loLLen = 0
yValueAxis.maximumTicks = 7
yValueAxis.minimumTickSpacing = 10
yValueAxis.origShiftIPC = None
yValueAxis.origShiftMin = None
yValueAxis.origShiftSpecialValue = None
yValueAxis.rangeRound = 'both'
yValueAxis.requiredRange = 0
yValueAxis.reverseDirection = 0
yValueAxis.skipEndL = 'none'
yValueAxis.strokeColor = Color(0,0,0,1)
yValueAxis.strokeDashArray = None
yValueAxis.strokeLineCap = 0
yValueAxis.strokeLineJoin = 0
yValueAxis.strokeMiterLimit = 10
yValueAxis.strokeWidth = 0.5
yValueAxis.style = 'normal'
yValueAxis.subGridEnd = None
yValueAxis.subGridStart = None
yValueAxis.subGridStrokeColor = Color(0,0,0,1)
yValueAxis.subGridStrokeDashArray = None
yValueAxis.subGridStrokeLineCap = 0
yValueAxis.subGridStrokeLineJoin = 0
yValueAxis.subGridStrokeMiterLimit = 10
yValueAxis.subGridStrokeWidth = 0.25
yValueAxis.subTickHi = 0
```

```
yValueAxis.subTickLo = 0
yValueAxis.subTickNum = 0
yValueAxis.tickAxisMode = 'axis'
yValueAxis.tickLeft = 2
yValueAxis.tickRight = 0
yValueAxis.valueMax = None
yValueAxis.valueMin = None
yValueAxis.valueStep = None
yValueAxis.visible = 1
yValueAxis.visibleAxis = 1
yValueAxis.visibleGrid = 0
yValueAxis.visibleLabels = 1
yValueAxis.visibleSubGrid = 0
yValueAxis.visibleSubTicks = 0
yValueAxis.visibleTicks = 1
yValueAxis.zrangePref = 0
```

## **ShadedPolyFiller(Filler, ShadedPolygon)**

### **Public Attributes**

**fillColor** filler interior color

**strokeColor** filler edge color

**strokeWidth** filler edge width

### *Example*

```
def demo(self):  
    msg = "demo() must be implemented for each Widget!"  
    raise shapes.NotImplementedError, msg
```

### *Properties of Example Widget*

## SimpleTimeSeriesPlot(LinePlot)

A customized version of LinePlot.

It uses NormalDateXValueAxis() and AdjYValueAxis() for the X and Y axes.

### Public Attributes

**annotations** list of callables, will be called with self, xscale, yscale.

**background** Handle to background object e.g. Rect(0,0,width,height).

**behindAxes** If true use separate line group.

**data** Data to be plotted, list of (lists of) x/y tuples.

**debug** Used only for debugging.

**fillColor** Color used for background interior of plot area.

**gridFirst** If true use draw grids before axes.

**height** Height of the chart.

**joinedLines** Display data points joined with lines if true.

**lineLabelArray** explicit array of line label values, must match size of data if present.

**lineLabelFormat** Formatting string or function used for data point labels.

**lineLabelNudge** Distance between a data point and its label.

**lineLabels** Handle to the list of data point labels.

**lines** Handle of the lines.

**reversePlotOrder** If true reverse plot order.

**strokeColor** Color used for background border of plot area.

**strokeWidth** Width plot area border.

**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**xValueAxis** Handle of the x axis.

**y** Y position of the lower-left corner of the chart.

**yValueAxis** Handle of the y axis.

### Example

```
def demo(self):
    """Shows basic use of a line chart."""

    drawing = Drawing(400, 200)

    data = [
        ((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 = data
    lp.joinedLines = 1
    lp.lineLabelFormat = '%2.0f'
    lp.strokeColor = colors.black
```

```

lp.lines[0].strokeColor = colors.red
lp.lines[0].symbol = makeMarker('FilledCircle')
lp.lines[1].strokeColor = colors.blue
lp.lines[1].symbol = makeMarker('FilledDiamond')

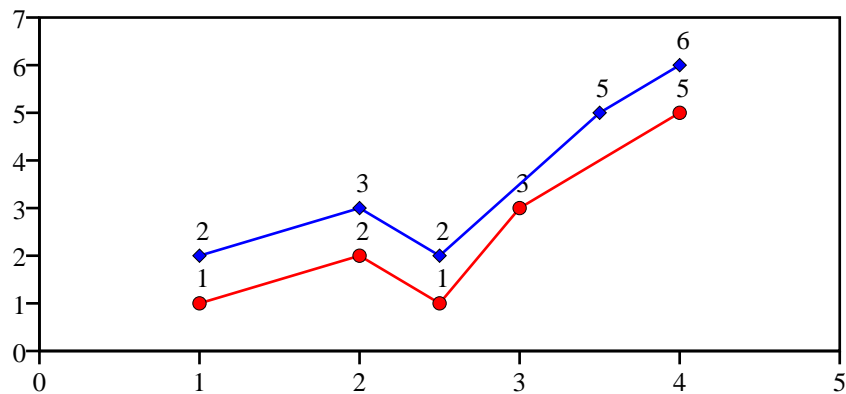
lp.xValueAxis.valueMin = 0
lp.xValueAxis.valueMax = 5
lp.xValueAxis.valueStep = 1

lp.yValueAxis.valueMin = 0
lp.yValueAxis.valueMax = 7
lp.yValueAxis.valueStep = 1

drawing.add(lp)

return drawing

```



### Properties of Example Widget

```

annotations = []
background = None
behindAxes = 0
data = [(19971202, 100.0),
        (19971231, 100.1704367),
        (19980131, 101.5639577),
        (19980228, 102.1879927),
        (19980331, 101.6337257),
        (19980430, 102.7640446),
        (19980531, 102.9198038),
        (19980630, 103.2593879),
        (19980731, 103.2516421),
        (19980831, 105.4744329),
        (19980930, 109.3242705),
        (19981031, 111.9859291),
        (19981130, 110.9184642),
        (19981231, 110.9184642),
        (19990131, 111.9882532),
        (19990228, 109.7912614),
        (19990331, 110.2418963),
        (19990430, 110.4279321),
        (19990531, 109.3395547),
        (19990630, 108.2341748),
        (19990731, 110.2129447),
        (19990831, 110.9683062),
        (19990930, 112.4425371),
        (19991031, 112.7314032),
        (19991130, 112.3509645),
        (19991231, 112.3660659),
        (20000131, 110.9255248),
        (20000229, 110.5266306),
        (20000331, 113.3116101),
        (20000430, 111.0449133),

```

```
(20000531, 111.702717),
(20000630, 113.5832178)],
[(19971202, 100.0),
(19971231, 100.0),
(19980131, 100.8),
(19980228, 102.0),
(19980331, 101.9),
(19980430, 103.0),
(19980531, 103.0),
(19980630, 103.1),
(19980731, 103.1),
(19980831, 102.8),
(19980930, 105.6),
(19981031, 108.3),
(19981130, 108.1),
(19981231, 111.9),
(19990131, 113.1),
(19990228, 110.2),
(19990331, 111.8),
(19990430, 112.3),
(19990531, 110.1),
(19990630, 109.3),
(19990731, 111.2),
(19990831, 111.7),
(19990930, 112.6),
(19991031, 113.2),
(19991130, 113.9),
(19991231, 115.4),
(20000131, 112.7),
(20000229, 113.9),
(20000331, 115.8),
(20000430, 112.2),
(20000531, 112.6),
(20000630, 114.6)]]
debug = 0
fillColor = None
gridFirst = 0
height = 85
joinedLines = 1
lineLabelArray = None
lineLabelFormat = None
lineLabelNudge = 10
lineLabels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1f24560>
lines = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1f244d0>
reversePlotOrder = 0
strokeColor = None
strokeWidth = 1
width = 180
x = 20
xValueAxis.abf_ignore_zero = False
xValueAxis.avoidBoundFrac = None
xValueAxis.avoidBoundSpace = None
xValueAxis.bottomAxisLabelSlack = 0.1
xValueAxis.dailyFreq = 0
xValueAxis.dayOfWeekName = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
xValueAxis.drawGridLast = False
xValueAxis.forceDatesEachYear = []
xValueAxis.forceEndDate = 0
xValueAxis.forceFirstDate = 0
xValueAxis.forceZero = 0
xValueAxis.gridEnd = None
xValueAxis.gridStart = None
xValueAxis.gridStrokeColor = Color(0,0,0,1)
xValueAxis.gridStrokeDashArray = None
xValueAxis.gridStrokeLineCap = 0
xValueAxis.gridStrokeLineJoin = 0
xValueAxis.gridStrokeMiterLimit = 10
xValueAxis.gridStrokeWidth = 0.25
xValueAxis.hiLen = 0
xValueAxis.joinAxis = None
xValueAxis.joinAxisMode = None
xValueAxis.joinAxisPos = None
xValueAxis.keepTickLabelsInside = 0
xValueAxis.labelAxisMode = 'axis'
xValueAxis.labelTextFormat = None
xValueAxis.labelTextPostFormat = None
xValueAxis.labelTextScale = None
xValueAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1f246c8>
xValueAxis.loLen = 0
```



```
xValueAxis.maximumTicks = 7
xValueAxis.minimumTickSpacing = 10
xValueAxis.monthName = ['January',
                        'February',
                        'March',
                        'April',
                        'May',
                        'June',
                        'July',
                        'August',
                        'September',
                        'October',
                        'November',
                        'December']

xValueAxis.niceMonth = 1
xValueAxis.origShiftIPC = None
xValueAxis.origShiftMin = None
xValueAxis.origShiftSpecialValue = None
xValueAxis.rangeRound = 'none'
xValueAxis.requiredRange = 0
xValueAxis.reverseDirection = 0
xValueAxis.skipEndL = 'none'
xValueAxis.specialTickClear = 0
xValueAxis.specifiedTickDates = None
xValueAxis.strokeColor = Color(0,0,0,1)
xValueAxis.strokeDashArray = None
xValueAxis.strokeLineCap = 0
xValueAxis.strokeLineJoin = 0
xValueAxis.strokeMiterLimit = 10
xValueAxis.strokeWidth = 1
xValueAxis.style = 'normal'
xValueAxis.subGridEnd = None
xValueAxis.subGridStart = None
xValueAxis.subGridStrokeColor = Color(0,0,0,1)
xValueAxis.subGridStrokeDashArray = None
xValueAxis.subGridStrokeLineCap = 0
xValueAxis.subGridStrokeLineJoin = 0
xValueAxis.subGridStrokeMiterLimit = 10
xValueAxis.subGridStrokeWidth = 0.25
xValueAxis.subTickHi = 0
xValueAxis.subTickLo = 0
xValueAxis.subTickNum = 0
xValueAxis.tickAxisMode = 'axis'
xValueAxis.tickDown = 5
xValueAxis.tickUp = 0
xValueAxis.valueMax = None
xValueAxis.valueMin = None
xValueAxis.valueStep = None
xValueAxis.valueSteps = None
xValueAxis.visible = 1
xValueAxis.visibleAxis = 1
xValueAxis.visibleGrid = 0
xValueAxis.visibleLabels = 1
xValueAxis.visibleSubGrid = 0
xValueAxis.visibleSubTicks = 0
xValueAxis.visibleTicks = 1
xValueAxis.xLabelFormat = '{mm}/{yy}'
xValueAxis.zrangePref = 0
y = 10
yValueAxis.abf_ignore_zero = False
yValueAxis.avoidBoundFrac = None
yValueAxis.avoidBoundSpace = None
yValueAxis.drawGridLast = False
yValueAxis.forceZero = 0
yValueAxis.gridEnd = None
yValueAxis.gridStart = None
yValueAxis.gridStrokeColor = Color(0,0,0,1)
yValueAxis.gridStrokeDashArray = None
yValueAxis.gridStrokeLineCap = 0
yValueAxis.gridStrokeLineJoin = 0
yValueAxis.gridStrokeMiterLimit = 10
yValueAxis.gridStrokeWidth = 0.25
yValueAxis.hilLen = 0
yValueAxis.joinAxis = None
yValueAxis.joinAxisMode = None
yValueAxis.joinAxisPos = None
yValueAxis.keepTickLabelsInside = 0
yValueAxis.labelAxisMode = 'axis'
yValueAxis.labelTextFormat = None
```

```
yValueAxis.labelTextPostFormat = None
yValueAxis.labelTextScale = None
yValueAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1f24368>
yValueAxis.loLLen = 0
yValueAxis.maximumTicks = 7
yValueAxis.minimumTickSpacing = 10
yValueAxis.origShiftIPC = None
yValueAxis.origShiftMin = None
yValueAxis.origShiftSpecialValue = None
yValueAxis.rangeRound = 'none'
yValueAxis.requiredRange = 0
yValueAxis.reverseDirection = 0
yValueAxis.skipEndL = 'none'
yValueAxis.strokeColor = Color(0,0,0,1)
yValueAxis.strokeDashArray = None
yValueAxis.strokeLineCap = 0
yValueAxis.strokeLineJoin = 0
yValueAxis.strokeMiterLimit = 10
yValueAxis.strokeWidth = 1
yValueAxis.style = 'normal'
yValueAxis.subGridEnd = None
yValueAxis.subGridStart = None
yValueAxis.subGridStrokeColor = Color(0,0,0,1)
yValueAxis.subGridStrokeDashArray = None
yValueAxis.subGridStrokeLineCap = 0
yValueAxis.subGridStrokeLineJoin = 0
yValueAxis.subGridStrokeMiterLimit = 10
yValueAxis.subGridStrokeWidth = 0.25
yValueAxis.subTickHi = 0
yValueAxis.subTickLo = 0
yValueAxis.subTickNum = 0
yValueAxis.tickAxisMode = 'axis'
yValueAxis.tickLeft = 5
yValueAxis.tickRight = 0
yValueAxis.valueMax = None
yValueAxis.valueMin = None
yValueAxis.valueStep = None
yValueAxis.visible = 1
yValueAxis.visibleAxis = 1
yValueAxis.visibleGrid = 0
yValueAxis.visibleLabels = 1
yValueAxis.visibleSubGrid = 0
yValueAxis.visibleSubTicks = 0
yValueAxis.visibleTicks = 1
yValueAxis.zrangePref = 0
```

## SplitLinePlot (AreaLinePlot)

### Public Attributes

**annotations** list of callables, will be called with self, xscale, yscale.

**background** Handle to background object e.g. Rect(0,0,width,height).

**behindAxes** If true use separate line group.

**data** Data to be plotted, list of (lists of) x/y tuples.

**debug** Used only for debugging.

**fillColor** Color used for background interior of plot area.

**gridFirst** If true use draw grids before axes.

**height** Height of the chart.

**joinedLines** Display data points joined with lines if true.

**lineLabelArray** explicit array of line label values, must match size of data if present.

**lineLabelFormat** Formatting string or function used for data point labels.

**lineLabelNudge** Distance between a data point and its label.

**lineLabels** Handle to the list of data point labels.

**lines** Handle of the lines.

**reversePlotOrder** If true reverse plot order.

**strokeColor** Color used for background border of plot area.

**strokeWidth** Width plot area border.

**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**xValueAxis** Handle of the x axis.

**y** Y position of the lower-left corner of the chart.

**yValueAxis** Handle of the y axis.

### Example

```
def demo(self):
    """Shows basic use of a line chart."""

    drawing = Drawing(400, 200)

    data = [
        ((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 = data
    lp.joinedLines = 1
    lp.lineLabelFormat = '%2.0f'
    lp.strokeColor = colors.black

    lp.lines[0].strokeColor = colors.red
    lp.lines[0].symbol = makeMarker('FilledCircle')
    lp.lines[1].strokeColor = colors.blue
```

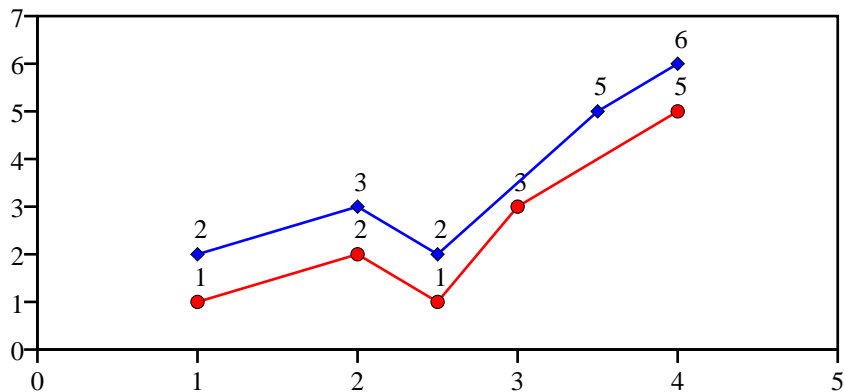
```
lp.lines[1].symbol = makeMarker('FilledDiamond')

lp.xValueAxis.valueMin = 0
lp.xValueAxis.valueMax = 5
lp.xValueAxis.valueStep = 1

lp.yValueAxis.valueMin = 0
lp.yValueAxis.valueMax = 7
lp.yValueAxis.valueStep = 1

drawing.add(lp)

return drawing
```



### *Properties of Example Widget*

```
annotations = []
background = None
behindAxes = 0
data = [(20030601, 0.95, 0.05, 0.0),
        (20030701, 0.95, 0.05, 0.0),
        (20030801, 0.95, 0.05, 0.0),
        (20030901, 0.95, 0.05, 0.0),
        (20031001, 0.95, 0.05, 0.0),
        (20031101, 0.95, 0.05, 0.0),
        (20031201, 0.95, 0.05, 0.0),
        (20040101, 0.95, 0.05, 0.0),
        (20040201, 0.95, 0.05, 0.0),
        (20040301, 0.95, 0.05, 0.0),
        (20040401, 0.95, 0.05, 0.0),
        (20040501, 0.95, 0.05, 0.0),
        (20040601, 0.95, 0.05, 0.0),
        (20040701, 0.95, 0.05, 0.0),
        (20040801, 0.95, 0.05, 0.0),
        (20040901, 0.95, 0.05, 0.0),
        (20041001, 0.95, 0.05, 0.0),
        (20041101, 0.95, 0.05, 0.0),
        (20041201, 0.95, 0.05, 0.0),
        (20050101, 0.95, 0.05, 0.0),
        (20050201, 0.95, 0.05, 0.0),
        (20050301, 0.95, 0.05, 0.0),
        (20050401, 0.95, 0.05, 0.0),
        (20050501, 0.95, 0.05, 0.0),
        (20050601, 0.95, 0.05, 0.0),
        (20050701, 0.95, 0.05, 0.0),
        (20050801, 0.95, 0.05, 0.0),
        (20050901, 0.95, 0.05, 0.0),
        (20051001, 0.95, 0.05, 0.0),
        (20051101, 0.95, 0.05, 0.0),
        (20051201, 0.95, 0.05, 0.0),
        (20060101, 0.95, 0.05, 0.0),
        (20060201, 0.95, 0.05, 0.0),
```

```
(20060301, 0.95, 0.05, 0.0),
(20060401, 0.95, 0.05, 0.0),
(20060501, 0.95, 0.05, 0.0),
(20060601, 0.95, 0.05, 0.0),
(20060701, 0.95, 0.05, 0.0),
(20060801, 0.95, 0.05, 0.0),
(20060901, 0.95, 0.05, 0.0),
(20061001, 0.95, 0.05, 0.0),
(20061101, 0.95, 0.05, 0.0),
(20061201, 0.95, 0.05, 0.0),
(20070101, 0.95, 0.05, 0.0),
(20070201, 0.95, 0.05, 0.0),
(20070301, 0.95, 0.05, 0.0),
(20070401, 0.95, 0.05, 0.0),
(20070501, 0.95, 0.05, 0.0),
(20070601, 0.95, 0.05, 0.0),
(20070701, 0.95, 0.05, 0.0),
(20070801, 0.95, 0.05, 0.0),
(20070901, 0.95, 0.05, 0.0),
(20071001, 0.95, 0.05, 0.0),
(20071101, 0.95, 0.05, 0.0),
(20071201, 0.95, 0.05, 0.0),
(20080101, 0.95, 0.05, 0.0),
(20080201, 0.95, 0.05, 0.0),
(20080301, 0.95, 0.05, 0.0),
(20080401, 0.95, 0.05, 0.0),
(20080501, 0.95, 0.05, 0.0),
(20080601, 0.95, 0.05, 0.0),
(20080701, 0.95, 0.05, 0.0),
(20080801, 0.95, 0.05, 0.0),
(20080901, 0.95, 0.05, 0.0),
(20081001, 0.95, 0.05, 0.0),
(20081101, 0.95, 0.05, 0.0),
(20081201, 0.95, 0.05, 0.0),
(20090101, 0.95, 0.05, 0.0),
(20090201, 0.91, 0.09, 0.0),
(20090301, 0.91, 0.09, 0.0),
(20090401, 0.91, 0.09, 0.0),
(20090501, 0.91, 0.09, 0.0),
(20090601, 0.91, 0.09, 0.0),
(20090701, 0.91, 0.09, 0.0),
(20090801, 0.91, 0.09, 0.0),
(20090901, 0.91, 0.09, 0.0),
(20091001, 0.91, 0.09, 0.0),
(20091101, 0.91, 0.09, 0.0),
(20091201, 0.91, 0.09, 0.0),
(20100101, 0.91, 0.09, 0.0),
(20100201, 0.81, 0.19, 0.0),
(20100301, 0.81, 0.19, 0.0),
(20100401, 0.81, 0.19, 0.0),
(20100501, 0.81, 0.19, 0.0),
(20100601, 0.81, 0.19, 0.0),
(20100701, 0.81, 0.19, 0.0),
(20100801, 0.81, 0.19, 0.0),
(20100901, 0.81, 0.19, 0.0),
(20101001, 0.81, 0.19, 0.0),
(20101101, 0.81, 0.19, 0.0),
(20101201, 0.81, 0.19, 0.0),
(20110101, 0.81, 0.19, 0.0),
(20110201, 0.72, 0.28, 0.0),
(20110301, 0.72, 0.28, 0.0),
(20110401, 0.72, 0.28, 0.0),
(20110501, 0.72, 0.28, 0.0),
(20110601, 0.72, 0.28, 0.0),
(20110701, 0.72, 0.28, 0.0),
(20110801, 0.72, 0.28, 0.0),
(20110901, 0.72, 0.28, 0.0),
(20111001, 0.72, 0.28, 0.0),
(20111101, 0.72, 0.28, 0.0),
(20111201, 0.72, 0.28, 0.0),
(20120101, 0.72, 0.28, 0.0),
(20120201, 0.53, 0.47, 0.0),
(20120301, 0.53, 0.47, 0.0),
(20120401, 0.53, 0.47, 0.0),
(20120501, 0.53, 0.47, 0.0),
(20120601, 0.53, 0.47, 0.0),
(20120701, 0.53, 0.47, 0.0),
(20120801, 0.53, 0.47, 0.0),
(20120901, 0.53, 0.47, 0.0),
```

```
(20121001, 0.53, 0.47, 0.0),
(20121101, 0.53, 0.47, 0.0),
(20121201, 0.53, 0.47, 0.0),
(20130101, 0.53, 0.47, 0.0),
(20130201, 0.44, 0.56, 0.0),
(20130301, 0.44, 0.56, 0.0),
(20130401, 0.44, 0.56, 0.0),
(20130501, 0.44, 0.56, 0.0),
(20130601, 0.44, 0.56, 0.0),
(20130701, 0.44, 0.56, 0.0),
(20130801, 0.44, 0.56, 0.0),
(20130901, 0.44, 0.56, 0.0),
(20131001, 0.44, 0.56, 0.0),
(20131101, 0.44, 0.56, 0.0),
(20131201, 0.44, 0.56, 0.0),
(20140101, 0.44, 0.56, 0.0),
(20140201, 0.36, 0.5, 0.14),
(20140301, 0.36, 0.5, 0.14),
(20140401, 0.36, 0.5, 0.14),
(20140501, 0.36, 0.5, 0.14),
(20140601, 0.36, 0.5, 0.14),
(20140701, 0.36, 0.5, 0.14),
(20140801, 0.36, 0.5, 0.14),
(20140901, 0.36, 0.5, 0.14),
(20141001, 0.36, 0.5, 0.14),
(20141101, 0.36, 0.5, 0.14),
(20141201, 0.36, 0.5, 0.14),
(20150101, 0.36, 0.5, 0.14),
(20150201, 0.3, 0.41, 0.29),
(20150301, 0.3, 0.41, 0.29),
(20150401, 0.3, 0.41, 0.29),
(20150501, 0.3, 0.41, 0.29),
(20150601, 0.3, 0.41, 0.29),
(20150701, 0.3, 0.41, 0.29),
(20150801, 0.3, 0.41, 0.29),
(20150901, 0.3, 0.41, 0.29),
(20151001, 0.3, 0.41, 0.29),
(20151101, 0.3, 0.41, 0.29),
(20151201, 0.3, 0.41, 0.29),
(20160101, 0.3, 0.41, 0.29),
(20160201, 0.26, 0.36, 0.38),
(20160301, 0.26, 0.36, 0.38),
(20160401, 0.26, 0.36, 0.38),
(20160501, 0.26, 0.36, 0.38),
(20160601, 0.26, 0.36, 0.38),
(20160701, 0.26, 0.36, 0.38),
(20160801, 0.26, 0.36, 0.38),
(20160901, 0.26, 0.36, 0.38),
(20161001, 0.26, 0.36, 0.38),
(20161101, 0.26, 0.36, 0.38),
(20161201, 0.26, 0.36, 0.38),
(20170101, 0.26, 0.36, 0.38),
(20170201, 0.2, 0.3, 0.5),
(20170301, 0.2, 0.3, 0.5),
(20170401, 0.2, 0.3, 0.5),
(20170501, 0.2, 0.3, 0.5),
(20170601, 0.2, 0.3, 0.5),
(20170701, 0.2, 0.3, 0.5),
(20170801, 0.2, 0.3, 0.5),
(20170901, 0.2, 0.3, 0.5),
(20171001, 0.2, 0.3, 0.5),
(20171101, 0.2, 0.3, 0.5),
(20171201, 0.2, 0.3, 0.5),
(20180101, 0.2, 0.3, 0.5),
(20180201, 0.13, 0.37, 0.5),
(20180301, 0.13, 0.37, 0.5),
(20180401, 0.13, 0.37, 0.5),
(20180501, 0.13, 0.37, 0.5),
(20180601, 0.13, 0.37, 0.5),
(20180701, 0.13, 0.37, 0.5),
(20180801, 0.13, 0.37, 0.5),
(20180901, 0.13, 0.37, 0.5),
(20181001, 0.13, 0.37, 0.5),
(20181101, 0.13, 0.37, 0.5),
(20181201, 0.13, 0.37, 0.5),
(20190101, 0.13, 0.37, 0.5),
(20190201, 0.1, 0.4, 0.5),
(20190301, 0.1, 0.4, 0.5),
(20190401, 0.1, 0.4, 0.5),
```

```
(20190501, 0.1, 0.4, 0.5),
(20190601, 0.1, 0.4, 0.5),
(20190701, 0.1, 0.4, 0.5),
(20190801, 0.1, 0.4, 0.5),
(20190901, 0.1, 0.4, 0.5),
(20191001, 0.1, 0.4, 0.5),
(20191101, 0.1, 0.4, 0.5),
(20191201, 0.1, 0.4, 0.5),
(20200101, 0.1, 0.4, 0.5)]
debug = 0
fillColor = None
gridFirst = 0
height = 85
joinedLines = 1
lineLabelArray = None
lineLabelFormat = None
lineLabelNudge = 10
lineLabels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1f350e0>
lines = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1f35050>
reversePlotOrder = 1
strokeColor = None
strokeWidth = 1
width = 180
x = 20
xValueAxis.abf_ignore_zero = False
xValueAxis.avoidBoundFrac = None
xValueAxis.avoidBoundSpace = None
xValueAxis.bottomAxisLabelSlack = 0.1
xValueAxis.dailyFreq = 0
xValueAxis.dayOfWeekName = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
xValueAxis.drawGridLast = False
xValueAxis.forceDatesEachYear = []
xValueAxis.forceEndDate = 0
xValueAxis.forceFirstDate = 0
xValueAxis.forceZero = 0
xValueAxis.gridEnd = None
xValueAxis.gridStart = None
xValueAxis.gridStrokeColor = Color(0,0,0,1)
xValueAxis.gridStrokeDashArray = None
xValueAxis.gridStrokeLineCap = 0
xValueAxis.gridStrokeLineJoin = 0
xValueAxis.gridStrokeMiterLimit = 10
xValueAxis.gridStrokeWidth = 0.25
xValueAxis.hiLen = 0
xValueAxis.joinAxis = None
xValueAxis.joinAxisMode = None
xValueAxis.joinAxisPos = None
xValueAxis.keepTickLabelsInside = 0
xValueAxis.labelAxisMode = 'axis'
xValueAxis.labelTextFormat = None
xValueAxis.labelTextPostFormat = None
xValueAxis.labelTextScale = None
xValueAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1f35248>
xValueAxis.loLen = 0
xValueAxis.maximumTicks = 7
xValueAxis.minimumTickSpacing = 10
xValueAxis.monthName = ['January',
                        'February',
                        'March',
                        'April',
                        'May',
                        'June',
                        'July',
                        'August',
                        'September',
                        'October',
                        'November',
                        'December']
xValueAxis.niceMonth = 1
xValueAxis.origShiftIPC = None
xValueAxis.origShiftMin = None
xValueAxis.origShiftSpecialValue = None
xValueAxis.rangeRound = 'none'
xValueAxis.requiredRange = 0
xValueAxis.reverseDirection = 0
xValueAxis.skipEndL = 'none'
xValueAxis.specialTickClear = 0
xValueAxis.specifiedTickDates = None
xValueAxis.strokeColor = Color(0,0,0,1)
```

```
xValueAxis.strokeDashArray = None
xValueAxis.strokeLineCap = 0
xValueAxis.strokeLineJoin = 0
xValueAxis.strokeMiterLimit = 10
xValueAxis.strokeWidth = 1
xValueAxis.style = 'normal'
xValueAxis.subGridEnd = None
xValueAxis.subGridStart = None
xValueAxis.subGridStrokeColor = Color(0,0,0,1)
xValueAxis.subGridStrokeDashArray = None
xValueAxis.subGridStrokeLineCap = 0
xValueAxis.subGridStrokeLineJoin = 0
xValueAxis.subGridStrokeMiterLimit = 10
xValueAxis.subGridStrokeWidth = 0.25
xValueAxis.subTickHi = 0
xValueAxis.subTickLo = 0
xValueAxis.subTickNum = 0
xValueAxis.tickAxisMode = 'axis'
xValueAxis.tickDown = 5
xValueAxis.tickUp = 0
xValueAxis.valueMax = None
xValueAxis.valueMin = None
xValueAxis.valueStep = None
xValueAxis.valueSteps = None
xValueAxis.visible = 1
xValueAxis.visibleAxis = 1
xValueAxis.visibleGrid = 0
xValueAxis.visibleLabels = 1
xValueAxis.visibleSubGrid = 0
xValueAxis.visibleSubTicks = 0
xValueAxis.visibleTicks = 1
xValueAxis.xLabelFormat = '{mm}/{yy}'
xValueAxis.zrangePref = 0
y = 10
yValueAxis.abf_ignore_zero = False
yValueAxis.avoidBoundFrac = None
yValueAxis.avoidBoundSpace = None
yValueAxis.drawGridLast = False
yValueAxis.forceZero = 0
yValueAxis.gridEnd = None
yValueAxis.gridStart = None
yValueAxis.gridStrokeColor = Color(0,0,0,1)
yValueAxis.gridStrokeDashArray = None
yValueAxis.gridStrokeLineCap = 0
yValueAxis.gridStrokeLineJoin = 0
yValueAxis.gridStrokeMiterLimit = 10
yValueAxis.gridStrokeWidth = 0.25
yValueAxis.hiLLen = 0
yValueAxis.joinAxis = None
yValueAxis.joinAxisMode = None
yValueAxis.joinAxisPos = None
yValueAxis.keepTickLabelsInside = 0
yValueAxis.labelAxisMode = 'axis'
yValueAxis.labelTextFormat = None
yValueAxis.labelTextPostFormat = None
yValueAxis.labelTextScale = None
yValueAxis.labelVOffset = 0
yValueAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1f34ea8>
yValueAxis.leftAxisOrigShiftIPC = 0
yValueAxis.leftAxisOrigShiftMin = 0
yValueAxis.leftAxisPercent = 0
yValueAxis.leftAxisSkipLL0 = 0
yValueAxis.loLLen = 0
yValueAxis.maximumTicks = 7
yValueAxis.minimumTickSpacing = 10
yValueAxis.origShiftIPC = None
yValueAxis.origShiftMin = None
yValueAxis.origShiftSpecialValue = None
yValueAxis.rangeRound = 'none'
yValueAxis.requiredRange = None
yValueAxis.reverseDirection = 0
yValueAxis.skipEndL = 'none'
yValueAxis.strokeColor = Color(0,0,0,1)
yValueAxis.strokeDashArray = None
yValueAxis.strokeLineCap = 0
yValueAxis.strokeLineJoin = 0
yValueAxis.strokeMiterLimit = 10
yValueAxis.strokeWidth = 1
yValueAxis.style = 'normal'
```



```
yValueAxis.subGridEnd = None
yValueAxis.subGridStart = None
yValueAxis.subGridStrokeColor = Color(0,0,0,1)
yValueAxis.subGridStrokeDashArray = None
yValueAxis.subGridStrokeLineCap = 0
yValueAxis.subGridStrokeLineJoin = 0
yValueAxis.subGridStrokeMiterLimit = 10
yValueAxis.subGridStrokeWidth = 0.25
yValueAxis.subTickHi = 0
yValueAxis.subTickLo = 0
yValueAxis.subTickNum = 0
yValueAxis.tickAxisMode = 'axis'
yValueAxis.tickLeft = 5
yValueAxis.tickRight = 0
yValueAxis.valueMax = None
yValueAxis.valueMin = None
yValueAxis.valueStep = None
yValueAxis.valueSteps = None
yValueAxis.visible = 1
yValueAxis.visibleAxis = 1
yValueAxis.visibleGrid = 0
yValueAxis.visibleLabels = 1
yValueAxis.visibleSubGrid = 0
yValueAxis.visibleSubTicks = 0
yValueAxis.visibleTicks = 1
yValueAxis.zrangePref = 0
```

## Functions

### `sample1a( ... )`

A line plot with non-equidistant points in x-axis.

#### *Example*

```
def sample1a():
    "A line plot with non-equidistant points in x-axis."

    drawing = Drawing(400, 200)

    data = [
        ((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 = data
    lp.joinedLines = 1
    lp.strokeColor = colors.black

    lp.lines.symbol = makeMarker('UK_Flag')

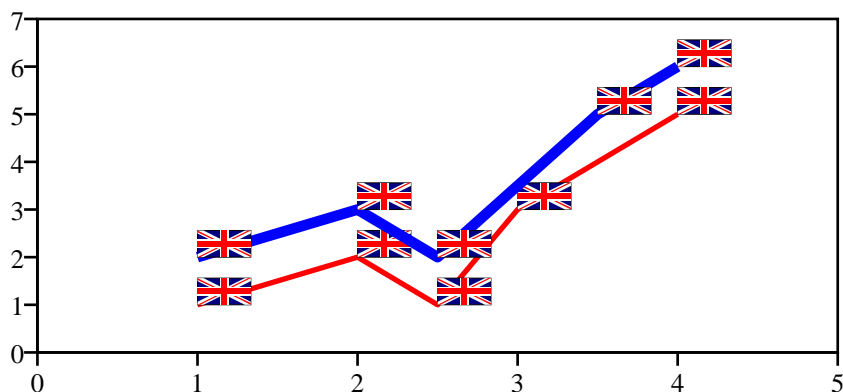
    lp.lines[0].strokeWidth = 2
    lp.lines[1].strokeWidth = 4

    lp.xValueAxis.valueMin = 0
    lp.xValueAxis.valueMax = 5
    lp.xValueAxis.valueStep = 1

    lp.yValueAxis.valueMin = 0
    lp.yValueAxis.valueMax = 7
    lp.yValueAxis.valueStep = 1

    drawing.add(lp)

    return drawing
```



**sample1b( ... )**

A line plot with non-equidistant points in x-axis.

*Example*

```
def sample1b():
    "A line plot with non-equidistant points in x-axis."

    drawing = Drawing(400, 200)

    data = [
        ((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()

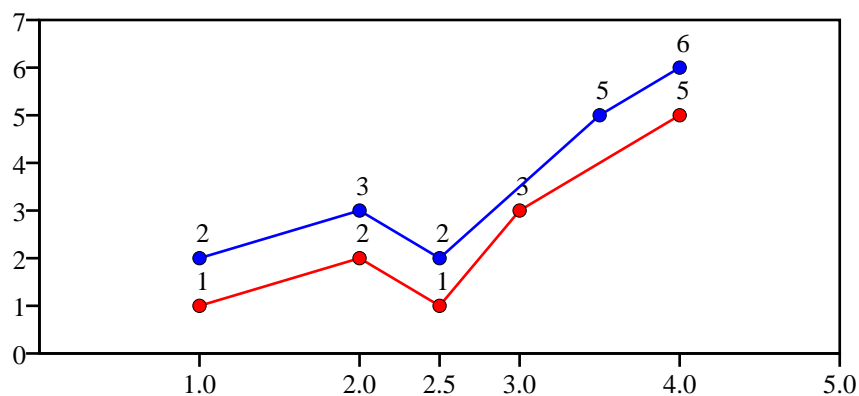
    lp.x = 50
    lp.y = 50
    lp.height = 125
    lp.width = 300
    lp.data = data
    lp.joinedLines = 1
    lp.lines.symbol = makeMarker('Circle')
    lp.lineLabelFormat = '%2.0f'
    lp.strokeColor = colors.black

    lp.xValueAxis.valueMin = 0
    lp.xValueAxis.valueMax = 5
    lp.xValueAxis.valueSteps = [1, 2, 2.5, 3, 4, 5]
    lp.xValueAxis.labelTextFormat = '%2.1f'

    lp.yValueAxis.valueMin = 0
    lp.yValueAxis.valueMax = 7
    lp.yValueAxis.valueStep = 1

    drawing.add(lp)

    return drawing
```



**sample1c( ... )**

A line plot with non-equidistant points in x-axis.

*Example*

```
def sample1c():
    "A line plot with non-equidistant points in x-axis."

    drawing = Drawing(400, 200)

    data = [
        ((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()

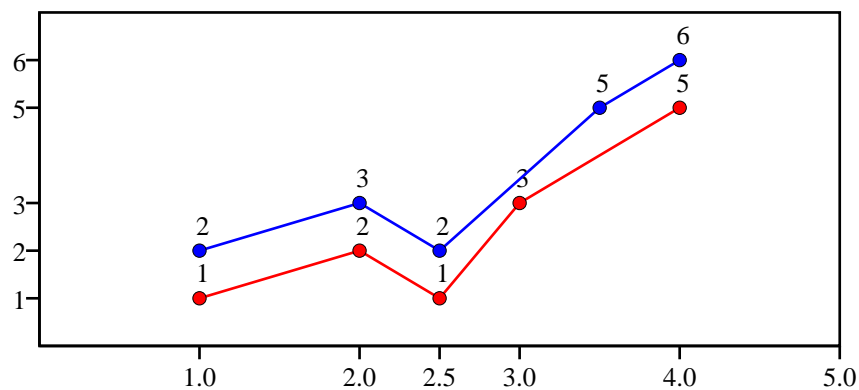
    lp.x = 50
    lp.y = 50
    lp.height = 125
    lp.width = 300
    lp.data = data
    lp.joinedLines = 1
    lp.lines[0].symbol = makeMarker('FilledCircle')
    lp.lines[1].symbol = makeMarker('Circle')
    lp.lineLabelFormat = '%2.0f'
    lp.strokeColor = colors.black

    lp.xValueAxis.valueMin = 0
    lp.xValueAxis.valueMax = 5
    lp.xValueAxis.valueSteps = [1, 2, 2.5, 3, 4, 5]
    lp.xValueAxis.labelTextFormat = '%2.1f'

    lp.yValueAxis.valueMin = 0
    lp.yValueAxis.valueMax = 7
    lp.yValueAxis.valueSteps = [1, 2, 3, 5, 6]

    drawing.add(lp)

    return drawing
```



**sample2( ... )**

A line plot with non-equidistant points in x-axis.

*Example*

```
def sample2():
    "A line plot with non-equidistant points in x-axis."

    drawing = Drawing(400, 200)

    data = [
        ('25/11/1991', 1),
        ('30/11/1991', 1.000933333),
        ('31/12/1991', 1.0062),
        ('31/01/1992', 1.0112),
        ('29/02/1992', 1.0158),
        ('31/03/1992', 1.020733333),
        ('30/04/1992', 1.026133333),
        ('31/05/1992', 1.030266667),
        ('30/06/1992', 1.034466667),
        ('31/07/1992', 1.038733333),
        ('31/08/1992', 1.0422),
        ('30/09/1992', 1.045533333),
        ('31/10/1992', 1.049866667),
        ('30/11/1992', 1.054733333),
        ('31/12/1992', 1.061),
    ],
    ]

    data[0] = preprocessData(data[0])

    lp = LinePlot()

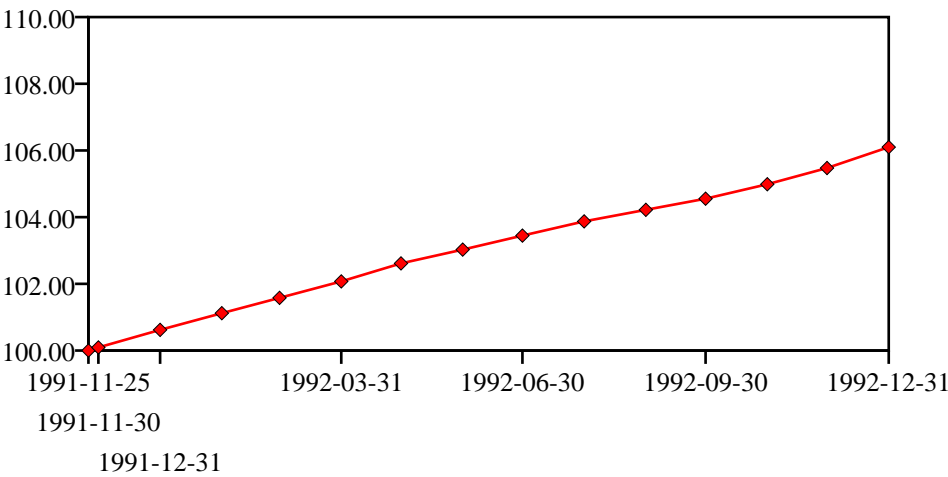
    lp.x = 50
    lp.y = 50
    lp.height = 125
    lp.width = 300
    lp.data = data
    lp.joinedLines = 1
    lp.lines.symbol = makeMarker('FilledDiamond')
    lp.strokeColor = colors.black

    start = mktime(mkTimeTuple('25/11/1991'))
    t0 = mktime(mkTimeTuple('30/11/1991'))
    t1 = mktime(mkTimeTuple('31/12/1991'))
    t2 = mktime(mkTimeTuple('31/03/1992'))
    t3 = mktime(mkTimeTuple('30/06/1992'))
    t4 = mktime(mkTimeTuple('30/09/1992'))
    end = mktime(mkTimeTuple('31/12/1992'))
    lp.xValueAxis.valueMin = start
    lp.xValueAxis.valueMax = end
    lp.xValueAxis.valueSteps = [start, t0, t1, t2, t3, t4, end]
    lp.xValueAxis.labelTextFormat = seconds2str
    lp.xValueAxis.labels[1].dy = -20
    lp.xValueAxis.labels[2].dy = -35

    lp.yValueAxis.labelTextFormat = '%4.2f'
    lp.yValueAxis.valueMin = 100
    lp.yValueAxis.valueMax = 110
    lp.yValueAxis.valueStep = 2

    drawing.add(lp)

    return drawing
```



## textlabels

#Copyright ReportLab Europe Ltd. 2000-2012

#see license.txt for license details

#history <http://www.reportlab.co.uk/cgi-bin/viewcvs.cgi/public/reportlab/trunk/reportlab/graphics/charts/textlabels.py>

## Classes

### BarChartLabel ( PMVLabel )

An extended Label allowing for nudging, lines visibility etc

### Public Attributes

**angle** angle of label: default (0), 90 is vertical, 180 is upside down, etc

**bottomPadding** padding at bottom of box

**boxAnchor** anchoring point of the label

**boxFillColor** the filling color of the box

**boxStrokeColor** border color of the box

**boxStrokeWidth** border width

**boxTarget** one of ('normal','anti','lo','hi')

**customDrawChanger** An instance of CustomDrawChanger to modify the behavior at draw time

**dx** delta x - offset

**dy** delta y - offset

**fillColor** label text color

**fixedEnd** None or fixed draw ends +/-

**fixedStart** None or fixed draw starts +/-

**fontName** the name of the font used

**fontSize** the size of the font

**height** the height of the text

**leading**

**leftPadding** padding at left of box

**lineStrokeColor** Color for a drawn line

**lineStrokeWidth** Non-zero for a drawn line

**maxWidth** maximum width the label can grow to

**nudge** Non-zero sign dependent nudge

**rightPadding** padding at right of box

**strokeColor** label text border color

**strokeWidth** label text border width

**text** the actual text to display

**textAnchor** the anchoring point of the text inside the label

**topPadding** padding at top of box

**useAscentDescent** If True then the font's Ascent Descent will be used to compute default heights and baseline.

**visible** True if the label is to be drawn

**width** the width of the label

**x**

**y**

#### *Example*

```
def demo(self):
    """This shows a label positioned with its top right corner
    at the top centre of the drawing, and rotated 45 degrees."""

    d = Drawing(200, 100)

    # mark the origin of the label
    d.add(Circle(100,90, 5, fillColor=colors.green))

    lab = Label()
    lab.setOrigin(100,90)
    lab.boxAnchor = 'ne'
    lab.angle = 45
    lab.dx = 0
    lab.dy = -20
    lab.boxStrokeColor = colors.green
    lab.setText('Another\nMulti-Line\nString')
    d.add(lab)

    return d
```

#### *Properties of Example Widget*

```
angle = 0
bottomPadding = 0
boxAnchor = 'c'
boxFillColor = None
boxStrokeColor = None
boxStrokeWidth = 0.5
boxTarget = 'normal'
dx = 0
dy = 0
fillColor = Color(0,0,0,1)
fixedEnd = None
fixedStart = None
fontName = 'Times-Roman'
fontSize = 10
height = None
leading = None
leftPadding = 0
lineStrokeColor = None
lineStrokeWidth = 0
maxWidth = None
nudge = 0
rightPadding = 0
strokeColor = None
strokeWidth = 0.1
textAnchor = 'start'
topPadding = 0
useAscentDescent = False
visible = 1
width = None
x = 0
y = 0
```



## Label (Widget)

A text label to attach to something else, such as a chart axis.

This allows you to specify an offset, angle and many anchor properties relative to the label's origin. It allows, for example, angled multiline axis labels.

### Public Attributes

**angle** angle of label: default (0), 90 is vertical, 180 is upside down, etc

**bottomPadding** padding at bottom of box

**boxAnchor** anchoring point of the label

**boxFillColor** the filling color of the box

**boxStrokeColor** border color of the box

**boxStrokeWidth** border width

**boxTarget** one of ('normal','anti','lo','hi')

**customDrawChanger** An instance of CustomDrawChanger to modify the behavior at draw time

**dx** delta x - offset

**dy** delta y - offset

**fillColor** label text color

**fontName** the name of the font used

**fontSize** the size of the font

**height** the height of the text

**leading**

**leftPadding** padding at left of box

**maxWidth** maximum width the label can grow to

**rightPadding** padding at right of box

**strokeColor** label text border color

**strokeWidth** label text border width

**text** the actual text to display

**textAnchor** the anchoring point of the text inside the label

**topPadding** padding at top of box

**useAscentDescent** If True then the font's Ascent Descent will be used to compute default heights and baseline.

**visible** True if the label is to be drawn

**width** the width of the label

**x**

**y**

### Example

```
def demo(self):
```

```
"""This shows a label positioned with its top right corner
at the top centre of the drawing, and rotated 45 degrees."""

d = Drawing(200, 100)

# mark the origin of the label
d.add(Circle(100,90, 5, fillColor=colors.green))

lab = Label()
lab.setOrigin(100,90)
lab.boxAnchor = 'ne'
lab.angle = 45
lab.dx = 0
lab.dy = -20
lab.boxStrokeColor = colors.green
lab.setText('Another\nMulti-Line\nString')
d.add(lab)

return d
```

### *Properties of Example Widget*

```
angle = 0
bottomPadding = 0
boxAnchor = 'c'
boxFillColor = None
boxStrokeColor = None
boxStrokeWidth = 0.5
boxTarget = 'normal'
dx = 0
dy = 0
fillColor = Color(0,0,0,1)
fontName = 'Times-Roman'
fontSize = 10
height = None
leading = None
leftPadding = 0
maxWidth = None
rightPadding = 0
strokeColor = None
strokeWidth = 0.1
textAnchor = 'start'
topPadding = 0
useAscentDescent = False
visible = 1
width = None
x = 0
y = 0
```

**NA\_Label ( BarChartLabel )**

An extended Label allowing for nudging, lines visibility etc

**Public Attributes**

**angle** angle of label: default (0), 90 is vertical, 180 is upside down, etc

**bottomPadding** padding at bottom of box

**boxAnchor** anchoring point of the label

**boxFillColor** the filling color of the box

**boxStrokeColor** border color of the box

**boxStrokeWidth** border width

**boxTarget** one of ('normal','anti','lo','hi')

**customDrawChanger** An instance of CustomDrawChanger to modify the behavior at draw time

**dx** delta x - offset

**dy** delta y - offset

**fillColor** label text color

**fixedEnd** None or fixed draw ends +/-

**fixedStart** None or fixed draw starts +/-

**fontName** the name of the font used

**fontSize** the size of the font

**height** the height of the text

**leading**

**leftPadding** padding at left of box

**lineStrokeColor** Color for a drawn line

**lineStrokeWidth** Non-zero for a drawn line

**maxWidth** maximum width the label can grow to

**nudge** Non-zero sign dependent nudge

**rightPadding** padding at right of box

**strokeColor** label text border color

**strokeWidth** label text border width

**text** Text to be used for N/A values

**textAnchor** the anchoring point of the text inside the label

**topPadding** padding at top of box

**useAscentDescent** If True then the font's Ascent Descent will be used to compute default heights and baseline.

**visible** True if the label is to be drawn

**width** the width of the label

**x**

**y**

*Example*

```
def demo(self):
    """This shows a label positioned with its top right corner
    at the top centre of the drawing, and rotated 45 degrees."""

    d = Drawing(200, 100)

    # mark the origin of the label
    d.add(Circle(100,90, 5, fillColor=colors.green))

    lab = Label()
    lab.setOrigin(100,90)
    lab.boxAnchor = 'ne'
    lab.angle = 45
    lab.dx = 0
    lab.dy = -20
    lab.boxStrokeColor = colors.green
    lab.setText('Another\nMulti-Line\nString')
    d.add(lab)

    return d
```

*Properties of Example Widget*

```
angle = 0
bottomPadding = 0
boxAnchor = 'c'
boxFillColor = None
boxStrokeColor = None
boxStrokeWidth = 0.5
boxTarget = 'normal'
dx = 0
dy = 0
fillColor = Color(0,0,0,1)
fixedEnd = None
fixedStart = None
fontName = 'Times-Roman'
fontSize = 10
height = None
leading = None
leftPadding = 0
lineStrokeColor = None
lineStrokeWidth = 0
maxWidth = None
nudge = 0
rightPadding = 0
strokeColor = None
strokeWidth = 0.1
text = 'n/a'
textAnchor = 'start'
topPadding = 0
useAscentDescent = False
visible = 1
width = None
x = 0
y = 0
```

## PMVLabel (Label)

### Public Attributes

**angle** angle of label: default (0), 90 is vertical, 180 is upside down, etc

**bottomPadding** padding at bottom of box

**boxAnchor** anchoring point of the label

**boxFillColor** the filling color of the box

**boxStrokeColor** border color of the box

**boxStrokeWidth** border width

**boxTarget** one of ('normal','anti','lo','hi')

**customDrawChanger** An instance of CustomDrawChanger to modify the behavior at draw time

**dx** delta x - offset

**dy** delta y - offset

**fillColor** label text color

**fontName** the name of the font used

**fontSize** the size of the font

**height** the height of the text

**leading**

**leftPadding** padding at left of box

**maxWidth** maximum width the label can grow to

**rightPadding** padding at right of box

**strokeColor** label text border color

**strokeWidth** label text border width

**text** the actual text to display

**textAnchor** the anchoring point of the text inside the label

**topPadding** padding at top of box

**useAscentDescent** If True then the font's Ascent Descent will be used to compute default heights and baseline.

**visible** True if the label is to be drawn

**width** the width of the label

**x**

**y**

### Example

```
def demo(self):
    """This shows a label positioned with its top right corner
    at the top centre of the drawing, and rotated 45 degrees."""

    d = Drawing(200, 100)

    # mark the origin of the label
    d.add(Circle(100,90, 5, fillColor=colors.green))
    lab = Label()
```

```
lab.setOrigin(100,90)
lab.boxAnchor = 'ne'
lab.angle = 45
lab.dx = 0
lab.dy = -20
lab.boxStrokeColor = colors.green
lab.setText('Another\nMulti-Line\nString')
d.add(lab)

return d
```

### *Properties of Example Widget*

```
angle = 0
bottomPadding = 0
boxAnchor = 'c'
boxFillColor = None
boxStrokeColor = None
boxStrokeWidth = 0.5
boxTarget = 'normal'
dx = 0
dy = 0
fillColor = Color(0,0,0,1)
fontName = 'Times-Roman'
fontSize = 10
height = None
leading = None
leftPadding = 0
maxWidth = None
rightPadding = 0
strokeColor = None
strokeWidth = 0.1
textAnchor = 'start'
topPadding = 0
useAscentDescent = False
visible = 1
width = None
x = 0
y = 0
```

## spider

### Spider Chart

Normal use shows variation of 5-10 parameters against some 'norm' or target. When there is more than one series, place the series with the largest numbers first, as it will be overdrawn by each successive one.

## Classes

### SpiderChart (PlotArea)

#### Public Attributes

**background** Handle to background object e.g. Rect(0,0,width,height).

**data** Data to be plotted, list of (lists of) numbers.

**debug** Used only for debugging.

**direction** 'clockwise' or 'anticlockwise'

**fillColor** Color of the plot area interior.

**height** Height of the chart.

**labels** optional list of labels to use for each data point

**spokeLabels** collection of spoke label descriptor objects

**spokes** collection of spoke descriptor objects

**startAngle** angle of first slice; like the compass, 0 is due North

**strandLabels** collection of strand label descriptor objects

**strands** collection of strand descriptor objects

**strokeColor** Color of the plot area border.

**strokeWidth** Width plot area border.

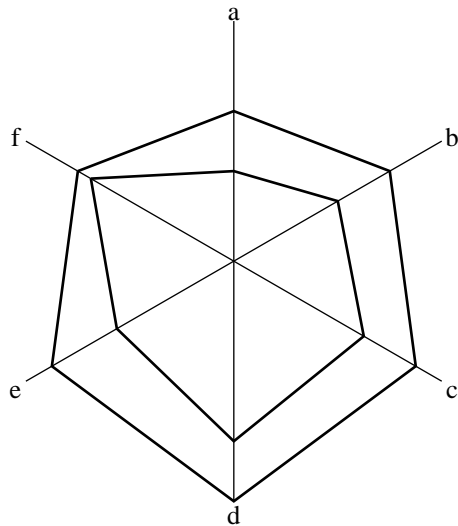
**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**y** Y position of the lower-left corner of the chart.

#### Example

```
def demo(self):
    d = Drawing(200, 200)
    d.add(SpiderChart())
    return d
```



### *Properties of Example Widget*

```
background = None
data = [[10, 12, 14, 16, 14, 12], [6, 8, 10, 12, 9, 11]]
debug = 0
direction = 'clockwise'
fillColor = None
height = 180
labels = ['a', 'b', 'c', 'd', 'e', 'f']
spokeLabels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x21c96c8>
spokes = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x21c9638>
startAngle = 90
strandLabels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x21c9758>
strands = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x21c95a8>
strokeColor = None
strokeWidth = 1
width = 180
x = 10
y = 10
```



## **SpokeLabel (WedgeLabel)**

### **Public Attributes**

**angle** angle of label: default (0), 90 is vertical, 180 is upside down, etc

**bottomPadding** padding at bottom of box

**boxAnchor** anchoring point of the label

**boxFillColor** the filling color of the box

**boxStrokeColor** border color of the box

**boxStrokeWidth** border width

**boxTarget** one of ('normal','anti','lo','hi')

**customDrawChanger** An instance of CustomDrawChanger to modify the behavior at draw time

**dx** delta x - offset

**dy** delta y - offset

**fillColor** label text color

**fontName** the name of the font used

**fontSize** the size of the font

**height** the height of the text

**leading**

**leftPadding** padding at left of box

**maxWidth** maximum width the label can grow to

**rightPadding** padding at right of box

**strokeColor** label text border color

**strokeWidth** label text border width

**text** the actual text to display

**textAnchor** the anchoring point of the text inside the label

**topPadding** padding at top of box

**useAscentDescent** If True then the font's Ascent Descent will be used to compute default heights and baseline.

**visible** True if the label is to be drawn

**width** the width of the label

**x**

**y**

### *Example*

```
def demo(self):
    """This shows a label positioned with its top right corner
    at the top centre of the drawing, and rotated 45 degrees."""

    d = Drawing(200, 100)

    # mark the origin of the label
    d.add(Circle(100,90, 5, fillColor=colors.green))
    lab = Label()
```

```
lab.setOrigin(100,90)
lab.boxAnchor = 'ne'
lab.angle = 45
lab.dx = 0
lab.dy = -20
lab.boxStrokeColor = colors.green
lab.setText('Another\nMulti-Line\nString')
d.add(lab)

return d
```

### *Properties of Example Widget*

```
angle = 0
bottomPadding = 0
boxAnchor = 'c'
boxFillColor = None
boxStrokeColor = None
boxStrokeWidth = 0.5
boxTarget = 'normal'
dx = 0
dy = 0
fillColor = Color(0,0,0,1)
fontName = 'Times-Roman'
fontSize = 10
height = None
leading = None
leftPadding = 0
maxWidth = None
rightPadding = 0
strokeColor = None
strokeWidth = 0.1
textAnchor = 'start'
topPadding = 0
useAscentDescent = False
visible = 1
width = None
x = 0
y = 0
```

## **StrandLabel ( SpokeLabel )**

### **Public Attributes**

**angle** angle of label: default (0), 90 is vertical, 180 is upside down, etc

**bottomPadding** padding at bottom of box

**boxAnchor** anchoring point of the label

**boxFillColor** the filling color of the box

**boxStrokeColor** border color of the box

**boxStrokeWidth** border width

**boxTarget** one of ('normal','anti','lo','hi')

**customDrawChanger** An instance of CustomDrawChanger to modify the behavior at draw time

**dR** radial shift for label

**dx** delta x - offset

**dy** delta y - offset

**fillColor** label text color

**fontName** the name of the font used

**fontSize** the size of the font

**format** Format for the label

**height** the height of the text

**leading**

**leftPadding** padding at left of box

**maxWidth** maximum width the label can grow to

**rightPadding** padding at right of box

**strokeColor** label text border color

**strokeWidth** label text border width

**text** the actual text to display

**textAnchor** the anchoring point of the text inside the label

**topPadding** padding at top of box

**useAscentDescent** If True then the font's Ascent Descent will be used to compute default heights and baseline.

**visible** True if the label is to be drawn

**width** the width of the label

**x**

**y**

### *Example*

```
def demo(self):
    """This shows a label positioned with its top right corner
    at the top centre of the drawing, and rotated 45 degrees."""
    d = Drawing(200, 100)
```

```
# mark the origin of the label
d.add(Circle(100,90, 5, fillColor=colors.green))

lab = Label()
lab.setOrigin(100,90)
lab.boxAnchor = 'ne'
lab.angle = 45
lab.dx = 0
lab.dy = -20
lab.boxStrokeColor = colors.green
lab.setText('Another\nMulti-Line\nString')
d.add(lab)

return d
```

### *Properties of Example Widget*

```
angle = 0
bottomPadding = 0
boxAnchor = 'c'
boxFillColor = None
boxStrokeColor = None
boxStrokeWidth = 0.5
boxTarget = 'normal'
dR = 0
dx = 0
dy = 0
fillColor = Color(0,0,0,1)
fontName = 'Times-Roman'
fontSize = 10
format = ''
height = None
leading = None
leftPadding = 0
maxWidth = None
rightPadding = 0
strokeColor = None
strokeWidth = 0.1
textAnchor = 'start'
topPadding = 0
useAscentDescent = False
visible = 1
width = None
x = 0
y = 0
```

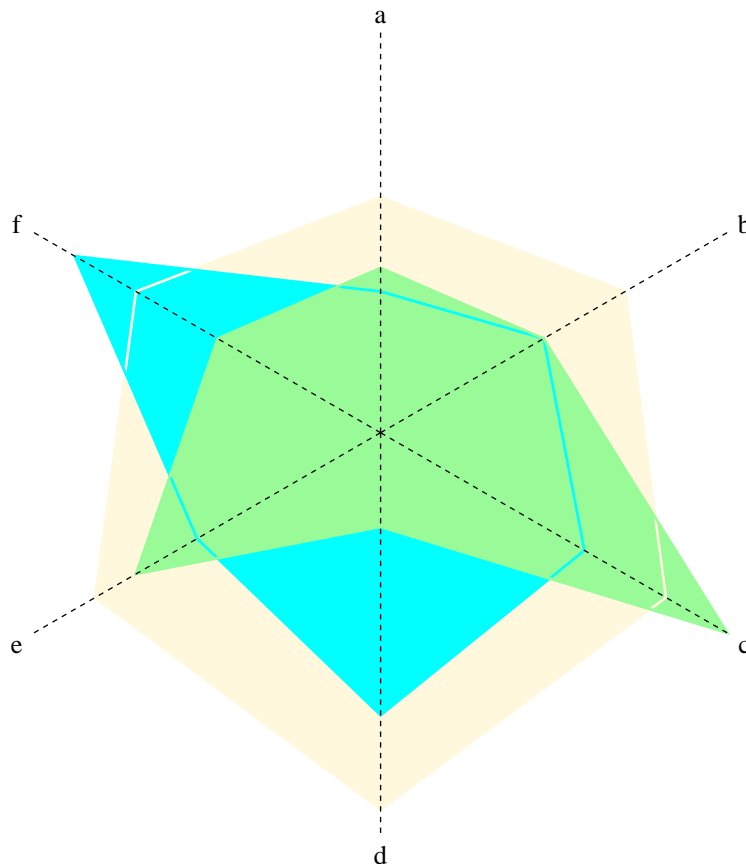
## Functions

### `sample1( ... )`

Make a simple spider chart

#### *Example*

```
def sample1():
    "Make a simple spider chart"
    d = Drawing(400, 400)
    sp = SpiderChart()
    sp.x = 50
    sp.y = 50
    sp.width = 300
    sp.height = 300
    sp.data = [[10,12,14,16,14,12], [6,8,10,12,9,15],[7,8,17,4,12,8]]
    sp.labels = ['a','b','c','d','e','f']
    sp.strands[0].strokeColor = colors.cornsilk
    sp.strands[1].strokeColor = colors.cyan
    sp.strands[2].strokeColor = colors.palegreen
    sp.strands[0].fillColor = colors.cornsilk
    sp.strands[1].fillColor = colors.cyan
    sp.strands[2].fillColor = colors.palegreen
    sp.spokes.strokeDashArray = (2,2)
    d.add(sp)
    return d
```



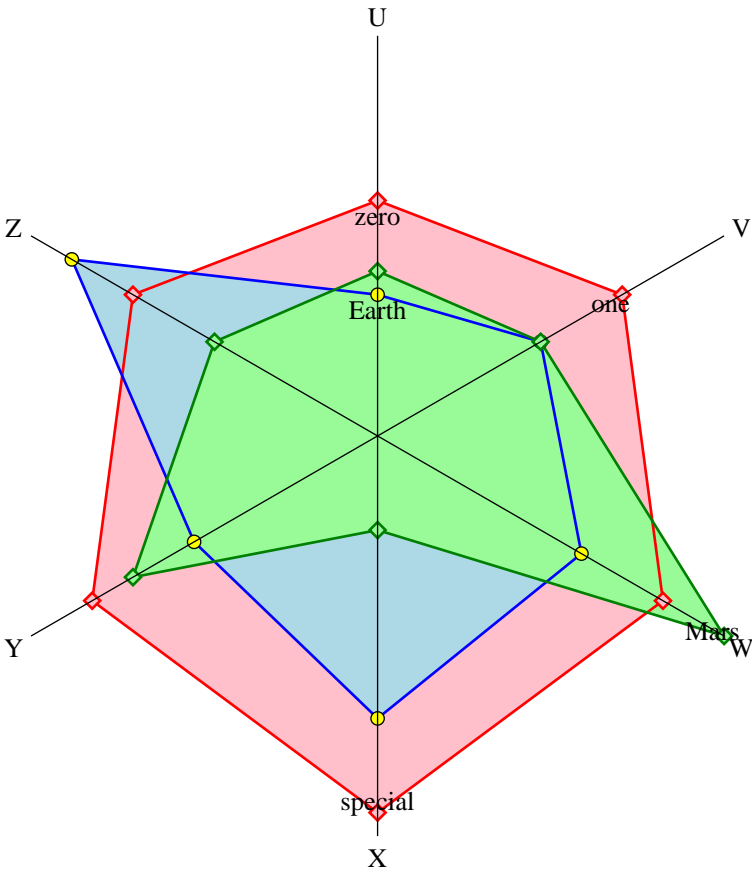


**sample2( ... )**

Make a spider chart with markers, but no fill

*Example*

```
def sample2():
    "Make a spider chart with markers, but no fill"
    d = Drawing(400, 400)
    sp = SpiderChart()
    sp.x = 50
    sp.y = 50
    sp.width = 300
    sp.height = 300
    sp.data = [[10,12,14,16,14,12], [6,8,10,12,9,15],[7,8,17,4,12,8]]
    sp.labels = ['U','V','W','X','Y','Z']
    sp.strands.strokeWidth = 1
    sp.strands[0].fillColor = colors.pink
    sp.strands[1].fillColor = colors.lightblue
    sp.strands[2].fillColor = colors.palegreen
    sp.strands[0].strokeColor = colors.red
    sp.strands[1].strokeColor = colors.blue
    sp.strands[2].strokeColor = colors.green
    sp.strands.symbol = "FilledDiamond"
    sp.strands[1].symbol = makeMarker("Circle")
    sp.strands[1].symbol.strokeWidth = 0.5
    sp.strands[1].symbol.fillColor = colors.yellow
    sp.strands.symbolSize = 6
    sp.strandLabels[0,3]._text = 'special'
    sp.strandLabels[0,1]._text = 'one'
    sp.strandLabels[0,0]._text = 'zero'
    sp.strandLabels[1,0]._text = 'Earth'
    sp.strandLabels[2,2]._text = 'Mars'
    sp.strandLabels.format = 'values'
    sp.strandLabels.dR = -5
    d.add(sp)
    return d
```





## dotbox

### Classes

#### DotBox(Widget)

Returns a dotbox widget.

### Public Attributes

**dotColor** Colour of the circle on the box

**dotDiameter** Diameter of the circle used for the 'dot'

**dotXPosition** X Position of the circle

**dotYPosition** Y Position of the circle

**gridColor** Colour for the box and gridding

**gridDivWidth** Width of each 'box'

**labelFontName** Name of font used for the labels

**labelFontSize** Size of font used for the labels

**labelOffset** Space between label text and grid edge

**strokeWidth** Width of the grid and dot outline

**x** X Position of dotbox

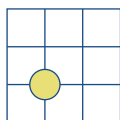
**xlabels** List of text labels for boxes on left hand side

**y** Y Position of dotbox

**ylabels** Text label for second box on left hand side

#### Example

```
def demo(self,drawing=None):
    if not drawing:
        tx,ty=self._getDrawingDimensions()
        drawing = Drawing(tx,ty)
        drawing.add(self.draw())
    return drawing
```



#### Properties of Example Widget

```
dotColor = Color(.909804,.878431,.466667,1)
dotDiameter = 11.338582677165356
dotXPosition = 1
dotYPosition = 1
gridColor = Color(.098039,.301961,.529412,1)
gridDivWidth = 14.173228346456693
labelFontName = 'Helvetica'
labelFontSize = 6
```

```
labelOffset = 5
strokeWidth = 0.5
x = 30
xlabels = ['Value', 'Blend', 'Growth']
y = 5
ylabels = ['Small', 'Medium', 'Large']
```

## axes

Collection of axes for charts.

The current collection comprises axes for charts using cartesian coordinate systems. All axes might have tick marks and labels. There are two dichotomies for axes: one of X and Y flavours and another of category and value flavours.

Category axes have an ordering but no metric. They are divided into a number of equal-sized buckets. Their tick marks or labels, if available, go BETWEEN the buckets, and the labels are placed below to/left of the X/Y-axis, respectively.

Value axes have an ordering AND metric. They correspond to a numeric quantity. Value axis have a real number quantity associated with it. The chart tells it where to go.

The most basic axis divides the number line into equal spaces and has tickmarks and labels associated with each; later we will add variants where you can specify the sampling interval.

The charts using axis tell them where the labels should be placed.

Axes of complementary X/Y flavours can be connected to each other in various ways, i.e. with a specific reference point, like an x/value axis to a y/value (or category) axis. In this case the connection can be either at the top or bottom of the former or at any absolute value (specified in points) or at some value of the former axes in its own coordinate system.

## Classes

### **AdjYValueAxis(YValueAxis)**

A Y-axis applying additional rules.

Depending on the data and some built-in rules, the axis may choose to adjust its range and origin.

## Public Attributes

**abf\_ignore\_zero** Set to True to make the avoidBoundFrac calculations treat zero as non-special

**annotations** list of annotations

**avoidBoundFrac** Fraction of interval to allow above and below.

**avoidBoundSpace** Space to allow above and below.

**drawGridLast** if true draw gridlines after everything else.

**forceZero** Ensure zero in range if true.

**gridEnd** End of grid lines wrt axis origin

**gridStart** Start of grid lines wrt axis origin

**gridStrokeColor** Color of grid lines.

**gridStrokeDashArray** Dash array used for grid lines.

**gridStrokeLineCap** Grid Line cap 0=butt, 1=round 2=square

**gridStrokeLineJoin** Grid Line join 0=miter, 1=round 2=bevel

**gridStrokeMiterLimit** Grid miter limit control miter line joins

**gridStrokeWidth** Width of grid lines.

**hiLLen** extra line length after end of the axis

**joinAxis** Join both axes if true.

**joinAxisMode** Mode used for connecting axis ('left', 'right', 'value', 'points', None).

**joinAxisPos** Position at which to join with other axis.

**keepTickLabelsInside** Ensure tick labels do not project beyond bounds of axis if true

**labelAxisMode** Like joinAxisMode, but for the axis labels

**labelTextFormat** Formatting string or function used for axis labels.

**labelTextPostFormat** Extra Formatting string.

**labelTextScale** Scaling for label tick values.

**labelVOffset** add this to the labels

**labels** Handle of the axis labels.

**leftAxisOrigShiftIPC** Lowest label shift interval ratio.

**leftAxisOrigShiftMin** Minimum amount to shift.

**leftAxisPercent** When true add percent sign to label values.

**leftAxisSkipLL0** Skip/Keep lowest tick label when true/false. Or skiplist

**loLLen** extra line length before start of the axis

**maximumTicks** Maximum number of ticks.

**minimumTickSpacing** Minimum value for distance between ticks.

**origShiftIPC** Lowest label shift interval ratio.

**origShiftMin** Minimum amount to shift.

**origShiftSpecialValue** special value for shift

**rangeRound** How to round the axis limits

**requiredRange** Minimum required value range.

**reverseDirection** If true reverse category direction.

**skipEndL** Skip high/low tick labels

**skipGrid** grid lines to skip top bottom both none

**strokeColor** Color of axis line and ticks.

**strokeDashArray** Dash array used for axis line.

**strokeLineCap** Line cap 0=butt, 1=round 2=square

**strokeLineJoin** Line join 0=miter, 1=round 2=bevel

**strokeMiterLimit** miter limit control miter line joins

**strokeWidth** Width of axis line and ticks.

**style** How values are plotted!

**subGridEnd** End of grid lines wrt axis origin

**subGridStart** Start of grid lines wrt axis origin

**subGridStrokeColor** Color of grid lines.

**subGridStrokeDashArray** Dash array used for grid lines.

**subGridStrokeLineCap** Grid Line cap 0=butt, 1=round 2=square

**subGridStrokeLineJoin** Grid Line join 0=miter, 1=round 2=bevel

**subGridStrokeMiterLimit** Grid miter limit control miter line joins

**subGridStrokeWidth** Width of grid lines.

**subTickHi** sub tick up or right

**subTickLo** sub tick down or left

**subTickNum** Number of axis sub ticks, if >0

**tickAxisMode** Like joinAxisMode, but for the ticks

**tickLeft** Tick length left of the axis.

**tickRight** Tick length right of the axis.

**valueMax** Maximum value on axis.

**valueMin** Minimum value on axis.

**valueStep** Step size used between ticks.

**valueSteps** List of step sizes used between ticks.

**visible** Display entire object, if true.

**visibleAxis** Display axis line, if true.

**visibleGrid** Display axis grid, if true.

**visibleLabels** Display axis labels, if true.

**visibleSubGrid** Display axis sub grid, if true.

**visibleSubTicks** Display axis sub ticks, if true.

**visibleTicks** Display axis ticks, if true.

**zrangePref** Zero range axis limit preference.

#### *Example*

```
def demo(self):
    data = [(10, 20, 30, 42)]
    self.setPosition(100, 10, 80)
    self.configure(data)

    drawing = Drawing(200, 100)
    drawing.add(self)
    return drawing
```

#### *Properties of Example Widget*

```
abf_ignore_zero = False
avoidBoundFrac = None
avoidBoundSpace = None
drawGridLast = False
forceZero = 0
gridEnd = None
gridStart = None
gridStrokeColor = Color(0,0,0,1)
gridStrokeDashArray = None
gridStrokeLineCap = 0
```

```
gridStrokeLineJoin = 0
gridStrokeMiterLimit = 10
gridStrokeWidth = 0.25
hiLen = 0
joinAxis = None
joinAxisMode = None
joinAxisPos = None
keepTickLabelsInside = 0
labelAxisMode = 'axis'
labelTextFormat = None
labelTextPostFormat = None
labelTextScale = None
labelVOffset = 0
labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x23826c8>
leftAxisOrigShiftIPC = 0.15
leftAxisOrigShiftMin = 12
leftAxisPercent = 1
leftAxisSkipLL0 = 0
loLen = 0
maximumTicks = 7
minimumTickSpacing = 10
origShiftIPC = None
origShiftMin = None
origShiftSpecialValue = None
rangeRound = 'none'
requiredRange = 30
reverseDirection = 0
skipEndL = 'none'
strokeColor = Color(0,0,0,1)
strokeDashArray = None
strokeLineCap = 0
strokeLineJoin = 0
strokeMiterLimit = 10
strokeWidth = 1
style = 'normal'
subGridEnd = None
subGridStart = None
subGridStrokeColor = Color(0,0,0,1)
subGridStrokeDashArray = None
subGridStrokeLineCap = 0
subGridStrokeLineJoin = 0
subGridStrokeMiterLimit = 10
subGridStrokeWidth = 0.25
subTickHi = 0
subTickLo = 0
subTickNum = 0
tickAxisMode = 'axis'
tickLeft = 5
tickRight = 0
valueMax = None
valueMin = None
valueStep = None
valueSteps = None
visible = 1
visibleAxis = 1
visibleGrid = 0
visibleLabels = 1
visibleSubGrid = 0
visibleSubTicks = 0
visibleTicks = 1
zrangePref = 0
```

## **CALabel ( PMVLabel )**

### **Public Attributes**

**angle** angle of label: default (0), 90 is vertical, 180 is upside down, etc

**bottomPadding** padding at bottom of box

**boxAnchor** anchoring point of the label

**boxFillColor** the filling color of the box

**boxStrokeColor** border color of the box

**boxStrokeWidth** border width

**boxTarget** one of ('normal','anti','lo','hi')

**customDrawChanger** An instance of CustomDrawChanger to modify the behavior at draw time

**dx** delta x - offset

**dy** delta y - offset

**fillColor** label text color

**fontName** the name of the font used

**fontSize** the size of the font

**height** the height of the text

**labelPosFrac** where in the category range [0,1] the labels should be anchored

**leading**

**leftPadding** padding at left of box

**maxWidth** maximum width the label can grow to

**rightPadding** padding at right of box

**strokeColor** label text border color

**strokeWidth** label text border width

**text** the actual text to display

**textAnchor** the anchoring point of the text inside the label

**topPadding** padding at top of box

**useAscentDescent** If True then the font's Ascent Descent will be used to compute default heights and baseline.

**visible** True if the label is to be drawn

**width** the width of the label

**x**

**y**

### *Example*

```
def demo(self):
    """This shows a label positioned with its top right corner
    at the top centre of the drawing, and rotated 45 degrees."""

    d = Drawing(200, 100)

    # mark the origin of the label
```

```
d.add(Circle(100,90, 5, fillColor=colors.green))

lab = Label()
lab.setOrigin(100,90)
lab.boxAnchor = 'ne'
lab.angle = 45
lab.dx = 0
lab.dy = -20
lab.boxStrokeColor = colors.green
lab.setText('Another\nMulti-Line\nString')
d.add(lab)

return d
```

### *Properties of Example Widget*

```
angle = 0
bottomPadding = 0
boxAnchor = 'c'
boxFillColor = None
boxStrokeColor = None
boxStrokeWidth = 0.5
boxTarget = 'normal'
dx = 0
dy = 0
fillColor = Color(0,0,0,1)
fontName = 'Times-Roman'
fontSize = 10
height = None
labelPosFrac = 0.5
leading = None
leftPadding = 0
maxWidth = None
rightPadding = 0
strokeColor = None
strokeWidth = 0.1
textAnchor = 'start'
topPadding = 0
useAscentDescent = False
visible = 1
width = None
x = 0
y = 0
```



## **CategoryAxis(\_AxisG)**

Abstract category axis, unusable in itself.

### **Public Attributes**

**annotations** list of annotations

**categoryNames** List of category names.

**drawGridLast** if true draw gridlines after everything else.

**gridEnd** End of grid lines wrt axis origin

**gridStart** Start of grid lines wrt axis origin

**gridStrokeColor** Color of grid lines.

**gridStrokeDashArray** Dash array used for grid lines.

**gridStrokeLineCap** Grid Line cap 0=butt, 1=round 2=square

**gridStrokeLineJoin** Grid Line join 0=miter, 1=round 2=bevel

**gridStrokeMiterLimit** Grid miter limit control miter line joins

**gridStrokeWidth** Width of grid lines.

**hiLLen** extra line length after end of the axis

**hiPad** extra inner space after end of the axis

**joinAxis** Join both axes if true.

**joinAxisPos** Position at which to join with other axis.

**labelAxisMode** Like joinAxisMode, but for the axis labels

**labels** Handle of the axis labels.

**loLLen** extra line length before start of the axis

**loPad** extra inner space before start of the axis

**reverseDirection** If true reverse category direction.

**skipGrid** grid lines to skip top bottom both none

**strokeColor** Color of axis line and ticks.

**strokeDashArray** Dash array used for axis line.

**strokeLineCap** Line cap 0=butt, 1=round 2=square

**strokeLineJoin** Line join 0=miter, 1=round 2=bevel

**strokeMiterLimit** miter limit control miter line joins

**strokeWidth** Width of axis line and ticks.

**style** How common category bars are plotted

**tickShift** Tick shift typically

**visible** Display entire object, if true.

**visibleAxis** Display axis line, if true.

**visibleGrid** Display axis grid, if true.

**visibleLabels** Display axis labels, if true.

**visibleTicks** Display axis ticks, if true.

## **NormalDateXValueAxis(XValueAxis)**

An X axis applying additional rules.

Depending on the data and some built-in rules, the axis displays normalDate values as nicely formatted dates.

The client chart should have NormalDate X values.

### **Public Attributes**

**abf\_ignore\_zero** Set to True to make the avoidBoundFrac calculations treat zero as non-special

**annotations** list of annotations

**avoidBoundFrac** Fraction of interval to allow above and below.

**avoidBoundSpace** Space to allow above and below.

**bottomAxisLabelSlack** Fractional amount used to adjust label spacing

**dailyFreq** True if we are to assume daily data to be ticked at end of month.

**dayOfWeekName** Weekday names.

**drawGridLast** if true draw gridlines after everything else.

**forceDatesEachYear** List of dates in format "31-Dec", "1-Jan". If present they will always be used for tick marks in the current year, rather than the dates chosen by the automatic algorithm. Hyphen compulsory, case of month optional.

**forceEndDate** Flag for enforced displaying of last date value.

**forceFirstDate** Flag for enforced displaying of first date value.

**forceZero** Ensure zero in range if true.

**gridEnd** End of grid lines wrt axis origin

**gridStart** Start of grid lines wrt axis origin

**gridStrokeColor** Color of grid lines.

**gridStrokeDashArray** Dash array used for grid lines.

**gridStrokeLineCap** Grid Line cap 0=butt, 1=round 2=square

**gridStrokeLineJoin** Grid Line join 0=miter, 1=round 2=bevel

**gridStrokeMiterLimit** Grid miter limit control miter line joins

**gridStrokeWidth** Width of grid lines.

**hiLLen** extra line length after end of the axis

**joinAxis** Join both axes if true.

**joinAxisMode** Mode used for connecting axis ('bottom', 'top', 'value', 'points', None).

**joinAxisPos** Position at which to join with other axis.

**keepTickLabelsInside** Ensure tick labels do not project beyond bounds of axis if true

**labelAxisMode** Like joinAxisMode, but for the axis labels

**labelTextFormat** Formatting string or function used for axis labels.

**labelTextPostFormat** Extra Formatting string.

**labelTextScale** Scaling for label tick values.

**labels** Handle of the axis labels.

**loLLen** extra line length before start of the axis

**maximumTicks** Maximum number of ticks.

**minimumTickSpacing** Minimum value for distance between ticks.

**monthName** Month names.

**niceMonth** Flag for displaying months 'nicely'.

**origShiftIPC** Lowest label shift interval ratio.

**origShiftMin** Minimum amount to shift.

**origShiftSpecialValue** special value for shift

**rangeRound** How to round the axis limits

**requiredRange** Minimum required value range.

**reverseDirection** If true reverse category direction.

**skipEndL** Skip high/low tick labels

**skipGrid** grid lines to skip top bottom both none

**specialTickClear** clear rather than delete close ticks when forced first/end dates

**specifiedTickDates** Actual tick values to use; no calculations done

**strokeColor** Color of axis line and ticks.

**strokeDashArray** Dash array used for axis line.

**strokeLineCap** Line cap 0=butt, 1=round 2=square

**strokeLineJoin** Line join 0=miter, 1=round 2=bevel

**strokeMiterLimit** miter limit control miter line joins

**strokeWidth** Width of axis line and ticks.

**style** How values are plotted!

**subGridEnd** End of grid lines wrt axis origin

**subGridStart** Start of grid lines wrt axis origin

**subGridStrokeColor** Color of grid lines.

**subGridStrokeDashArray** Dash array used for grid lines.

**subGridStrokeLineCap** Grid Line cap 0=butt, 1=round 2=square

**subGridStrokeLineJoin** Grid Line join 0=miter, 1=round 2=bevel

**subGridStrokeMiterLimit** Grid miter limit control miter line joins

**subGridStrokeWidth** Width of grid lines.

**subTickHi** sub tick up or right

**subTickLo** sub tick down or left

**subTickNum** Number of axis sub ticks, if >0

**tickAxisMode** Like joinAxisMode, but for the ticks

**tickDown** Tick length down the axis.

**tickUp** Tick length up the axis.

**valueMax** Maximum value on axis.

**valueMin** Minimum value on axis.

**valueStep** Step size used between ticks.

**valueSteps** List of step sizes used between ticks.

**visible** Display entire object, if true.

**visibleAxis** Display axis line, if true.

**visibleGrid** Display axis grid, if true.

**visibleLabels** Display axis labels, if true.

**visibleSubGrid** Display axis sub grid, if true.

**visibleSubTicks** Display axis sub ticks, if true.

**visibleTicks** Display axis ticks, if true.

**xLabelFormat** Label format string (e.g. '{mm}/{yy}') or function.

**zrangePref** Zero range axis limit preference.

#### *Example*

```
def demo(self):
    self.setPosition(20, 50, 150)
    self.configure([(10,20,30,40,50)])

    d = Drawing(200, 100)
    d.add(self)
    return d
```

#### *Properties of Example Widget*

```
abf_ignore_zero = False
avoidBoundFrac = None
avoidBoundSpace = None
bottomAxisLabelSlack = 0.1
dailyFreq = 0
dayOfWeekName = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
drawGridLast = False
forceDatesEachYear = []
forceEndDate = 0
forceFirstDate = 0
forceZero = 0
gridEnd = None
gridStart = None
gridStrokeColor = Color(0,0,0,1)
gridStrokeDashArray = None
gridStrokeLineCap = 0
gridStrokeLineJoin = 0
gridStrokeMiterLimit = 10
gridStrokeWidth = 0.25
hiLen = 0
joinAxis = None
joinAxisMode = None
joinAxisPos = None
keepTickLabelsInside = 0
labelAxisMode = 'axis'
labelTextFormat = None
labelTextPostFormat = None
labelTextScale = None
labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x245a560>
loLen = 0
maximumTicks = 7
minimumTickSpacing = 10
monthName = ['January',
              'February',
              'March',
              'April',
              'May',
              'June',
              'July',
              'August',
```

```
        'September',
        'October',
        'November',
        'December']
niceMonth = 1
origShiftIPC = None
origShiftMin = None
origShiftSpecialValue = None
rangeRound = 'none'
requiredRange = 0
reverseDirection = 0
skipEndL = 'none'
specialTickClear = 0
specifiedTickDates = None
strokeColor = Color(0,0,0,1)
strokeDashArray = None
strokeLineCap = 0
strokeLineJoin = 0
strokeMiterLimit = 10
strokeWidth = 1
style = 'normal'
subGridEnd = None
subGridStart = None
subGridStrokeColor = Color(0,0,0,1)
subGridStrokeDashArray = None
subGridStrokeLineCap = 0
subGridStrokeLineJoin = 0
subGridStrokeMiterLimit = 10
subGridStrokeWidth = 0.25
subTickHi = 0
subTickLo = 0
subTickNum = 0
tickAxisMode = 'axis'
tickDown = 5
tickUp = 0
valueMax = None
valueMin = None
valueStep = None
valueSteps = None
visible = 1
visibleAxis = 1
visibleGrid = 0
visibleLabels = 1
visibleSubGrid = 0
visibleSubTicks = 0
visibleTicks = 1
xLabelFormat = '{mm}/{yy}'
zrangePref = 0
```

## **ValueAxis(\_AxisG)**

Abstract value axis, unusable in itself.

### **Public Attributes**

**abf\_ignore\_zero** Set to True to make the avoidBoundFrac calculations treat zero as non-special

**annotations** list of annotations

**avoidBoundFrac** Fraction of interval to allow above and below.

**avoidBoundSpace** Space to allow above and below.

**drawGridLast** if true draw gridlines after everything else.

**forceZero** Ensure zero in range if true.

**gridEnd** End of grid lines wrt axis origin

**gridStart** Start of grid lines wrt axis origin

**gridStrokeColor** Color of grid lines.

**gridStrokeDashArray** Dash array used for grid lines.

**gridStrokeLineCap** Grid Line cap 0=butt, 1=round 2=square

**gridStrokeLineJoin** Grid Line join 0=miter, 1=round 2=bevel

**gridStrokeMiterLimit** Grid miter limit control miter line joins

**gridStrokeWidth** Width of grid lines.

**hiLLen** extra line length after end of the axis

**keepTickLabelsInside** Ensure tick labels do not project beyond bounds of axis if true

**labelAxisMode** Like joinAxisMode, but for the axis labels

**labelTextFormat** Formatting string or function used for axis labels.

**labelTextPostFormat** Extra Formatting string.

**labelTextScale** Scaling for label tick values.

**labels** Handle of the axis labels.

**loLLen** extra line length before start of the axis

**maximumTicks** Maximum number of ticks.

**minimumTickSpacing** Minimum value for distance between ticks.

**origShiftIPC** Lowest label shift interval ratio.

**origShiftMin** Minimum amount to shift.

**origShiftSpecialValue** special value for shift

**rangeRound** How to round the axis limits

**requiredRange** Minimum required value range.

**reverseDirection** If true reverse category direction.

**skipEndL** Skip high/low tick labels

**skipGrid** grid lines to skip top bottom both none

**strokeColor** Color of axis line and ticks.

**strokeDashArray** Dash array used for axis line.

**strokeLineCap** Line cap 0=butt, 1=round 2=square  
**strokeLineJoin** Line join 0=miter, 1=round 2=bevel  
**strokeMiterLimit** miter limit control miter line joins  
**strokeWidth** Width of axis line and ticks.  
**style** How values are plotted!  
**subGridEnd** End of grid lines wrt axis origin  
**subGridStart** Start of grid lines wrt axis origin  
**subGridStrokeColor** Color of grid lines.  
**subGridStrokeDashArray** Dash array used for grid lines.  
**subGridStrokeLineCap** Grid Line cap 0=butt, 1=round 2=square  
**subGridStrokeLineJoin** Grid Line join 0=miter, 1=round 2=bevel  
**subGridStrokeMiterLimit** Grid miter limit control miter line joins  
**subGridStrokeWidth** Width of grid lines.  
**subTickHi** sub tick up or right  
**subTickLo** sub tick down or left  
**subTickNum** Number of axis sub ticks, if >0  
**tickAxisMode** Like joinAxisMode, but for the ticks  
**valueMax** Maximum value on axis.  
**valueMin** Minimum value on axis.  
**valueStep** Step size used between ticks.  
**valueSteps** List of step sizes used between ticks.  
**visible** Display entire object, if true.  
**visibleAxis** Display axis line, if true.  
**visibleGrid** Display axis grid, if true.  
**visibleLabels** Display axis labels, if true.  
**visibleSubGrid** Display axis sub grid, if true.  
**visibleSubTicks** Display axis sub ticks, if true.  
**visibleTicks** Display axis ticks, if true.  
**zrangePref** Zero range axis limit preference.

## **XCategoryAxis(\_XTicks, CategoryAxis)**

X/category axis

### **Public Attributes**

**annotations** list of annotations  
**categoryNames** List of category names.  
**drawGridLast** if true draw gridlines after everything else.  
**gridEnd** End of grid lines wrt axis origin  
**gridStart** Start of grid lines wrt axis origin

**gridStrokeColor** Color of grid lines.

**gridStrokeDashArray** Dash array used for grid lines.

**gridStrokeLineCap** Grid Line cap 0=butt, 1=round 2=square

**gridStrokeLineJoin** Grid Line join 0=miter, 1=round 2=bevel

**gridStrokeMiterLimit** Grid miter limit control miter line joins

**gridStrokeWidth** Width of grid lines.

**hiLLen** extra line length after end of the axis

**hiPad** extra inner space after end of the axis

**joinAxis** Join both axes if true.

**joinAxisMode** Mode used for connecting axis ('bottom', 'top', 'value', 'points', None).

**joinAxisPos** Position at which to join with other axis.

**labelAxisMode** Like joinAxisMode, but for the axis labels

**labels** Handle of the axis labels.

**loLLen** extra line length before start of the axis

**loPad** extra inner space before start of the axis

**reverseDirection** If true reverse category direction.

**skipGrid** grid lines to skip top bottom both none

**strokeColor** Color of axis line and ticks.

**strokeDashArray** Dash array used for axis line.

**strokeLineCap** Line cap 0=butt, 1=round 2=square

**strokeLineJoin** Line join 0=miter, 1=round 2=bevel

**strokeMiterLimit** miter limit control miter line joins

**strokeWidth** Width of axis line and ticks.

**style** How common category bars are plotted

**tickDown** Tick length down the axis.

**tickShift** Tick shift typically

**tickUp** Tick length up the axis.

**visible** Display entire object, if true.

**visibleAxis** Display axis line, if true.

**visibleGrid** Display axis grid, if true.

**visibleLabels** Display axis labels, if true.

**visibleTicks** Display axis ticks, if true.

## **XValueAxis(\_XTicks, ValueAxis)**

X/value axis

## **Public Attributes**

**abf\_ignore\_zero** Set to True to make the avoidBoundFrac calculations treat zero as non-special



**annotations** list of annotations

**avoidBoundFrac** Fraction of interval to allow above and below.

**avoidBoundSpace** Space to allow above and below.

**drawGridLast** if true draw gridlines after everything else.

**forceZero** Ensure zero in range if true.

**gridEnd** End of grid lines wrt axis origin

**gridStart** Start of grid lines wrt axis origin

**gridStrokeColor** Color of grid lines.

**gridStrokeDashArray** Dash array used for grid lines.

**gridStrokeLineCap** Grid Line cap 0=butt, 1=round 2=square

**gridStrokeLineJoin** Grid Line join 0=miter, 1=round 2=bevel

**gridStrokeMiterLimit** Grid miter limit control miter line joins

**gridStrokeWidth** Width of grid lines.

**hiLLen** extra line length after end of the axis

**joinAxis** Join both axes if true.

**joinAxisMode** Mode used for connecting axis ('bottom', 'top', 'value', 'points', None).

**joinAxisPos** Position at which to join with other axis.

**keepTickLabelsInside** Ensure tick labels do not project beyond bounds of axis if true

**labelAxisMode** Like joinAxisMode, but for the axis labels

**labelTextFormat** Formatting string or function used for axis labels.

**labelTextPostFormat** Extra Formatting string.

**labelTextScale** Scaling for label tick values.

**labels** Handle of the axis labels.

**loLLen** extra line length before start of the axis

**maximumTicks** Maximum number of ticks.

**minimumTickSpacing** Minimum value for distance between ticks.

**origShiftIPC** Lowest label shift interval ratio.

**origShiftMin** Minimum amount to shift.

**origShiftSpecialValue** special value for shift

**rangeRound** How to round the axis limits

**requiredRange** Minimum required value range.

**reverseDirection** If true reverse category direction.

**skipEndL** Skip high/low tick labels

**skipGrid** grid lines to skip top bottom both none

**strokeColor** Color of axis line and ticks.

**strokeDashArray** Dash array used for axis line.

**strokeLineCap** Line cap 0=butt, 1=round 2=square

**strokeLineJoin** Line join 0=miter, 1=round 2=bevel

**strokeMiterLimit** miter limit control miter line joins

**strokeWidth** Width of axis line and ticks.

**style** How values are plotted!

**subGridEnd** End of grid lines wrt axis origin

**subGridStart** Start of grid lines wrt axis origin

**subGridStrokeColor** Color of grid lines.

**subGridStrokeDashArray** Dash array used for grid lines.

**subGridStrokeLineCap** Grid Line cap 0=butt, 1=round 2=square

**subGridStrokeLineJoin** Grid Line join 0=miter, 1=round 2=bevel

**subGridStrokeMiterLimit** Grid miter limit control miter line joins

**subGridStrokeWidth** Width of grid lines.

**subTickHi** sub tick up or right

**subTickLo** sub tick down or left

**subTickNum** Number of axis sub ticks, if >0

**tickAxisMode** Like joinAxisMode, but for the ticks

**tickDown** Tick length down the axis.

**tickUp** Tick length up the axis.

**valueMax** Maximum value on axis.

**valueMin** Minimum value on axis.

**valueStep** Step size used between ticks.

**valueSteps** List of step sizes used between ticks.

**visible** Display entire object, if true.

**visibleAxis** Display axis line, if true.

**visibleGrid** Display axis grid, if true.

**visibleLabels** Display axis labels, if true.

**visibleSubGrid** Display axis sub grid, if true.

**visibleSubTicks** Display axis sub ticks, if true.

**visibleTicks** Display axis ticks, if true.

**zrangePref** Zero range axis limit preference.

#### *Example*

```
def demo(self):
    self.setPosition(20, 50, 150)
    self.configure([(10,20,30,40,50)])

    d = Drawing(200, 100)
    d.add(self)
    return d
```

#### *Properties of Example Widget*

```
abf_ignore_zero = False
avoidBoundFrac = None
avoidBoundSpace = None
drawGridLast = False
forceZero = 0
```

```
gridEnd = None
gridStart = None
gridStrokeColor = Color(0,0,0,1)
gridStrokeDashArray = None
gridStrokeLineCap = 0
gridStrokeLineJoin = 0
gridStrokeMiterLimit = 10
gridStrokeWidth = 0.25
hiLen = 0
joinAxis = None
joinAxisMode = None
joinAxisPos = None
keepTickLabelsInside = 0
labelAxisMode = 'axis'
labelTextFormat = None
labelTextPostFormat = None
labelTextScale = None
labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x2548440>
loLen = 0
maximumTicks = 7
minimumTickSpacing = 10
origShiftIPC = None
origShiftMin = None
origShiftSpecialValue = None
rangeRound = 'none'
requiredRange = 0
reverseDirection = 0
skipEndL = 'none'
strokeColor = Color(0,0,0,1)
strokeDashArray = None
strokeLineCap = 0
strokeLineJoin = 0
strokeMiterLimit = 10
strokeWidth = 1
style = 'normal'
subGridEnd = None
subGridStart = None
subGridStrokeColor = Color(0,0,0,1)
subGridStrokeDashArray = None
subGridStrokeLineCap = 0
subGridStrokeLineJoin = 0
subGridStrokeMiterLimit = 10
subGridStrokeWidth = 0.25
subTickHi = 0
subTickLo = 0
subTickNum = 0
tickAxisMode = 'axis'
tickDown = 5
tickUp = 0
valueMax = None
valueMin = None
valueStep = None
visible = 1
visibleAxis = 1
visibleGrid = 0
visibleLabels = 1
visibleSubGrid = 0
visibleSubTicks = 0
visibleTicks = 1
zrangePref = 0
```

**YCategoryAxis(\_YTicks, CategoryAxis)**

Y/category axis

**Public Attributes****annotations** list of annotations**categoryNames** List of category names.**drawGridLast** if true draw gridlines after everything else.**gridEnd** End of grid lines wrt axis origin**gridStart** Start of grid lines wrt axis origin**gridStrokeColor** Color of grid lines.**gridStrokeDashArray** Dash array used for grid lines.**gridStrokeLineCap** Grid Line cap 0=butt, 1=round 2=square**gridStrokeLineJoin** Grid Line join 0=miter, 1=round 2=bevel**gridStrokeMiterLimit** Grid miter limit control miter line joins**gridStrokeWidth** Width of grid lines.**hiLLen** extra line length after end of the axis**hiPad** extra inner space after end of the axis**joinAxis** Join both axes if true.**joinAxisMode** Mode used for connecting axis ('left', 'right', 'value', 'points', None).**joinAxisPos** Position at which to join with other axis.**labelAxisMode** Like joinAxisMode, but for the axis labels**labels** Handle of the axis labels.**loLLen** extra line length before start of the axis**loPad** extra inner space before start of the axis**reverseDirection** If true reverse category direction.**skipGrid** grid lines to skip top bottom both none**strokeColor** Color of axis line and ticks.**strokeDashArray** Dash array used for axis line.**strokeLineCap** Line cap 0=butt, 1=round 2=square**strokeLineJoin** Line join 0=miter, 1=round 2=bevel**strokeMiterLimit** miter limit control miter line joins**strokeWidth** Width of axis line and ticks.**style** How common category bars are plotted**tickLeft** Tick length left of the axis.**tickRight** Tick length right of the axis.**tickShift** Tick shift typically**visible** Display entire object, if true.**visibleAxis** Display axis line, if true.

**visibleGrid** Display axis grid, if true.

**visibleLabels** Display axis labels, if true.

**visibleTicks** Display axis ticks, if true.

## **YValueAxis(\_YTicks, ValueAxis)**

Y/value axis

### **Public Attributes**

**abf\_ignore\_zero** Set to True to make the avoidBoundFrac calculations treat zero as non-special

**annotations** list of annotations

**avoidBoundFrac** Fraction of interval to allow above and below.

**avoidBoundSpace** Space to allow above and below.

**drawGridLast** if true draw gridlines after everything else.

**forceZero** Ensure zero in range if true.

**gridEnd** End of grid lines wrt axis origin

**gridStart** Start of grid lines wrt axis origin

**gridStrokeColor** Color of grid lines.

**gridStrokeDashArray** Dash array used for grid lines.

**gridStrokeLineCap** Grid Line cap 0=butt, 1=round 2=square

**gridStrokeLineJoin** Grid Line join 0=miter, 1=round 2=bevel

**gridStrokeMiterLimit** Grid miter limit control miter line joins

**gridStrokeWidth** Width of grid lines.

**hiLLen** extra line length after end of the axis

**joinAxis** Join both axes if true.

**joinAxisMode** Mode used for connecting axis ('left', 'right', 'value', 'points', None).

**joinAxisPos** Position at which to join with other axis.

**keepTickLabelsInside** Ensure tick labels do not project beyond bounds of axis if true

**labelAxisMode** Like joinAxisMode, but for the axis labels

**labelTextFormat** Formatting string or function used for axis labels.

**labelTextPostFormat** Extra Formatting string.

**labelTextScale** Scaling for label tick values.

**labels** Handle of the axis labels.

**loLLen** extra line length before start of the axis

**maximumTicks** Maximum number of ticks.

**minimumTickSpacing** Minimum value for distance between ticks.

**origShiftIPC** Lowest label shift interval ratio.

**origShiftMin** Minimum amount to shift.

**origShiftSpecialValue** special value for shift

**rangeRound** How to round the axis limits

**requiredRange** Minimum required value range.

**reverseDirection** If true reverse category direction.

**skipEndL** Skip high/low tick labels

**skipGrid** grid lines to skip top bottom both none

**strokeColor** Color of axis line and ticks.

**strokeDashArray** Dash array used for axis line.

**strokeLineCap** Line cap 0=butt, 1=round 2=square

**strokeLineJoin** Line join 0=miter, 1=round 2=bevel

**strokeMiterLimit** miter limit control miter line joins

**strokeWidth** Width of axis line and ticks.

**style** How values are plotted!

**subGridEnd** End of grid lines wrt axis origin

**subGridStart** Start of grid lines wrt axis origin

**subGridStrokeColor** Color of grid lines.

**subGridStrokeDashArray** Dash array used for grid lines.

**subGridStrokeLineCap** Grid Line cap 0=butt, 1=round 2=square

**subGridStrokeLineJoin** Grid Line join 0=miter, 1=round 2=bevel

**subGridStrokeMiterLimit** Grid miter limit control miter line joins

**subGridStrokeWidth** Width of grid lines.

**subTickHi** sub tick up or right

**subTickLo** sub tick down or left

**subTickNum** Number of axis sub ticks, if >0

**tickAxisMode** Like joinAxisMode, but for the ticks

**tickLeft** Tick length left of the axis.

**tickRight** Tick length right of the axis.

**valueMax** Maximum value on axis.

**valueMin** Minimum value on axis.

**valueStep** Step size used between ticks.

**valueSteps** List of step sizes used between ticks.

**visible** Display entire object, if true.

**visibleAxis** Display axis line, if true.

**visibleGrid** Display axis grid, if true.

**visibleLabels** Display axis labels, if true.

**visibleSubGrid** Display axis sub grid, if true.

**visibleSubTicks** Display axis sub ticks, if true.

**visibleTicks** Display axis ticks, if true.

**zrangePref** Zero range axis limit preference.

*Example*

```
def demo(self):
    data = [(10, 20, 30, 42)]
    self.setPosition(100, 10, 80)
    self.configure(data)

    drawing = Drawing(200, 100)
    drawing.add(self)
    return drawing
```

*Properties of Example Widget*

```
abf_ignore_zero = False
avoidBoundFrac = None
avoidBoundSpace = None
drawGridLast = False
forceZero = 0
gridEnd = None
gridStart = None
gridStrokeColor = Color(0,0,0,1)
gridStrokeDashArray = None
gridStrokeLineCap = 0
gridStrokeLineJoin = 0
gridStrokeMiterLimit = 10
gridStrokeWidth = 0.25
hiLLen = 0
joinAxis = None
joinAxisMode = None
joinAxisPos = None
keepTickLabelsInside = 0
labelAxisMode = 'axis'
labelTextFormat = None
labelTextPostFormat = None
labelTextScale = None
labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x260b680>
loLLen = 0
maximumTicks = 7
minimumTickSpacing = 10
origShiftIPC = None
origShiftMin = None
origShiftSpecialValue = None
rangeRound = 'none'
requiredRange = 0
reverseDirection = 0
skipEndL = 'none'
strokeColor = Color(0,0,0,1)
strokeDashArray = None
strokeLineCap = 0
strokeLineJoin = 0
strokeMiterLimit = 10
strokeWidth = 1
style = 'normal'
subGridEnd = None
subGridStart = None
subGridStrokeColor = Color(0,0,0,1)
subGridStrokeDashArray = None
subGridStrokeLineCap = 0
subGridStrokeLineJoin = 0
subGridStrokeMiterLimit = 10
subGridStrokeWidth = 0.25
subTickHi = 0
subTickLo = 0
subTickNum = 0
tickAxisMode = 'axis'
tickLeft = 5
tickRight = 0
valueMax = None
valueMin = None
valueStep = None
visible = 1
visibleAxis = 1
visibleGrid = 0
visibleLabels = 1
```

```
visibleSubGrid = 0
visibleSubTicks = 0
visibleTicks = 1
zrangePref = 0
```



## **`_AxisG(Widget)`**

### **Public Attributes**

#### *Example*

```
def demo(self):  
    msg = "demo() must be implemented for each Widget!"  
    raise shapes.NotImplementedError, msg
```

#### *Properties of Example Widget*

## Functions

### **sample0a( ... )**

Sample drawing with one xcat axis and two buckets.

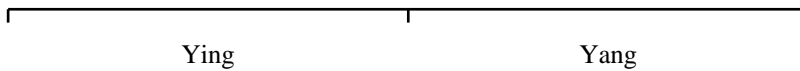
#### *Example*

```
def sample0a():
    "Sample drawing with one xcat axis and two buckets."

    drawing = Drawing(400, 200)

    data = [(10, 20)]

    xAxis = XCategoryAxis()
    xAxis.setPosition(75, 75, 300)
    xAxis.configure(data)
    xAxis.categoryNames = ['Ying', 'Yang']
    xAxis.labels.boxAnchor = 'n'
    drawing.add(xAxis)
    return drawing
```



**sample0b( ... )**

Sample drawing with one xcat axis and one bucket only.

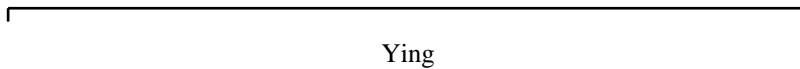
*Example*

```
def sample0b():
    "Sample drawing with one xcat axis and one bucket only."

    drawing = Drawing(400, 200)

    data = [(10,)]

    xAxis = XCategoryAxis()
    xAxis.setPosition(75, 75, 300)
    xAxis.configure(data)
    xAxis.categoryNames = ['Ying']
    xAxis.labels.boxAnchor = 'n'
    drawing.add(xAxis)
    return drawing
```

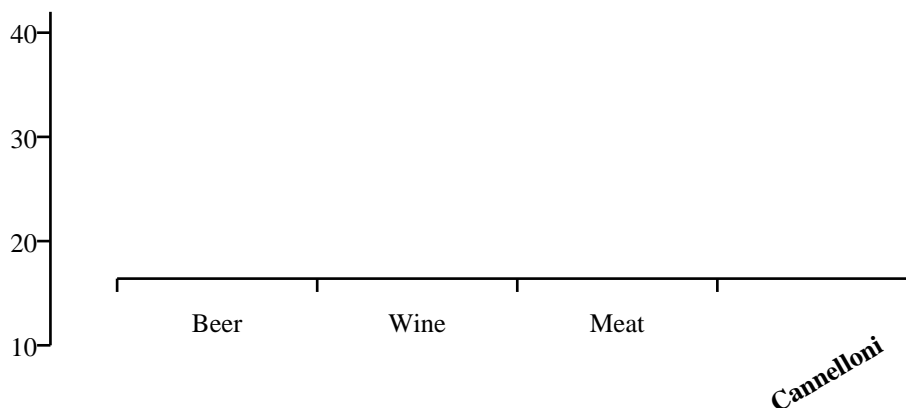


**sample1( ... )**

Sample drawing containing two unconnected axes.

*Example*

```
def sample1():
    "Sample drawing containing two unconnected axes."
    from reportlab.graphics.shapes import _baseGFontNameB
    drawing = Drawing(400, 200)
    data = [(10, 20, 30, 42)]
    xAxis = XCategoryAxis()
    xAxis.setPosition(75, 75, 300)
    xAxis.configure(data)
    xAxis.categoryNames = ['Beer', 'Wine', 'Meat', 'Cannelloni']
    xAxis.labels.boxAnchor = 'n'
    xAxis.labels[3].dy = -15
    xAxis.labels[3].angle = 30
    xAxis.labels[3].fontName = _baseGFontNameB
    yAxis = YValueAxis()
    yAxis.setPosition(50, 50, 125)
    yAxis.configure(data)
    drawing.add(xAxis)
    drawing.add(yAxis)
    return drawing
```

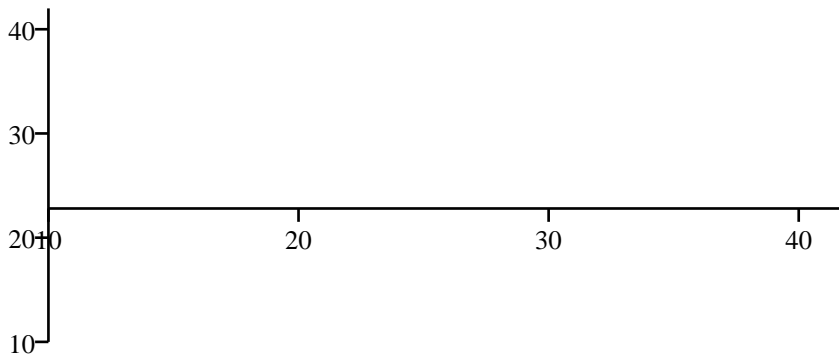


**sample4a( ... )**

Sample drawing, xvalue/yvalue axes, y connected at 100 pts to x.

*Example*

```
def sample4a():
    "Sample drawing, xvalue/yvalue axes, y connected at 100 pts to x."
    drawing = Drawing(400, 200)
    data = [(10, 20, 30, 42)]
    yAxis = YValueAxis()
    yAxis.setPosition(50, 50, 125)
    yAxis.configure(data)
    xAxis = XValueAxis()
    xAxis._length = 300
    xAxis.joinAxis = yAxis
    xAxis.joinAxisMode = 'points'
    xAxis.joinAxisPos = 100
    xAxis.configure(data)
    drawing.add(xAxis)
    drawing.add(yAxis)
    return drawing
```

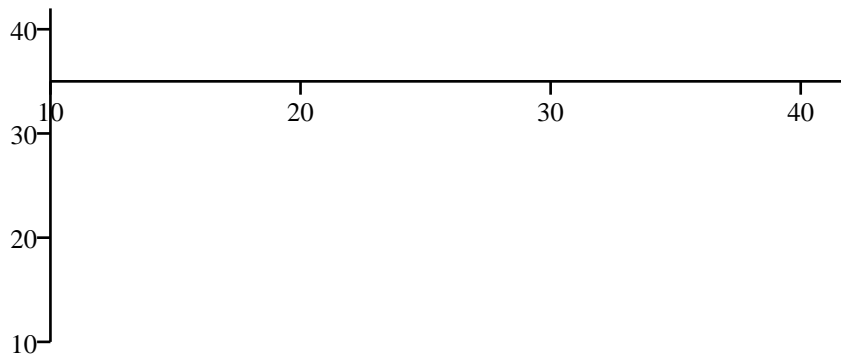


**sample4b( ... )**

Sample drawing, xvalue/yvalue axes, y connected at value 35 of x.

*Example*

```
def sample4b():
    "Sample drawing, xvalue/yvalue axes, y connected at value 35 of x."
    drawing = Drawing(400, 200)
    data = [(10, 20, 30, 42)]
    yAxis = YValueAxis()
    yAxis.setPosition(50, 50, 125)
    yAxis.configure(data)
    xAxis = XValueAxis()
    xAxis._length = 300
    xAxis.joinAxis = yAxis
    xAxis.joinAxisMode = 'value'
    xAxis.joinAxisPos = 35
    xAxis.configure(data)
    drawing.add(xAxis)
    drawing.add(yAxis)
    return drawing
```

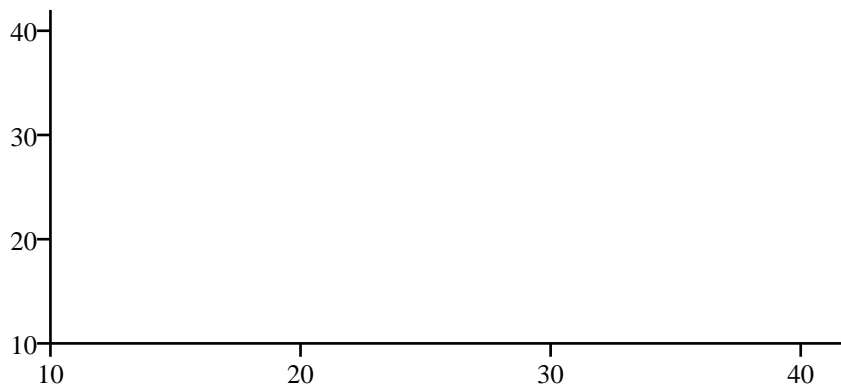


**sample4c( ... )**

Sample drawing, xvalue/yvalue axes, y connected to bottom of x.

*Example*

```
def sample4c():
    "Sample drawing, xvalue/yvalue axes, y connected to bottom of x."
    drawing = Drawing(400, 200)
    data = [(10, 20, 30, 42)]
    yAxis = YValueAxis()
    yAxis.setPosition(50, 50, 125)
    yAxis.configure(data)
    xAxis = XValueAxis()
    xAxis._length = 300
    xAxis.joinAxis = yAxis
    xAxis.joinAxisMode = 'bottom'
    xAxis.configure(data)
    drawing.add(xAxis)
    drawing.add(yAxis)
    return drawing
```

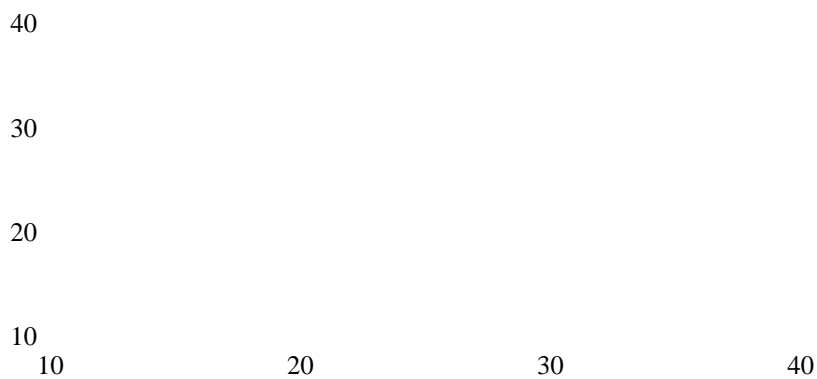


**sample4c1( ... )**

xvalue/yvalue axes, without drawing axis lines/ticks.

*Example*

```
def sample4c1():
    "xvalue/yvalue axes, without drawing axis lines/ticks."
    drawing = Drawing(400, 200)
    data = [(10, 20, 30, 42)]
    yAxis = YValueAxis()
    yAxis.setPosition(50, 50, 125)
    yAxis.configure(data)
    yAxis.visibleAxis = 0
    yAxis.visibleTicks = 0
    xAxis = XValueAxis()
    xAxis._length = 300
    xAxis.joinAxis = yAxis
    xAxis.joinAxisMode = 'bottom'
    xAxis.configure(data)
    xAxis.visibleAxis = 0
    xAxis.visibleTicks = 0
    drawing.add(xAxis)
    drawing.add(yAxis)
    return drawing
```



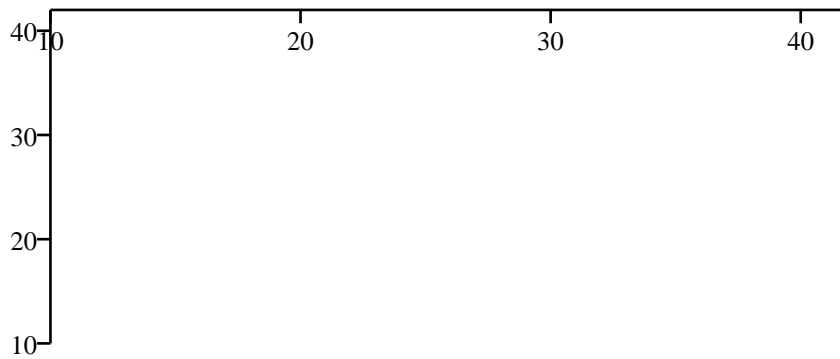


**sample4d( ... )**

Sample drawing, xvalue/yvalue axes, y connected to top of x.

*Example*

```
def sample4d():
    "Sample drawing, xvalue/yvalue axes, y connected to top of x."
    drawing = Drawing(400, 200)
    data = [(10, 20, 30, 42)]
    yAxis = YValueAxis()
    yAxis.setPosition(50, 50, 125)
    yAxis.configure(data)
    xAxis = XValueAxis()
    xAxis._length = 300
    xAxis.joinAxis = yAxis
    xAxis.joinAxisMode = 'top'
    xAxis.configure(data)
    drawing.add(xAxis)
    drawing.add(yAxis)
    return drawing
```

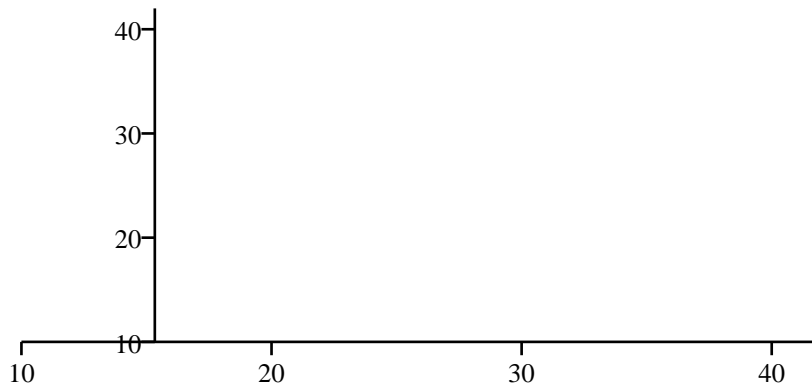


**sample5a( ... )**

Sample drawing, xvalue/yvalue axes, y connected at 100 pts to x.

*Example*

```
def sample5a():
    "Sample drawing, xvalue/yvalue axes, y connected at 100 pts to x."
    drawing = Drawing(400, 200)
    data = [(10, 20, 30, 42)]
    xAxis = XValueAxis()
    xAxis.setPosition(50, 50, 300)
    xAxis.configure(data)
    yAxis = YValueAxis()
    yAxis.setPosition(50, 50, 125)
    yAxis.joinAxis = xAxis
    yAxis.joinAxisMode = 'points'
    yAxis.joinAxisPos = 100
    yAxis.configure(data)
    drawing.add(xAxis)
    drawing.add(yAxis)
    return drawing
```

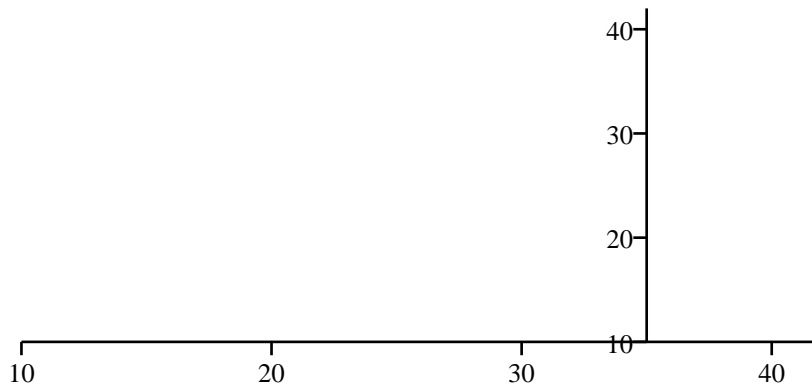


**sample5b( ... )**

Sample drawing, xvalue/yvalue axes, y connected at value 35 of x.

*Example*

```
def sample5b():
    "Sample drawing, xvalue/yvalue axes, y connected at value 35 of x."
    drawing = Drawing(400, 200)
    data = [(10, 20, 30, 42)]
    xAxis = XValueAxis()
    xAxis.setPosition(50, 50, 300)
    xAxis.configure(data)
    yAxis = YValueAxis()
    yAxis.setPosition(50, 50, 125)
    yAxis.joinAxis = xAxis
    yAxis.joinAxisMode = 'value'
    yAxis.joinAxisPos = 35
    yAxis.configure(data)
    drawing.add(xAxis)
    drawing.add(yAxis)
    return drawing
```

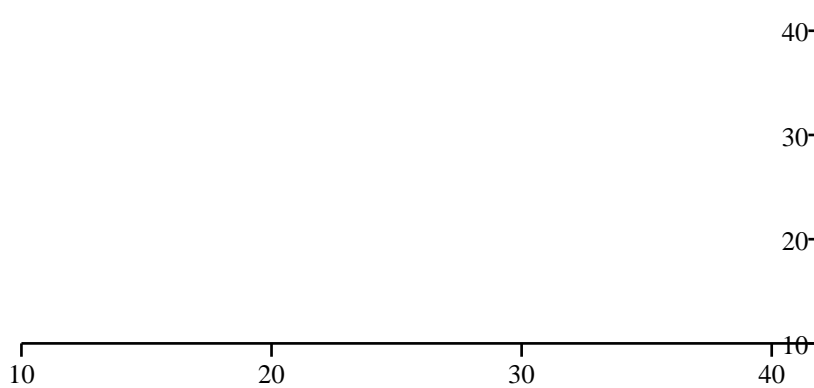


**sample5c( ... )**

Sample drawing, xvalue/yvalue axes, y connected at right of x.

*Example*

```
def sample5c():
    "Sample drawing, xvalue/yvalue axes, y connected at right of x."
    drawing = Drawing(400, 200)
    data = [(10, 20, 30, 42)]
    xAxis = XValueAxis()
    xAxis.setPosition(50, 50, 300)
    xAxis.configure(data)
    yAxis = YValueAxis()
    yAxis.setPosition(50, 50, 125)
    yAxis.joinAxis = xAxis
    yAxis.joinAxisMode = 'right'
    yAxis.configure(data)
    drawing.add(xAxis)
    drawing.add(yAxis)
    return drawing
```

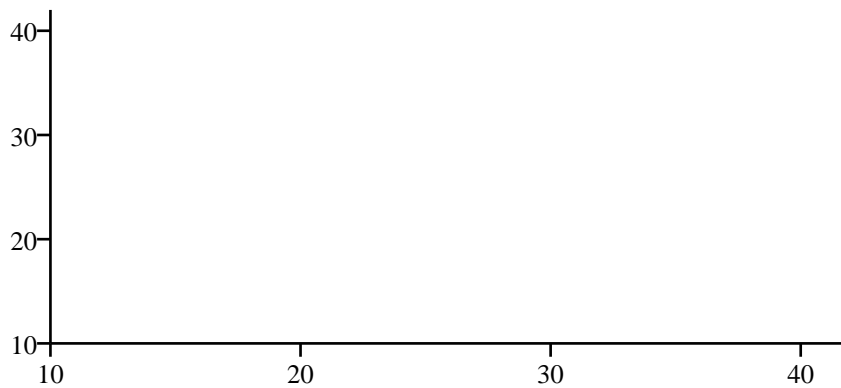


**sample5d( ... )**

Sample drawing, xvalue/yvalue axes, y connected at left of x.

*Example*

```
def sample5d():
    "Sample drawing, xvalue/yvalue axes, y connected at left of x."
    drawing = Drawing(400, 200)
    data = [(10, 20, 30, 42)]
    xAxis = XValueAxis()
    xAxis.setPosition(50, 50, 300)
    xAxis.configure(data)
    yAxis = YValueAxis()
    yAxis.setPosition(50, 50, 125)
    yAxis.joinAxis = xAxis
    yAxis.joinAxisMode = 'left'
    yAxis.configure(data)
    drawing.add(xAxis)
    drawing.add(yAxis)
    return drawing
```



**sample6a( ... )**

Sample drawing, xcat/yvalue axes, x connected at top of y.

*Example*

```
def sample6a():
    "Sample drawing, xcat/yvalue axes, x connected at top of y."
    drawing = Drawing(400, 200)
    data = [(10, 20, 30, 42)]
    yAxis = YValueAxis()
    yAxis.setPosition(50, 50, 125)
    yAxis.configure(data)
    xAxis = XCategoryAxis()
    xAxis._length = 300
    xAxis.configure(data)
    xAxis.joinAxis = yAxis
    xAxis.joinAxisMode = 'top'
    xAxis.categoryNames = ['Beer', 'Wine', 'Meat', 'Cannelloni']
    xAxis.labels.boxAnchor = 'n'
    drawing.add(xAxis)
    drawing.add(yAxis)
    return drawing
```



**sample6b( ... )**

Sample drawing, xcat/yvalue axes, x connected at bottom of y.

*Example*

```
def sample6b():
    "Sample drawing, xcat/yvalue axes, x connected at bottom of y."
    drawing = Drawing(400, 200)
    data = [(10, 20, 30, 42)]
    yAxis = YValueAxis()
    yAxis.setPosition(50, 50, 125)
    yAxis.configure(data)
    xAxis = XCategoryAxis()
    xAxis._length = 300
    xAxis.configure(data)
    xAxis.joinAxis = yAxis
    xAxis.joinAxisMode = 'bottom'
    xAxis.categoryNames = ['Beer', 'Wine', 'Meat', 'Cannelloni']
    xAxis.labels.boxAnchor = 'n'
    drawing.add(xAxis)
    drawing.add(yAxis)
    return drawing
```

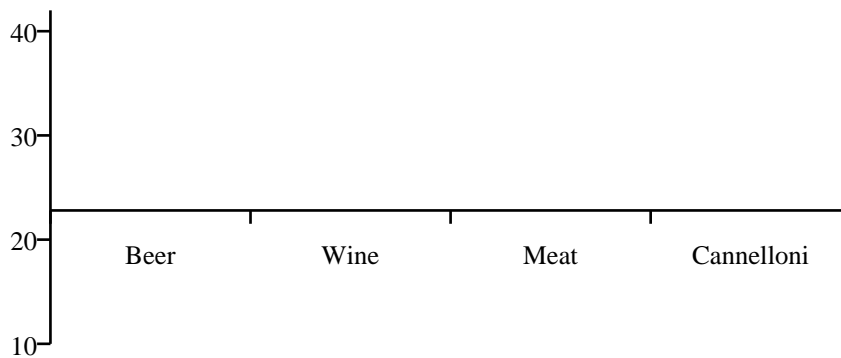


**sample6c( ... )**

Sample drawing, xcat/yvalue axes, x connected at 100 pts to y.

*Example*

```
def sample6c():
    "Sample drawing, xcat/yvalue axes, x connected at 100 pts to y."
    drawing = Drawing(400, 200)
    data = [(10, 20, 30, 42)]
    yAxis = YValueAxis()
    yAxis.setPosition(50, 50, 125)
    yAxis.configure(data)
    xAxis = XCategoryAxis()
    xAxis._length = 300
    xAxis.configure(data)
    xAxis.joinAxis = yAxis
    xAxis.joinAxisMode = 'points'
    xAxis.joinAxisPos = 100
    xAxis.categoryNames = ['Beer', 'Wine', 'Meat', 'Cannelloni']
    xAxis.labels.boxAnchor = 'n'
    drawing.add(xAxis)
    drawing.add(yAxis)
    return drawing
```



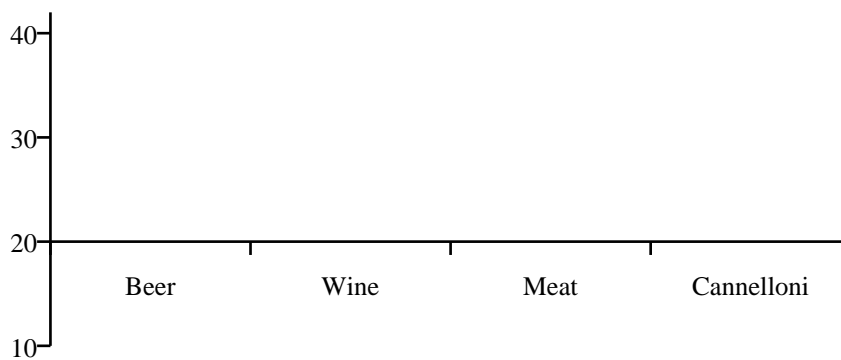


**sample6d( ... )**

Sample drawing, xcat/yvalue axes, x connected at value 20 of y.

*Example*

```
def sample6d():
    "Sample drawing, xcat/yvalue axes, x connected at value 20 of y."
    drawing = Drawing(400, 200)
    data = [(10, 20, 30, 42)]
    yAxis = YValueAxis()
    yAxis.setPosition(50, 50, 125)
    yAxis.configure(data)
    xAxis = XCategoryAxis()
    xAxis._length = 300
    xAxis.configure(data)
    xAxis.joinAxis = yAxis
    xAxis.joinAxisMode = 'value'
    xAxis.joinAxisPos = 20
    xAxis.categoryNames = ['Beer', 'Wine', 'Meat', 'Cannelloni']
    xAxis.labels.boxAnchor = 'n'
    drawing.add(xAxis)
    drawing.add(yAxis)
    return drawing
```

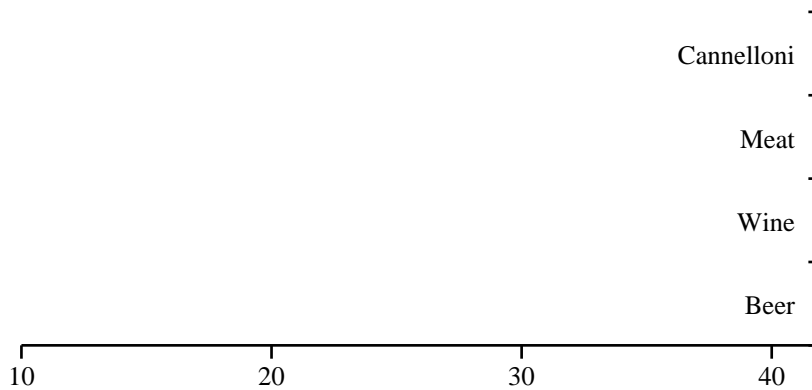


**sample7a( ... )**

Sample drawing, xvalue/ycat axes, y connected at right of x.

*Example*

```
def sample7a():
    "Sample drawing, xvalue/ycat axes, y connected at right of x."
    drawing = Drawing(400, 200)
    data = [(10, 20, 30, 42)]
    xAxis = XValueAxis()
    xAxis._length = 300
    xAxis.configure(data)
    yAxis = YCategoryAxis()
    yAxis.setPosition(50, 50, 125)
    yAxis.joinAxis = xAxis
    yAxis.joinAxisMode = 'right'
    yAxis.categoryNames = ['Beer', 'Wine', 'Meat', 'Cannelloni']
    yAxis.labels.boxAnchor = 'e'
    yAxis.configure(data)
    drawing.add(xAxis)
    drawing.add(yAxis)
    return drawing
```

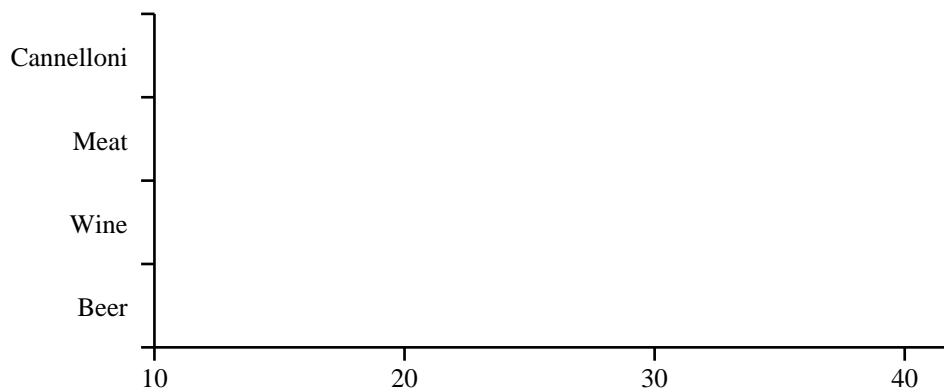


**sample7b( ... )**

Sample drawing, xvalue/ycat axes, y connected at left of x.

*Example*

```
def sample7b():
    "Sample drawing, xvalue/ycat axes, y connected at left of x."
    drawing = Drawing(400, 200)
    data = [(10, 20, 30, 42)]
    xAxis = XValueAxis()
    xAxis._length = 300
    xAxis.configure(data)
    yAxis = YCategoryAxis()
    yAxis.setPosition(50, 50, 125)
    yAxis.joinAxis = xAxis
    yAxis.joinAxisMode = 'left'
    yAxis.categoryNames = ['Beer', 'Wine', 'Meat', 'Cannelloni']
    yAxis.labels.boxAnchor = 'e'
    yAxis.configure(data)
    drawing.add(xAxis)
    drawing.add(yAxis)
    return drawing
```

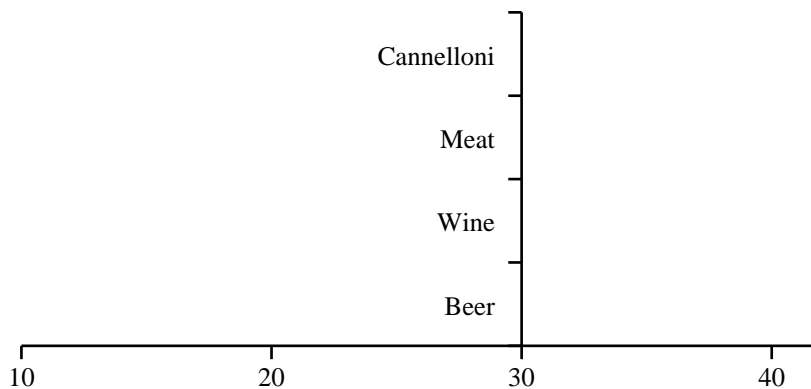


**sample7c( ... )**

Sample drawing, xvalue/ycat axes, y connected at value 30 of x.

*Example*

```
def sample7c():
    "Sample drawing, xvalue/ycat axes, y connected at value 30 of x."
    drawing = Drawing(400, 200)
    data = [(10, 20, 30, 42)]
    xAxis = XValueAxis()
    xAxis._length = 300
    xAxis.configure(data)
    yAxis = YCategoryAxis()
    yAxis.setPosition(50, 50, 125)
    yAxis.joinAxis = xAxis
    yAxis.joinAxisMode = 'value'
    yAxis.joinAxisPos = 30
    yAxis.categoryNames = ['Beer', 'Wine', 'Meat', 'Cannelloni']
    yAxis.labels.boxAnchor = 'e'
    yAxis.configure(data)
    drawing.add(xAxis)
    drawing.add(yAxis)
    return drawing
```

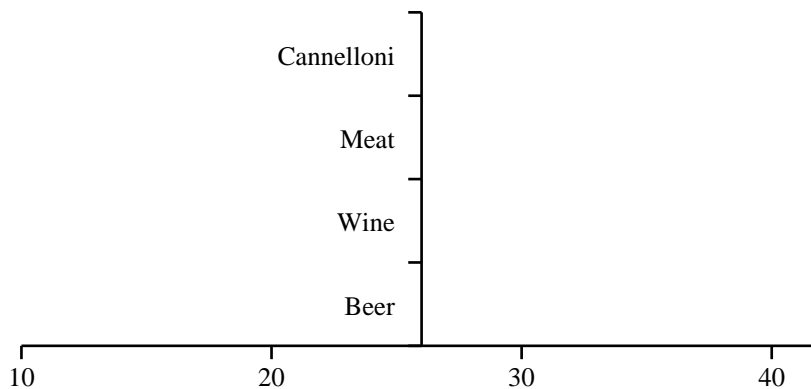


**sample7d( ... )**

Sample drawing, xvalue/ycat axes, y connected at 200 pts to x.

*Example*

```
def sample7d():
    "Sample drawing, xvalue/ycat axes, y connected at 200 pts to x."
    drawing = Drawing(400, 200)
    data = [(10, 20, 30, 42)]
    xAxis = XValueAxis()
    xAxis._length = 300
    xAxis.configure(data)
    yAxis = YCategoryAxis()
    yAxis.setPosition(50, 50, 125)
    yAxis.joinAxis = xAxis
    yAxis.joinAxisMode = 'points'
    yAxis.joinAxisPos = 200
    yAxis.categoryNames = ['Beer', 'Wine', 'Meat', 'Cannelloni']
    yAxis.labels.boxAnchor = 'e'
    yAxis.configure(data)
    drawing.add(xAxis)
    drawing.add(yAxis)
    return drawing
```



## areas

This module defines a Area mixin classes

## Classes

### PlotArea(Widget)

Abstract base class representing a chart's plot area, pretty unusable by itself.

### Public Attributes

**background** Handle to background object e.g. Rect(0,0,width,height).

**debug** Used only for debugging.

**fillColor** Color of the plot area interior.

**height** Height of the chart.

**strokeColor** Color of the plot area border.

**strokeWidth** Width plot area border.

**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**y** Y position of the lower-left corner of the chart.

### Example

```
def demo(self):
    msg = "demo() must be implemented for each Widget!"
    raise shapes.NotImplementedError, msg
```

### Properties of Example Widget

```
background = None
debug = 0
fillColor = None
height = 85
strokeColor = None
strokeWidth = 1
width = 180
x = 20
y = 10
```

## slidebox

### Classes

#### **SlideBox(Widget)**

Returns a slidebox widget

#### **Public Attributes**

**background** Colour of the background to the drawing (if any)

**bottomPadding** Padding at bottom of drawing

**boxHeight** Height of the boxes

**boxOutlineColor** Colour used to outline the boxes (if any)

**boxOutlineWidth** Width of the box outline (if any)

**boxSpacing** Space between the boxes

**boxWidth** Width of the boxes

**endColor** Color of last box

**labelFillColor** Colour for number insides

**labelFontName** Name of font used for the labels

**labelFontSize** Size of font used for the labels

**labelStrokeColor** Colour for for number outlines

**leftPadding** Padding on left of drawing

**numberOfBoxes** How many boxes there are

**rightPadding** Padding on right of drawing

**sourceLabelFillColor** Colour ink for the 'source' label (bottom right)

**sourceLabelFontName** Name of font used for the 'source' label

**sourceLabelFontSize** Font size for the 'source' label

**sourceLabelOffset** Padding at bottom of drawing

**sourceLabelText** Text used for the 'source' label (can be empty)

**startColor** Color of first box

**topPadding** Padding at top of drawing

**triangleFillColor** Colour of indicator triangles

**triangleHeight** Height of indicator triangles

**trianglePosition** Which box is highlighted by the triangles

**triangleStrokeColor** Colour of indicator triangle outline

**triangleStrokeWidth** Colour of indicator triangle outline

**triangleWidth** Width of indicator triangles

#### *Example*

```
def demo(self,drawing=None):
```

```
from reportlab.lib import colors
if not drawing:
    tx,ty=self._getDrawingDimensions()
    drawing = Drawing(tx,ty)
drawing.add(self.draw())
return drawing
```



Source: ReportLab

### *Properties of Example Widget*

```
background = None
bottomPadding = 5
boxHeight = 15.590551181102363
boxOutlineColor = Color(0,0,0,1)
boxOutlineWidth = 0.58
boxSpacing = 2.1259842519685037
boxWidth = 20.69291338582677
endColor = Color(.098039,.301961,.529412,1)
labelFillColor = Color(1,1,1,1)
labelFontName = 'Helvetica-Bold'
labelFontSize = 10
labelStrokeColor = Color(0,0,0,1)
leftPadding = 5
numberOfBoxes = 7
rightPadding = 5
sourceLabelFillColor = Color(0,0,0,1)
sourceLabelFontName = 'Helvetica-Oblique'
sourceLabelFontSize = 6
sourceLabelOffset = 5.669291338582678
sourceLabelText = 'Source: ReportLab'
startColor = Color(.909804,.878431,.466667,1)
topPadding = 5
triangleFillColor = Color(1,1,1,1)
triangleHeight = 3.401574803149606
trianglePosition = 7
triangleStrokeColor = Color(0,0,0,1)
triangleStrokeWidth = 0.58
triangleWidth = 10.771653543307087
```



## linecharts

This module defines a very preliminary Line Chart example.

### Classes

#### **AbstractLineChart (PlotArea)**

##### **Public Attributes**

**background** Handle to background object e.g. Rect(0,0,width,height).

**debug** Used only for debugging.

**fillColor** Color of the plot area interior.

**height** Height of the chart.

**strokeColor** Color of the plot area border.

**strokeWidth** Width plot area border.

**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**y** Y position of the lower-left corner of the chart.

##### *Example*

```
def demo(self):
    msg = "demo() must be implemented for each Widget!"
    raise shapes.NotImplementedError, msg
```

##### *Properties of Example Widget*

```
background = None
debug = 0
fillColor = None
height = 85
strokeColor = None
strokeWidth = 1
width = 180
x = 20
y = 10
```

## HorizontalLineChart (LineChart)

Line chart with multiple lines.

A line chart is assumed to have one category and one value axis. Despite its generic name this particular line chart class has a vertical value axis and a horizontal category one. It may evolve into individual horizontal and vertical variants (like with the existing bar charts).

Available attributes are:

x: x-position of lower-left chart origin

y: y-position of lower-left chart origin

width: chart width

height: chart height

useAbsolute: disables auto-scaling of chart elements (?)

lineLabelNudge: distance of data labels to data points

lineLabels: labels associated with data values

lineLabelFormat: format string or callback function

groupSpacing: space between categories

joinedLines: enables drawing of lines

strokeColor: color of chart lines (?)

fillColor: color for chart background (?)

lines: style list, used cyclically for data series

valueAxis: value axis object

categoryAxis: category axis object

categoryNames: category names

data: chart data, a list of data series of equal length

## Public Attributes

**annotations** list of callables, will be called with self, xscale, yscale.

**background** Handle to background object e.g. Rect(0,0,width,height).

**categoryAxis** Handle of the category axis.

**categoryNames** List of category names.

**data** Data to be plotted, list of (lists of) numbers.

**debug** Used only for debugging.

**fillColor** Color of the plot area interior.

**groupSpacing** ? - Likely to disappear.

**height** Height of the chart.

**inFill** Whether infilling should be done.

**joinedLines** Display data points joined with lines if true.

**lineLabelArray** explicit array of line label values, must match size of data if present.

**lineLabelFormat** Formatting string or function used for data point labels.

**lineLabelNudge** Distance between a data point and its label.

**lineLabels** Handle to the list of data point labels.

**lines** Handle of the lines.

**reversePlotOrder** If true reverse plot order.

**strokeColor** Color of the plot area border.

**strokeWidth** Width plot area border.

**useAbsolute** Flag to use absolute spacing values.

**valueAxis** Handle of the value axis.

**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**y** Y position of the lower-left corner of the chart.

### *Example*

```
def demo(self):
    """Shows basic use of a line chart."""

    drawing = Drawing(200, 100)

    data = [
        (13, 5, 20, 22, 37, 45, 19, 4),
        (14, 10, 21, 28, 38, 46, 25, 5)
    ]

    lc = HorizontalLineChart()

    lc.x = 20
    lc.y = 10
    lc.height = 85
    lc.width = 170
    lc.data = data
    lc.lines.symbol = makeMarker('Circle')

    drawing.add(lc)

    return drawing
```

### *Properties of Example Widget*

```
background = None
categoryAxis.categoryNames = None
categoryAxis.drawGridLast = False
categoryAxis.gridEnd = None
categoryAxis.gridStart = None
categoryAxis.gridStrokeColor = Color(0,0,0,1)
categoryAxis.gridStrokeDashArray = None
categoryAxis.gridStrokeLineCap = 0
categoryAxis.gridStrokeLineJoin = 0
categoryAxis.gridStrokeMiterLimit = 10
categoryAxis.gridStrokeWidth = 0.25
categoryAxis.hiLLen = 0
categoryAxis.hiPad = 0
categoryAxis.joinAxis = None
categoryAxis.joinAxisMode = None
categoryAxis.joinAxisPos = None
categoryAxis.labelAxisMode = 'axis'
categoryAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x277ddd0>
categoryAxis.loLLen = 0
categoryAxis.loPad = 0
categoryAxis.reverseDirection = 0
categoryAxis.strokeColor = Color(0,0,0,1)
categoryAxis.strokeDashArray = None
categoryAxis.strokeLineCap = 0
categoryAxis.strokeLineJoin = 0
categoryAxis.strokeMiterLimit = 10
categoryAxis.strokeWidth = 1
```

```
categoryAxis.style = 'parallel'
categoryAxis.tickDown = 5
categoryAxis.tickShift = 0
categoryAxis.tickUp = 0
categoryAxis.visible = 1
categoryAxis.visibleAxis = 1
categoryAxis.visibleGrid = 0
categoryAxis.visibleLabels = 1
categoryAxis.visibleTicks = 1
categoryNames = ('North', 'South', 'East', 'West')
data = [(100, 110, 120, 130), (70, 80, 80, 90)]
debug = 0
fillColor = None
groupSpacing = 1
height = 85
inFill = 0
joinedLines = 1
lineLabelArray = None
lineLabelFormat = None
lineLabelNudge = 10
lineLabels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x277dfc8>
lines = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x277df38>
reversePlotOrder = 0
strokeColor = None
strokeWidth = 1
useAbsolute = 0
valueAxis.abf_ignore_zero = False
valueAxis.avoidBoundFrac = None
valueAxis.avoidBoundSpace = None
valueAxis.drawGridLast = False
valueAxis.forceZero = 0
valueAxis.gridEnd = None
valueAxis.gridStart = None
valueAxis.gridStrokeColor = Color(0,0,0,1)
valueAxis.gridStrokeDashArray = None
valueAxis.gridStrokeLineCap = 0
valueAxis.gridStrokeLineJoin = 0
valueAxis.gridStrokeMiterLimit = 10
valueAxis.gridStrokeWidth = 0.25
valueAxis.hilLen = 0
valueAxis.joinAxis = None
valueAxis.joinAxisMode = None
valueAxis.joinAxisPos = None
valueAxis.keepTickLabelsInside = 0
valueAxis.labelAxisMode = 'axis'
valueAxis.labelTextFormat = None
valueAxis.labelTextPostFormat = None
valueAxis.labelTextScale = None
valueAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x277dea8>
valueAxis.loLen = 0
valueAxis.maximumTicks = 7
valueAxis.minimumTickSpacing = 10
valueAxis.origShiftIPC = None
valueAxis.origShiftMin = None
valueAxis.origShiftSpecialValue = None
valueAxis.rangeRound = 'none'
valueAxis.requiredRange = 0
valueAxis.reverseDirection = 0
valueAxis.skipEndL = 'none'
valueAxis.strokeColor = Color(0,0,0,1)
valueAxis.strokeDashArray = None
valueAxis.strokeLineCap = 0
valueAxis.strokeLineJoin = 0
valueAxis.strokeMiterLimit = 10
valueAxis.strokeWidth = 1
valueAxis.style = 'normal'
valueAxis.subGridEnd = None
valueAxis.subGridStart = None
valueAxis.subGridStrokeColor = Color(0,0,0,1)
valueAxis.subGridStrokeDashArray = None
valueAxis.subGridStrokeLineCap = 0
valueAxis.subGridStrokeLineJoin = 0
valueAxis.subGridStrokeMiterLimit = 10
valueAxis.subGridStrokeWidth = 0.25
valueAxis.subTickHi = 0
valueAxis.subTickLo = 0
valueAxis.subTickNum = 0
valueAxis.tickAxisMode = 'axis'
valueAxis.tickLeft = 5
```

```
valueAxis.tickRight = 0
valueAxis.valueMax = None
valueAxis.valueMin = None
valueAxis.valueStep = None
valueAxis.visible = 1
valueAxis.visibleAxis = 1
valueAxis.visibleGrid = 0
valueAxis.visibleLabels = 1
valueAxis.visibleSubGrid = 0
valueAxis.visibleSubTicks = 0
valueAxis.visibleTicks = 1
valueAxis.zrangePref = 0
width = 180
x = 20
y = 10
```

## HorizontalLineChart3D(HorizontalLineChart)

### Public Attributes

**annotations** list of callables, will be called with self, xscale, yscale.

**background** Handle to background object e.g. Rect(0,0,width,height).

**categoryAxis** Handle of the category axis.

**categoryNames** List of category names.

**data** Data to be plotted, list of (lists of) numbers.

**debug** Used only for debugging.

**fillColor** Color of the plot area interior.

**groupSpacing** ? - Likely to disappear.

**height** Height of the chart.

**inFill** Whether infilling should be done.

**joinedLines** Display data points joined with lines if true.

**lineLabelArray** explicit array of line label values, must match size of data if present.

**lineLabelFormat** Formatting string or function used for data point labels.

**lineLabelNudge** Distance between a data point and its label.

**lineLabels** Handle to the list of data point labels.

**lines** Handle of the lines.

**reversePlotOrder** If true reverse plot order.

**strokeColor** Color of the plot area border.

**strokeWidth** Width plot area border.

**theta\_x**  $dx/dz$

**theta\_y**  $dy/dz$

**useAbsolute** Flag to use absolute spacing values.

**valueAxis** Handle of the value axis.

**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**y** Y position of the lower-left corner of the chart.

**zDepth** depth of an individual series

**zSpace** z gap around series

### Example

```
def demo(self):
    """Shows basic use of a line chart."""

    drawing = Drawing(200, 100)

    data = [
        (13, 5, 20, 22, 37, 45, 19, 4),
        (14, 10, 21, 28, 38, 46, 25, 5)
    ]

    lc = HorizontalLineChart()
    lc.x = 20
```

```
lc.y = 10
lc.height = 85
lc.width = 170
lc.data = data
lc.lines.symbol = makeMarker('Circle')

drawing.add(lc)

return drawing
```

### *Properties of Example Widget*

```
background = None
categoryAxis.categoryNames = None
categoryAxis.drawGridLast = False
categoryAxis.gridEnd = None
categoryAxis.gridStart = None
categoryAxis.gridStrokeColor = Color(0,0,0,1)
categoryAxis.gridStrokeDashArray = None
categoryAxis.gridStrokeLineCap = 0
categoryAxis.gridStrokeLineJoin = 0
categoryAxis.gridStrokeMiterLimit = 10
categoryAxis.gridStrokeWidth = 0.25
categoryAxis.hiLLen = 0
categoryAxis.hiPad = 0
categoryAxis.joinAxis = None
categoryAxis.joinAxisMode = None
categoryAxis.joinAxisPos = None
categoryAxis.labelAxisMode = 'axis'
categoryAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x2795368>
categoryAxis.loLLen = 0
categoryAxis.loPad = 0
categoryAxis.reverseDirection = 0
categoryAxis.strokeColor = Color(0,0,0,1)
categoryAxis.strokeDashArray = None
categoryAxis.strokeLineCap = 0
categoryAxis.strokeLineJoin = 0
categoryAxis.strokeMiterLimit = 10
categoryAxis.strokeWidth = 1
categoryAxis.style = 'parallel'
categoryAxis.tickDown = 5
categoryAxis.tickShift = 0
categoryAxis.tickUp = 0
categoryAxis.visible = 1
categoryAxis.visibleAxis = 1
categoryAxis.visibleGrid = 0
categoryAxis.visibleLabels = 1
categoryAxis.visibleTicks = 1
categoryNames = ('North', 'South', 'East', 'West')
data = [(100, 110, 120, 130), (70, 80, 80, 90)]
debug = 0
fillColor = None
groupSpacing = 1
height = 85
inFill = 0
joinedLines = 1
lineLabelArray = None
lineLabelFormat = None
lineLabelNudge = 10
lineLabels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x2795560>
lines = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x27954d0>
reversePlotOrder = 0
strokeColor = None
strokeWidth = 1
useAbsolute = 0
valueAxis.abf_ignore_zero = False
valueAxis.avoidBoundFrac = None
valueAxis.avoidBoundSpace = None
valueAxis.drawGridLast = False
valueAxis.forceZero = 0
valueAxis.gridEnd = None
valueAxis.gridStart = None
valueAxis.gridStrokeColor = Color(0,0,0,1)
valueAxis.gridStrokeDashArray = None
valueAxis.gridStrokeLineCap = 0
valueAxis.gridStrokeLineJoin = 0
```

```
valueAxis.gridStrokeMiterLimit = 10
valueAxis.gridStrokeWidth = 0.25
valueAxis.hiLLen = 0
valueAxis.joinAxis = None
valueAxis.joinAxisMode = None
valueAxis.joinAxisPos = None
valueAxis.keepTickLabelsInside = 0
valueAxis.labelAxisMode = 'axis'
valueAxis.labelTextFormat = None
valueAxis.labelTextPostFormat = None
valueAxis.labelTextScale = None
valueAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x2795440>
valueAxis.loLLen = 0
valueAxis.maximumTicks = 7
valueAxis.minimumTickSpacing = 10
valueAxis.origShiftIPC = None
valueAxis.origShiftMin = None
valueAxis.origShiftSpecialValue = None
valueAxis.rangeRound = 'none'
valueAxis.requiredRange = 0
valueAxis.reverseDirection = 0
valueAxis.skipEndL = 'none'
valueAxis.strokeColor = Color(0,0,0,1)
valueAxis.strokeDashArray = None
valueAxis.strokeLineCap = 0
valueAxis.strokeLineJoin = 0
valueAxis.strokeMiterLimit = 10
valueAxis.strokeWidth = 1
valueAxis.style = 'normal'
valueAxis.subGridEnd = None
valueAxis.subGridStart = None
valueAxis.subGridStrokeColor = Color(0,0,0,1)
valueAxis.subGridStrokeDashArray = None
valueAxis.subGridStrokeLineCap = 0
valueAxis.subGridStrokeLineJoin = 0
valueAxis.subGridStrokeMiterLimit = 10
valueAxis.subGridStrokeWidth = 0.25
valueAxis.subTickHi = 0
valueAxis.subTickLo = 0
valueAxis.subTickNum = 0
valueAxis.tickAxisMode = 'axis'
valueAxis.tickLeft = 5
valueAxis.tickRight = 0
valueAxis.valueMax = None
valueAxis.valueMin = None
valueAxis.valueStep = None
valueAxis.visible = 1
valueAxis.visibleAxis = 1
valueAxis.visibleGrid = 0
valueAxis.visibleLabels = 1
valueAxis.visibleSubGrid = 0
valueAxis.visibleSubTicks = 0
valueAxis.visibleTicks = 1
valueAxis.zrangePref = 0
width = 180
x = 20
y = 10
```



## LineChart (AbstractLineChart)

### Public Attributes

**background** Handle to background object e.g. Rect(0,0,width,height).

**debug** Used only for debugging.

**fillColor** Color of the plot area interior.

**height** Height of the chart.

**strokeColor** Color of the plot area border.

**strokeWidth** Width plot area border.

**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**y** Y position of the lower-left corner of the chart.

### Example

```
def demo(self):
    msg = "demo() must be implemented for each Widget!"
    raise shapes.NotImplementedError, msg
```

### Properties of Example Widget

```
background = None
debug = 0
fillColor = None
height = 85
strokeColor = None
strokeWidth = 1
width = 180
x = 20
y = 10
```

## **SampleHorizontalLineChart(HorizontalLineChart)**

Sample class overwriting one method to draw additional horizontal lines.

### **Public Attributes**

**annotations** list of callables, will be called with self, xscale, yscale.

**background** Handle to background object e.g. Rect(0,0,width,height).

**categoryAxis** Handle of the category axis.

**categoryNames** List of category names.

**data** Data to be plotted, list of (lists of) numbers.

**debug** Used only for debugging.

**fillColor** Color of the plot area interior.

**groupSpacing** ? - Likely to disappear.

**height** Height of the chart.

**inFill** Whether infilling should be done.

**joinedLines** Display data points joined with lines if true.

**lineLabelArray** explicit array of line label values, must match size of data if present.

**lineLabelFormat** Formatting string or function used for data point labels.

**lineLabelNudge** Distance between a data point and its label.

**lineLabels** Handle to the list of data point labels.

**lines** Handle of the lines.

**reversePlotOrder** If true reverse plot order.

**strokeColor** Color of the plot area border.

**strokeWidth** Width plot area border.

**useAbsolute** Flag to use absolute spacing values.

**valueAxis** Handle of the value axis.

**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**y** Y position of the lower-left corner of the chart.

### *Example*

```
def demo(self):
    """Shows basic use of a line chart."""

    drawing = Drawing(200, 100)

    data = [
        (13, 5, 20, 22, 37, 45, 19, 4),
        (14, 10, 21, 28, 38, 46, 25, 5)
    ]

    lc = SampleHorizontalLineChart()

    lc.x = 20
    lc.y = 10
    lc.height = 85
    lc.width = 170
    lc.data = data
```

```
lc.strokeColor = colors.white
lc.fillColor = colors.HexColor(0xCCCCCC)

drawing.add(lc)

return drawing
```

### *Properties of Example Widget*

```
background = None
categoryAxis.categoryNames = None
categoryAxis.drawGridLast = False
categoryAxis.gridEnd = None
categoryAxis.gridStart = None
categoryAxis.gridStrokeColor = Color(0,0,0,1)
categoryAxis.gridStrokeDashArray = None
categoryAxis.gridStrokeLineCap = 0
categoryAxis.gridStrokeLineJoin = 0
categoryAxis.gridStrokeMiterLimit = 10
categoryAxis.gridStrokeWidth = 0.25
categoryAxis.hiLLen = 0
categoryAxis.hiPad = 0
categoryAxis.joinAxis = None
categoryAxis.joinAxisMode = None
categoryAxis.joinAxisPos = None
categoryAxis.labelAxisMode = 'axis'
categoryAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x27aadd0>
categoryAxis.loLLen = 0
categoryAxis.loPad = 0
categoryAxis.reverseDirection = 0
categoryAxis.strokeColor = Color(0,0,0,1)
categoryAxis.strokeDashArray = None
categoryAxis.strokeLineCap = 0
categoryAxis.strokeLineJoin = 0
categoryAxis.strokeMiterLimit = 10
categoryAxis.strokeWidth = 1
categoryAxis.style = 'parallel'
categoryAxis.tickDown = 5
categoryAxis.tickShift = 0
categoryAxis.tickUp = 0
categoryAxis.visible = 1
categoryAxis.visibleAxis = 1
categoryAxis.visibleGrid = 0
categoryAxis.visibleLabels = 1
categoryAxis.visibleTicks = 1
categoryNames = ('North', 'South', 'East', 'West')
data = [(100, 110, 120, 130), (70, 80, 80, 90)]
debug = 0
fillColor = None
groupSpacing = 1
height = 85
inFill = 0
joinedLines = 1
lineLabelArray = None
lineLabelFormat = None
lineLabelNudge = 10
lineLabels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x27aafc8>
lines = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x27aaf38>
reversePlotOrder = 0
strokeColor = None
strokeWidth = 1
useAbsolute = 0
valueAxis.abf_ignore_zero = False
valueAxis.avoidBoundFrac = None
valueAxis.avoidBoundSpace = None
valueAxis.drawGridLast = False
valueAxis.forceZero = 0
valueAxis.gridEnd = None
valueAxis.gridStart = None
valueAxis.gridStrokeColor = Color(0,0,0,1)
valueAxis.gridStrokeDashArray = None
valueAxis.gridStrokeLineCap = 0
valueAxis.gridStrokeLineJoin = 0
valueAxis.gridStrokeMiterLimit = 10
valueAxis.gridStrokeWidth = 0.25
valueAxis.hiLLen = 0
```

```
valueAxis.joinAxis = None
valueAxis.joinAxisMode = None
valueAxis.joinAxisPos = None
valueAxis.keepTickLabelsInside = 0
valueAxis.labelAxisMode = 'axis'
valueAxis.labelTextFormat = None
valueAxis.labelTextPostFormat = None
valueAxis.labelTextScale = None
valueAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x27aaea8>
valueAxis.loLlen = 0
valueAxis.maximumTicks = 7
valueAxis.minimumTickSpacing = 10
valueAxis.origShiftIPC = None
valueAxis.origShiftMin = None
valueAxis.origShiftSpecialValue = None
valueAxis.rangeRound = 'none'
valueAxis.requiredRange = 0
valueAxis.reverseDirection = 0
valueAxis.skipEndL = 'none'
valueAxis.strokeColor = Color(0,0,0,1)
valueAxis.strokeDashArray = None
valueAxis.strokeLineCap = 0
valueAxis.strokeLineJoin = 0
valueAxis.strokeMiterLimit = 10
valueAxis.strokeWidth = 1
valueAxis.style = 'normal'
valueAxis.subGridEnd = None
valueAxis.subGridStart = None
valueAxis.subGridStrokeColor = Color(0,0,0,1)
valueAxis.subGridStrokeDashArray = None
valueAxis.subGridStrokeLineCap = 0
valueAxis.subGridStrokeLineJoin = 0
valueAxis.subGridStrokeMiterLimit = 10
valueAxis.subGridStrokeWidth = 0.25
valueAxis.subTickHi = 0
valueAxis.subTickLo = 0
valueAxis.subTickNum = 0
valueAxis.tickAxisMode = 'axis'
valueAxis.tickLeft = 5
valueAxis.tickRight = 0
valueAxis.valueMax = None
valueAxis.valueMin = None
valueAxis.valueStep = None
valueAxis.visible = 1
valueAxis.visibleAxis = 1
valueAxis.visibleGrid = 0
valueAxis.visibleLabels = 1
valueAxis.visibleSubGrid = 0
valueAxis.visibleSubTicks = 0
valueAxis.visibleTicks = 1
valueAxis.zrangePref = 0
width = 180
x = 20
y = 10
```

## VerticalLineChart (LineChart)

### Public Attributes

**background** Handle to background object e.g. Rect(0,0,width,height).

**debug** Used only for debugging.

**fillColor** Color of the plot area interior.

**height** Height of the chart.

**strokeColor** Color of the plot area border.

**strokeWidth** Width plot area border.

**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**y** Y position of the lower-left corner of the chart.

### Example

```
def demo(self):
    msg = "demo() must be implemented for each Widget!"
    raise shapes.NotImplementedError, msg
```

### Properties of Example Widget

```
background = None
debug = 0
fillColor = None
height = 85
strokeColor = None
strokeWidth = 1
width = 180
x = 20
y = 10
```

## Functions

**sample1( ... )**

### Example

```
def sample1():
    drawing = Drawing(400, 200)

    data = [
        (13, 5, 20, 22, 37, 45, 19, 4),
        (5, 20, 46, 38, 23, 21, 6, 14)
    ]

    lc = HorizontalLineChart()

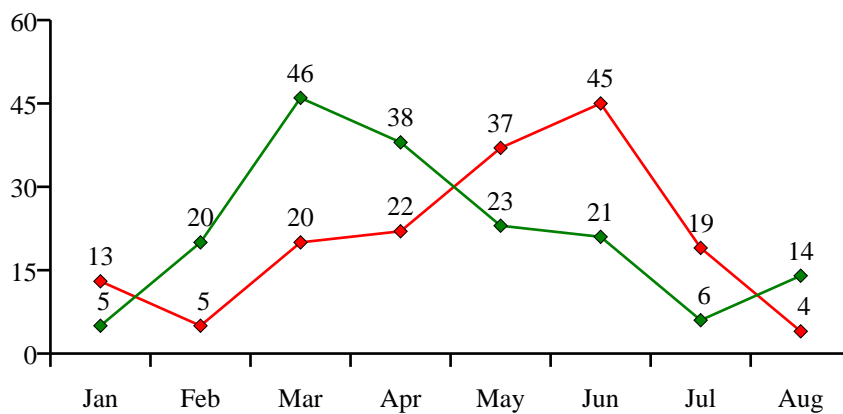
    lc.x = 50
    lc.y = 50
    lc.height = 125
    lc.width = 300
    lc.data = data
    lc.joinedLines = 1
    lc.lines.symbol = makeMarker('FilledDiamond')
    lc.lineLabelFormat = '%2.0f'

    catNames = 'Jan Feb Mar Apr May Jun Jul Aug'.split(' ')
    lc.categoryAxis.categoryNames = catNames
    lc.categoryAxis.labels.boxAnchor = 'n'

    lc.valueAxis.valueMin = 0
    lc.valueAxis.valueMax = 60
    lc.valueAxis.valueStep = 15

    drawing.add(lc)

    return drawing
```



**sample1a( ... )***Example*

```
def sample1a():
    drawing = Drawing(400, 200)

    data = [
        (13, 5, 20, 22, 37, 45, 19, 4),
        (5, 20, 46, 38, 23, 21, 6, 14)
    ]

    lc = SampleHorizontalLineChart()

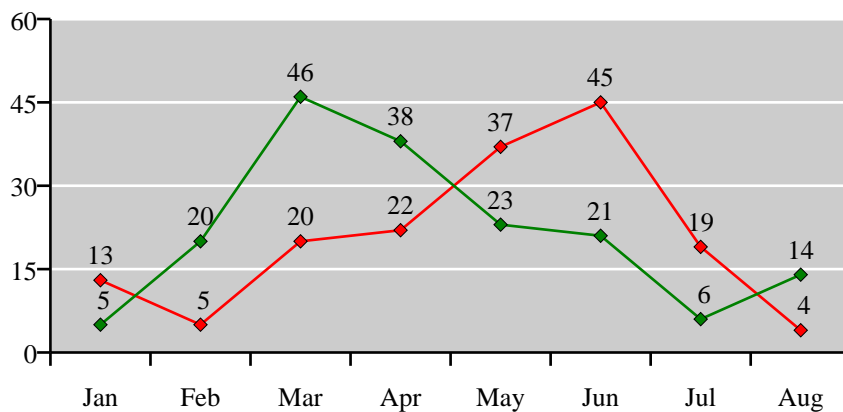
    lc.x = 50
    lc.y = 50
    lc.height = 125
    lc.width = 300
    lc.data = data
    lc.joinedLines = 1
    lc.strokeColor = colors.white
    lc.fillColor = colors.HexColor(0xCCCCCC)
    lc.lines.symbol = makeMarker('FilledDiamond')
    lc.lineLabelFormat = '%2.0f'

    catNames = 'Jan Feb Mar Apr May Jun Jul Aug'.split(' ')
    lc.categoryAxis.categoryNames = catNames
    lc.categoryAxis.labels.boxAnchor = 'n'

    lc.valueAxis.valueMin = 0
    lc.valueAxis.valueMax = 60
    lc.valueAxis.valueStep = 15

    drawing.add(lc)

    return drawing
```



**sample2( ... )***Example*

```
def sample2():
    drawing = Drawing(400, 200)

    data = [
        (13, 5, 20, 22, 37, 45, 19, 4),
        (5, 20, 46, 38, 23, 21, 6, 14)
    ]

    lc = HorizontalLineChart()

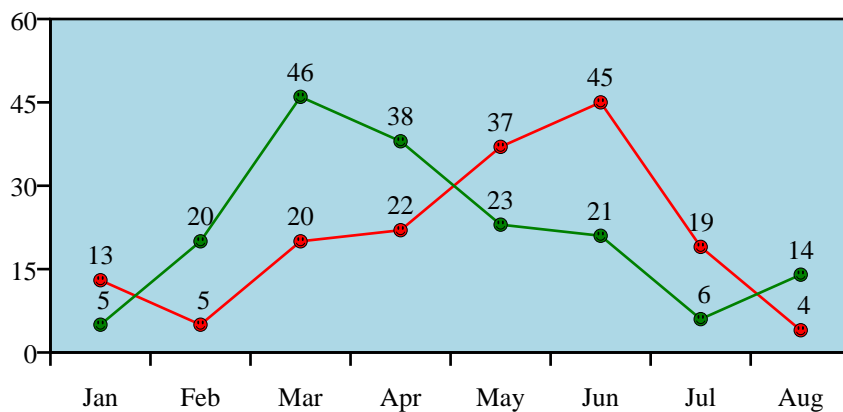
    lc.x = 50
    lc.y = 50
    lc.height = 125
    lc.width = 300
    lc.data = data
    lc.joinedLines = 1
    lc.lines.symbol = makeMarker('Smiley')
    lc.lineLabelFormat = '%2.0f'
    lc.strokeColor = colors.black
    lc.fillColor = colors.lightblue

    catNames = 'Jan Feb Mar Apr May Jun Jul Aug'.split(' ')
    lc.categoryAxis.categoryNames = catNames
    lc.categoryAxis.labels.boxAnchor = 'n'

    lc.valueAxis.valueMin = 0
    lc.valueAxis.valueMax = 60
    lc.valueAxis.valueStep = 15

    drawing.add(lc)

    return drawing
```





**sample3( ... )***Example*

```
def sample3():
    drawing = Drawing(400, 200)

    data = [
        (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 = data
    lc.joinedLines = 1
    lc.lineLabelFormat = '%2.0f'
    lc.strokeColor = colors.black

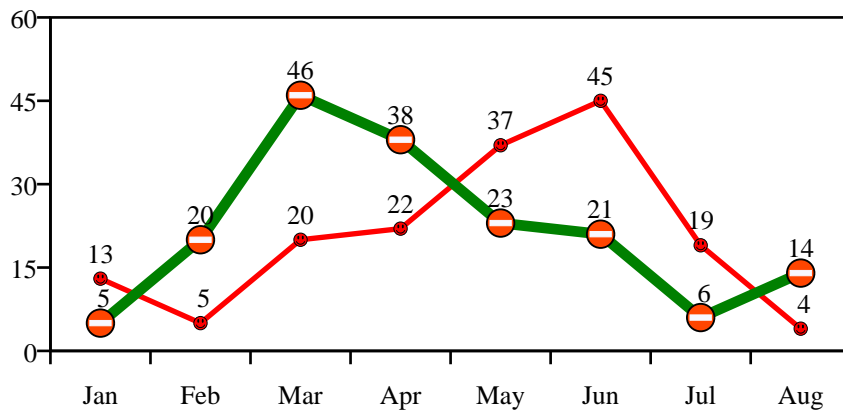
    lc.lines[0].symbol = makeMarker('Smiley')
    lc.lines[1].symbol = NoEntry
    lc.lines[0].strokeWidth = 2
    lc.lines[1].strokeWidth = 4

    catNames = 'Jan Feb Mar Apr May Jun Jul Aug'.split(' ')
    lc.categoryAxis.categoryNames = catNames
    lc.categoryAxis.labels.boxAnchor = 'n'

    lc.valueAxis.valueMin = 0
    lc.valueAxis.valueMax = 60
    lc.valueAxis.valueStep = 15

    drawing.add(lc)

    return drawing
```



## piecharts

Basic Pie Chart class.

This permits you to customize and pop out individual wedges;  
supports elliptical and circular pies.

## Classes

### AbstractPieChart (PlotArea)

#### Public Attributes

**background** Handle to background object e.g. Rect(0,0,width,height).

**debug** Used only for debugging.

**fillColor** Color of the plot area interior.

**height** Height of the chart.

**strokeColor** Color of the plot area border.

**strokeWidth** Width plot area border.

**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**y** Y position of the lower-left corner of the chart.

#### Example

```
def demo(self):  
    msg = "demo() must be implemented for each Widget!"  
    raise shapes.NotImplementedError, msg
```

#### Properties of Example Widget

```
background = None  
debug = 0  
fillColor = None  
height = 85  
strokeColor = None  
strokeWidth = 1  
width = 180  
x = 20  
y = 10
```

## LegendedPie(Pie)

Pie with a two part legend (one editable with swatches, one hidden without swatches).

### Public Attributes

**background** Handle to background object e.g. Rect(0,0,width,height).

**bottomPadding** Padding at bottom of drawing

**checkLabelOverlap** If true check and attempt to fix standard label overlaps(default off)

**data** List of numbers defining wedge sizes; need not sum to 1

**debug** Used only for debugging.

**direction** 'clockwise' or 'anticlockwise'

**drawLegend** If true then create and draw legend

**fillColor** Color of the plot area interior.

**height** Height of the chart.

**innerRadiusFraction** fraction of radii to start wedges at

**labels** Optional list of labels to use for each data point

**leftPadding** Padding on left of drawing

**legend1** Handle to legend for pie

**legendNumberFormat** Formatting routine for number on right hand side of legend.

**legendNumberOffset** Horizontal space between legend and numbers on r/hand side

**legend\_data** Numbers used on r/hand side of legend (or None)

**legend\_names** Names used in legend (or None)

**orderMode** None

**other\_threshold** A value for doing thresholding, not used yet.

**pieAndLegend\_colors** Colours used for both swatches and pie

**pointerLabelMode**

**rightPadding** Padding on right of drawing

**sameRadii** If true make x/y radii the same(default off)

**sideLabels** If true attempt to make piechart with labels along side and pointers

**sideLabelsOffset** The fraction of the pie width that the labels are situated at from the edges of the pie

**simpleLabels** If true(default) use a simple String not an advanced WedgeLabel. A WedgeLabel is customisable using the properties prefixed label\_ in the collection slices.

**slices** Collection of wedge descriptor objects

**startAngle** Angle of first slice; 0 is due East

**strokeColor** Color of the plot area border.

**strokeWidth** Width plot area border.

**topPadding** Padding at top of drawing

**wedgeRecord** callable(wedge,\*args,\*\*kws)

**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**xradius** X direction Radius

**y** Y position of the lower-left corner of the chart.

**yradius** Y direction Radius

### Example

```
def demo(self, drawing=None):
    if not drawing:
        tx,ty = self._getDrawingDimensions()
        drawing = Drawing(tx, ty)
    drawing.add(self.draw())
    return drawing
```

### Properties of Example Widget

```
background = None
bottomPadding = 5
checkLabelOverlap = 0
data = [38.4, 20.7, 18.9, 15.4, 6.6]
debug = 0
direction = 'clockwise'
drawLegend = 1
fillColor = None
height = 100
innerRadiusFraction = None
labels = None
leftPadding = 5
legend1.alignment = 'right'
legend1.autoXPadding = 5
legend1.autoYPadding = 2
legend1.boxAnchor = 'nw'
legend1.colEndCallout = None
legend1.colorNamePairs = [(Color(1,0,0,1), 'red'),
                           (Color(0,0,1,1), 'blue'),
                           (Color(0,.501961,0,1), 'green'),
                           (Color(1,.752941,.796078,1), 'pink'),
                           (Color(1,1,0,1), 'yellow'),
                           (PCMYKColor(11,11,72,0,spotName='PANTONE 458 CV',alpha=100), 'AAA:'),
                           (PCMYKColor(100,65,0,30,spotName='PANTONE 288 CV',alpha=100), 'AA:'),
                           (PCMYKColor(11,11,72,0,spotName='PANTONE 458 CV',density=75,alpha=100), 'A:'),
                           (PCMYKColor(100,65,0,30,spotName='PANTONE 288 CV',density=75,alpha=100), 'BBB:'),
                           (PCMYKColor(11,11,72,0,spotName='PANTONE 458 CV',density=50,alpha=100), 'NR:')]
legend1.columnMaximum = 7
legend1.deltax = 5.67
legend1.deltay = 14.17
legend1.dividerColor = Color(0,0,0,1)
legend1.dividerDashArray = None
legend1.dividerLines = 0
legend1.dividerOffsX = (0, 0)
legend1.dividerOffsY = 0
legend1.dividerWidth = 0.5
legend1.dx = 5.67
legend1.dxTextSpace = 11.39
legend1.dy = 5.67
legend1.fillColor = Color(0,0,0,1)
legend1.fontName = 'Helvetica-Bold'
legend1.fontSize = 6
legend1.strokeColor = Color(0,0,0,1)
legend1.strokeWidth = 0.5
legend1.subCols = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x28a6bd8>
legend1.swatchMarker = None
legend1.swdx = 0
legend1.swdy = 0
legend1.variColumn = 0
legend1.x = 117
legend1.y = 100
legend1.yGap = 0
legendNumberFormat = '%.1f%%'
```

```
legendNumberOffset = 51
legend_data = [38.4, 20.7, 18.9, 15.4, 6.6]
legend_names = ['AAA:', 'AA:', 'A:', 'BBB:', 'NR:']
orderMode = 'fixed'
pieAndLegend_colors = [PCMYKColor(11,11,72,0,spotName='PANTONE 458 CV',alpha=100),
                        PCMYKColor(100,65,0,30,spotName='PANTONE 288 CV',alpha=100),
                        PCMYKColor(11,11,72,0,spotName='PANTONE 458 CV',density=75,alpha=100),
                        PCMYKColor(100,65,0,30,spotName='PANTONE 288 CV',density=75,alpha=100),
                        PCMYKColor(11,11,72,0,spotName='PANTONE 458 CV',density=50,alpha=100),
                        PCMYKColor(100,65,0,30,spotName='PANTONE 288 CV',density=50,alpha=100)]

pointerLabelMode = None
rightPadding = 5
sameRadii = False
sideLabels = 0
sideLabelsOffset = 0.1
simpleLabels = 1
slices = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x28a6758>
startAngle = 90
strokeColor = None
strokeWidth = 1
topPadding = 5
width = 100
x = 0
xradius = None
y = 0
yradius = None
```

## **Pie(AbstractPieChart)**

### **Public Attributes**

**background** Handle to background object e.g. Rect(0,0,width,height).

**checkLabelOverlap** If true check and attempt to fix standard label overlaps(default off)

**data** List of numbers defining wedge sizes; need not sum to 1

**debug** Used only for debugging.

**direction** 'clockwise' or 'anticlockwise'

**fillColor** Color of the plot area interior.

**height** Height of the chart.

**innerRadiusFraction** fraction of radii to start wedges at

**labels** Optional list of labels to use for each data point

**orderMode** None

**other\_threshold** A value for doing thresholding, not used yet.

**pointerLabelMode**

**sameRadii** If true make x/y radii the same(default off)

**sideLabels** If true attempt to make piechart with labels along side and pointers

**sideLabelsOffset** The fraction of the pie width that the labels are situated at from the edges of the pie

**simpleLabels** If true(default) use a simple String not an advanced WedgeLabel. A WedgeLabel is customisable using the properties prefixed label\_ in the collection slices.

**slices** Collection of wedge descriptor objects

**startAngle** Angle of first slice; 0 is due East

**strokeColor** Color of the plot area border.

**strokeWidth** Width plot area border.

**wedgeRecord** callable(wedge,\*args,\*\*kws)

**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**xradius** X direction Radius

**y** Y position of the lower-left corner of the chart.

**yradius** Y direction Radius

### *Example*

```
def demo(self):
    d = Drawing(200, 100)

    pc = Pie()
    pc.x = 50
    pc.y = 10
    pc.width = 100
    pc.height = 80
    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.slices[3].fontColor = colors.red
pc.slices[0].fillColor = colors.darkcyan
pc.slices[1].fillColor = colors.blueviolet
pc.slices[2].fillColor = colors.blue
pc.slices[3].fillColor = colors.cyan
pc.slices[4].fillColor = colors.aquamarine
pc.slices[5].fillColor = colors.cadetblue
pc.slices[6].fillColor = colors.lightcoral

d.add(pc)
return d
```

### *Properties of Example Widget*

```
background = None
checkLabelOverlap = 0
data = [1, 2.3, 1.7, 4.2]
debug = 0
direction = 'clockwise'
fillColor = None
height = 100
innerRadiusFraction = None
labels = None
orderMode = 'fixed'
pointerLabelMode = None
sameRadii = False
sideLabels = 0
sideLabelsOffset = 0.1
simpleLabels = 1
slices = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x28b7ea8>
startAngle = 90
strokeColor = None
strokeWidth = 1
width = 100
x = 0
xradius = None
y = 0
yradius = None
```

## Pie3d(Pie)

### Public Attributes

**angle\_3d** The view angle.

**background** Handle to background object e.g. Rect(0,0,width,height).

**checkLabelOverlap** If true check and attempt to fix standard label overlaps(default off)

**data** List of numbers defining wedge sizes; need not sum to 1

**debug** Used only for debugging.

**depth\_3d** depth of the pie.

**direction** 'clockwise' or 'anticlockwise'

**fillColor** Color of the plot area interior.

**height** Height of the chart.

**innerRadiusFraction** fraction of radii to start wedges at

**labels** Optional list of labels to use for each data point

**orderMode** None

**other\_threshold** A value for doing thresholding, not used yet.

**perspective** A flattening parameter.

**pointerLabelMode**

**sameRadii** If true make x/y radii the same(default off)

**sideLabels** If true attempt to make piechart with labels along side and pointers

**sideLabelsOffset** The fraction of the pie width that the labels are situated at from the edges of the pie

**simpleLabels** If true(default) use a simple String not an advanced WedgeLabel. A WedgeLabel is customisable using the properties prefixed label\_ in the collection slices.

**slices** Collection of wedge descriptor objects

**startAngle** Angle of first slice; 0 is due East

**strokeColor** Color of the plot area border.

**strokeWidth** Width plot area border.

**wedgeRecord** callable(wedge,\*args,\*\*kwds)

**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**xradius** X direction Radius

**y** Y position of the lower-left corner of the chart.

**yradius** Y direction Radius

### Example

```
def demo(self):
    d = Drawing(200, 100)

    pc = Pie()
    pc.x = 50
    pc.y = 10
    pc.width = 100
```



```
pc.height = 80
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.slices[3].fontColor = colors.red
pc.slices[0].fillColor = colors.darkcyan
pc.slices[1].fillColor = colors.blueviolet
pc.slices[2].fillColor = colors.blue
pc.slices[3].fillColor = colors.cyan
pc.slices[4].fillColor = colors.aquamarine
pc.slices[5].fillColor = colors.cadetblue
pc.slices[6].fillColor = colors.lightcoral
self.slices[1].visible = 0
self.slices[3].visible = 1
self.slices[4].visible = 1
self.slices[5].visible = 1
self.slices[6].visible = 0

d.add(pc)
return d
```

### *Properties of Example Widget*

```
background = None
checkLabelOverlap = 0
data = [12.5, 20.1, 2.0, 22.0, 5.0, 18.0, 13.0]
debug = 0
direction = 'clockwise'
fillColor = None
height = 200
innerRadiusFraction = None
labels = None
orderMode = 'fixed'
pointerLabelMode = None
sameRadii = False
sideLabels = 0
sideLabelsOffset = 0.1
simpleLabels = 1
slices = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x296c7e8>
startAngle = 90
strokeColor = None
strokeWidth = 1
width = 300
x = 0
xradius = None
y = 0
yradius = None
```

## WedgeLabel (Label)

### Public Attributes

**angle** angle of label: default (0), 90 is vertical, 180 is upside down, etc

**bottomPadding** padding at bottom of box

**boxAnchor** anchoring point of the label

**boxFillColor** the filling color of the box

**boxStrokeColor** border color of the box

**boxStrokeWidth** border width

**boxTarget** one of ('normal','anti','lo','hi')

**customDrawChanger** An instance of CustomDrawChanger to modify the behavior at draw time

**dx** delta x - offset

**dy** delta y - offset

**fillColor** label text color

**fontName** the name of the font used

**fontSize** the size of the font

**height** the height of the text

**leading**

**leftPadding** padding at left of box

**maxWidth** maximum width the label can grow to

**rightPadding** padding at right of box

**strokeColor** label text border color

**strokeWidth** label text border width

**text** the actual text to display

**textAnchor** the anchoring point of the text inside the label

**topPadding** padding at top of box

**useAscentDescent** If True then the font's Ascent Descent will be used to compute default heights and baseline.

**visible** True if the label is to be drawn

**width** the width of the label

**x**

**y**

### Example

```
def demo(self):
    """This shows a label positioned with its top right corner
    at the top centre of the drawing, and rotated 45 degrees."""

    d = Drawing(200, 100)

    # mark the origin of the label
    d.add(Circle(100,90, 5, fillColor=colors.green))
    lab = Label()
```

```
lab.setOrigin(100,90)
lab.boxAnchor = 'ne'
lab.angle = 45
lab.dx = 0
lab.dy = -20
lab.boxStrokeColor = colors.green
lab.setText('Another\nMulti-Line\nString')
d.add(lab)

return d
```

### *Properties of Example Widget*

```
angle = 0
bottomPadding = 0
boxAnchor = 'c'
boxFillColor = None
boxStrokeColor = None
boxStrokeWidth = 0.5
boxTarget = 'normal'
dx = 0
dy = 0
fillColor = Color(0,0,0,1)
fontName = 'Times-Roman'
fontSize = 10
height = None
leading = None
leftPadding = 0
maxWidth = None
rightPadding = 0
strokeColor = None
strokeWidth = 0.1
textAnchor = 'start'
topPadding = 0
useAscentDescent = False
visible = 1
width = None
x = 0
y = 0
```

## Functions

### `sample0a( ... )`

Make a degenerated pie chart with only one slice.

#### *Example*

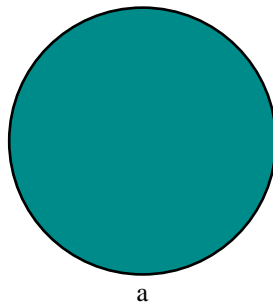
```
def sample0a():
    "Make a degenerated pie chart with only one slice."

    d = Drawing(400, 200)

    pc = Pie()
    pc.x = 150
    pc.y = 50
    pc.data = [10]
    pc.labels = ['a']
    pc.slices.strokeWidth=1#0.5

    d.add(pc)

    return d
```



**sample0b( ... )**

Make a degenerated pie chart with only one slice.

*Example*

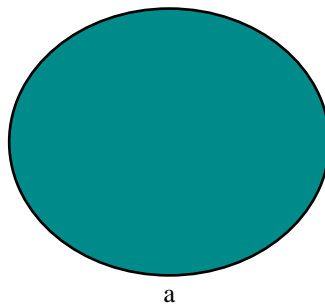
```
def sample0b():
    "Make a degenerated pie chart with only one slice."

    d = Drawing(400, 200)

    pc = Pie()
    pc.x = 150
    pc.y = 50
    pc.width = 120
    pc.height = 100
    pc.data = [10]
    pc.labels = ['a']
    pc.slices.strokeWidth=1#0.5

    d.add(pc)

    return d
```



**sample1( ... )**

Make a typical pie chart with with one slice treated in a special way.

*Example*

```
def sample1():
    "Make a typical pie chart with with one slice treated in a special way."

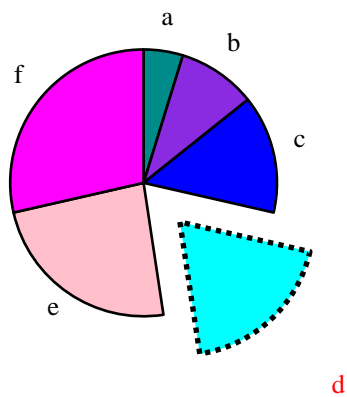
    d = Drawing(400, 200)

    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=1#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 = colors.red

    d.add(pc)

    return d
```



**sample2( ... )**

Make a pie chart with nine slices.

*Example*

```
def sample2():
    "Make a pie chart with nine slices."

    d = Drawing(400, 200)

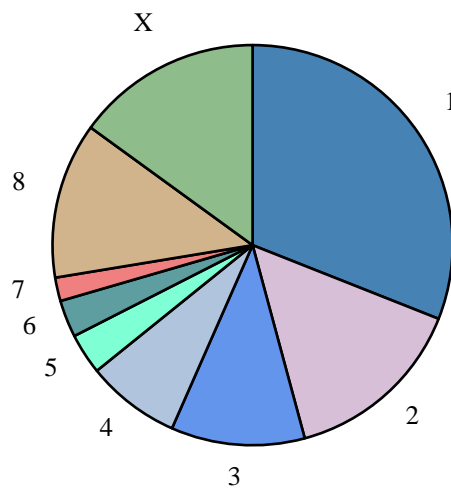
    pc = Pie()
    pc.x = 125
    pc.y = 25
    pc.data = [0.31, 0.148, 0.108,
               0.076, 0.033, 0.03,
               0.019, 0.126, 0.15]
    pc.labels = ['1', '2', '3', '4', '5', '6', '7', '8', 'X']

    pc.width = 150
    pc.height = 150
    pc.slices.strokeWidth=1#0.5

    pc.slices[0].fillColor = colors.steelblue
    pc.slices[1].fillColor = colors.thistle
    pc.slices[2].fillColor = colors.cornflower
    pc.slices[3].fillColor = colors.lightsteelblue
    pc.slices[4].fillColor = colors.aquamarine
    pc.slices[5].fillColor = colors.cadetblue
    pc.slices[6].fillColor = colors.lightcoral
    pc.slices[7].fillColor = colors.tan
    pc.slices[8].fillColor = colors.darkseagreen

    d.add(pc)

    return d
```



**sample3( ... )**

Make a pie chart with a very slim slice.

*Example*

```
def sample3():
    "Make a pie chart with a very slim slice."

    d = Drawing(400, 200)

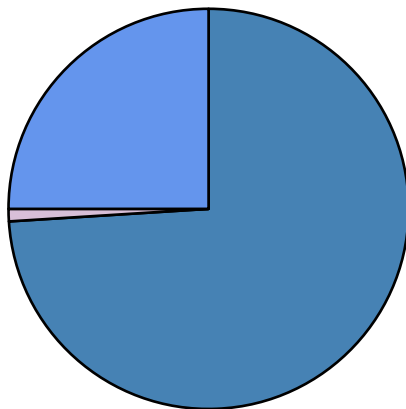
    pc = Pie()
    pc.x = 125
    pc.y = 25

    pc.data = [74, 1, 25]

    pc.width = 150
    pc.height = 150
    pc.slices.strokeWidth=1#0.5
    pc.slices[0].fillColor = colors.steelblue
    pc.slices[1].fillColor = colors.thistle
    pc.slices[2].fillColor = colors.cornflower

    d.add(pc)

    return d
```





**sample4( ... )**

Make a pie chart with several very slim slices.

*Example*

```
def sample4():
    "Make a pie chart with several very slim slices."

    d = Drawing(400, 200)

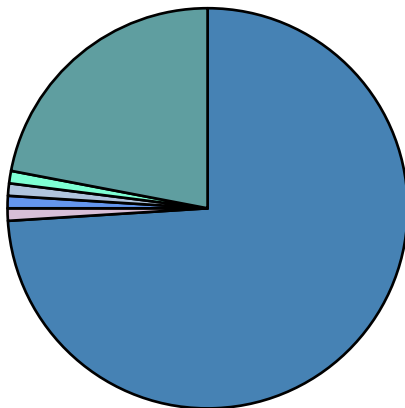
    pc = Pie()
    pc.x = 125
    pc.y = 25

    pc.data = [74, 1, 1, 1, 1, 22]

    pc.width = 150
    pc.height = 150
    pc.slices.strokeWidth=1#0.5
    pc.slices[0].fillColor = colors.steelblue
    pc.slices[1].fillColor = colors.thistle
    pc.slices[2].fillColor = colors.cornflower
    pc.slices[3].fillColor = colors.lightsteelblue
    pc.slices[4].fillColor = colors.aquamarine
    pc.slices[5].fillColor = colors.cadetblue

    d.add(pc)

    return d
```



**sample5( ... )**

Make a pie with side labels.

*Example*

```
def sample5():
    "Make a pie with side labels."

    d = Drawing(400, 200)

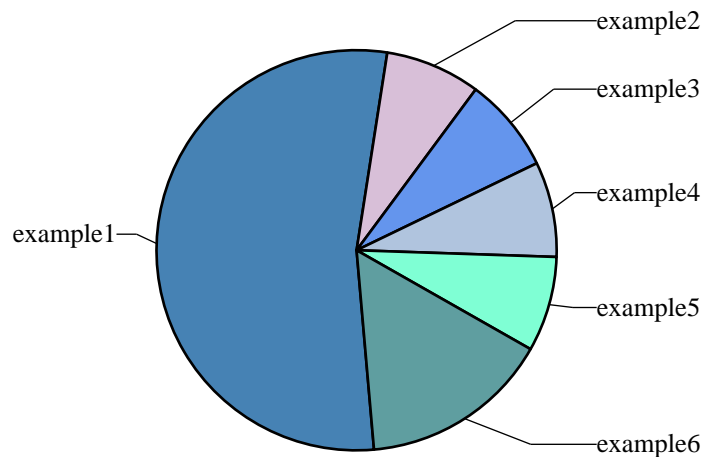
    pc = Pie()
    pc.x = 125
    pc.y = 25

    pc.data = [7, 1, 1, 1, 1, 2]
    pc.labels = ['example1', 'example2', 'example3', 'example4', 'example5', 'example6']
    pc.sideLabels = 1

    pc.width = 150
    pc.height = 150
    pc.slices.strokeWidth=1#0.5
    pc.slices[0].fillColor = colors.steelblue
    pc.slices[1].fillColor = colors.thistle
    pc.slices[2].fillColor = colors.cornflower
    pc.slices[3].fillColor = colors.lightsteelblue
    pc.slices[4].fillColor = colors.aquamarine
    pc.slices[5].fillColor = colors.cadetblue

    d.add(pc)

    return d
```

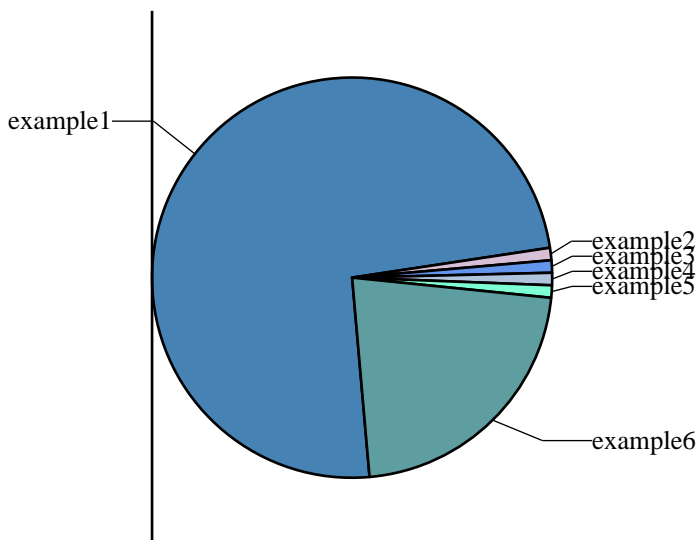


**sample6( ... )**

Illustrates the pie moving to leave space for the left labels

*Example*

```
def sample6():  
    "Illustrates the pie moving to leave space for the left labels"  
  
    d = Drawing(400, 200)  
  
    pc = Pie()  
    "The x value of the pie chart is 0"  
    pc.x = 0  
    pc.y = 25  
  
    pc.data = [74, 1, 1, 1, 1, 22]  
    pc.labels = ['example1', 'example2', 'example3', 'example4', 'example5', 'example6']  
    pc.sideLabels = 1  
  
    pc.width = 150  
    pc.height = 150  
    pc.slices.strokeWidth=1#0.5  
    pc.slices[0].fillColor = colors.steelblue  
    pc.slices[1].fillColor = colors.thistle  
    pc.slices[2].fillColor = colors.cornflower  
    pc.slices[3].fillColor = colors.lightsteelblue  
    pc.slices[4].fillColor = colors.aquamarine  
    pc.slices[5].fillColor = colors.cadetblue  
  
    l = Line(0,0,0,200)  
  
    d.add(pc)  
    d.add(l)  
  
    return d
```



```
sample7( ... )
```

## Case with overlapping pointers

### Example

```
def sample7():

    "Case with overlapping pointers"

    d = Drawing(400, 200)

    pc = Pie()
    pc.y = 50
    pc.x = 150
    pc.width = 100
    pc.height = 100

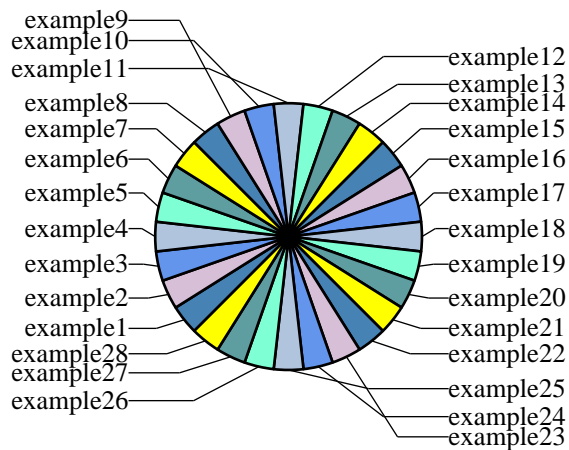
    pc.data = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
    pc.labels = ['example1', 'example2', 'example3', 'example4', 'example5', 'example6', 'example7',
                'example8', 'example9', 'example10', 'example11', 'example12', 'example13', 'example14',
                'example15', 'example16', 'example17', 'example18', 'example19', 'example20', 'example21',
                'example22', 'example23', 'example24', 'example25', 'example26', 'example27', 'example28']

    pc.sideLabels = 1
    pc.checkLabelOverlap = 1
    pc.simpleLabels = 0

    pc.slices.strokeWidth=1#0.5
    pc.slices[0].fillColor = colors.steelblue
    pc.slices[1].fillColor = colors.thistle
    pc.slices[2].fillColor = colors.cornflower
    pc.slices[3].fillColor = colors.lightsteelblue
    pc.slices[4].fillColor = colors.aquamarine
    pc.slices[5].fillColor = colors.cadetblue

    d.add(pc)

    return d
```

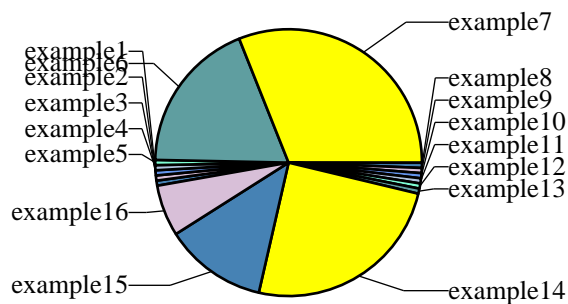


**sample8( ... )**

Case with overlapping labels

*Example*

```
def sample8():  
    "Case with overlapping labels"  
    "Labels overlap if they do not belong to adjacent pie slices due to nature of checkLabelOverlap"  
  
    d = Drawing(400, 200)  
  
    pc = Pie()  
    pc.y = 50  
    pc.x = 150  
    pc.width = 100  
    pc.height = 100  
  
    pc.data = [1, 1, 1, 1, 1, 30, 50, 1, 1, 1, 1, 1, 1, 40, 20, 10]  
    pc.labels = ['example1', 'example2', 'example3', 'example4', 'example5', 'example6', 'example7',  
                'example8', 'example9', 'example10', 'example11', 'example12', 'example13', 'example14',  
                'example15', 'example16']  
    pc.sideLabels = 1  
    pc.checkLabelOverlap = 1  
  
    pc.slices.strokeWidth=1#0.5  
    pc.slices[0].fillColor = colors.steelblue  
    pc.slices[1].fillColor = colors.thistle  
    pc.slices[2].fillColor = colors.cornflower  
    pc.slices[3].fillColor = colors.lightsteelblue  
    pc.slices[4].fillColor = colors.aquamarine  
    pc.slices[5].fillColor = colors.cadetblue  
  
    d.add(pc)  
  
    return d
```

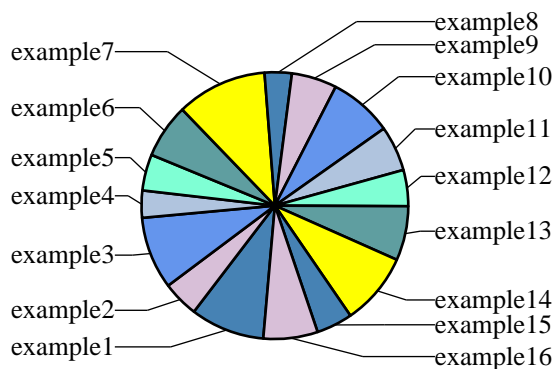


**sample9( ... )**

Case with overlapping labels

*Example*

```
def sample9():  
    "Case with overlapping labels"  
    "Labels overlap if they do not belong to adjacent pies due to nature of checkLabelOverlap"  
  
    d = Drawing(400, 200)  
  
    pc = Pie()  
    pc.x = 125  
    pc.y = 50  
  
    pc.data = [41, 20, 40, 15, 20, 30, 50, 15, 25, 35, 25, 20, 30, 40, 20, 30]  
    pc.labels = ['example1', 'example2', 'example3', 'example4', 'example5', 'example6', 'example7',  
                'example8', 'example9', 'example10', 'example11', 'example12', 'example13', 'example14',  
                'example15', 'example16']  
    pc.sideLabels = 1  
    pc.checkLabelOverlap = 1  
  
    pc.width = 100  
    pc.height = 100  
    pc.slices.strokeWidth=1#0.5  
    pc.slices[0].fillColor = colors.steelblue  
    pc.slices[1].fillColor = colors.thistle  
    pc.slices[2].fillColor = colors.cornflower  
    pc.slices[3].fillColor = colors.lightsteelblue  
    pc.slices[4].fillColor = colors.aquamarine  
    pc.slices[5].fillColor = colors.cadetblue  
  
    d.add(pc)  
  
    return d
```



## **barcharts**

This module defines a variety of Bar Chart components.

The basic flavors are stacked and side-by-side, available in horizontal and vertical versions.

## **Classes**

### **BarChart (PlotArea)**

Abstract base class, unusable by itself.

## **Public Attributes**

**annotations** list of callables, will be called with self, xscale, yscale.

**background** Handle to background object e.g. Rect(0,0,width,height).

**barLabelArray** explicit array of bar label values, must match size of data if present.

**barLabelCallOut** Callout function(label) label.\_callOutInfo = (self,g,rowNo,colNo,x,y,width,height,x00,y00,x0,y0)

**barLabelFormat** Formatting string or function used for bar labels.

**barLabels** Handle to the list of bar labels.

**barRecord** callable(bar,label=labelText,value=value,\*\*kwds) to record bar information

**barSpacing** Width between individual bars.

**barWidth** The width of an individual bar.

**bars** Handle of the individual bars.

**categoryAxis** Handle of the category axis.

**categoryLabelBarOrder** where any label bar should appear first/last

**categoryLabelBarSize** width to leave for a category label to go between categories.

**categoryNALabel** Label to use for a group of N/A values.

**data** Data to be plotted, list of (lists of) numbers.

**debug** Used only for debugging.

**fillColor** Color of the plot area interior.

**groupSpacing** Width between groups of bars.

**height** Height of the chart.

**naLabel** Label to use for N/A values.

**reversePlotOrder** If true, reverse common category plot order.

**strokeColor** Color of the plot area border.

**strokeWidth** Width plot area border.

**useAbsolute** Flag to use absolute spacing values; use string of gsb for finer control (g=groupSpacing,s=barSpacing,b=barWidth).

**valueAxis** Handle of the value axis.

**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**y** Y position of the lower-left corner of the chart.

**zIndexOverrides** None (the default ie use old z ordering scheme) or a ',' separated list of key=value (int/float) for new zIndex ordering. If used defaults are background=0, categoryAxis=1, valueAxis=2, bars=3, barLabels=4, categoryAxisGrid=5, valueAxisGrid=6, annotations=7

## BarChart3D(BarChart)

### Public Attributes

**annotations** list of callables, will be called with self, xscale, yscale.

**background** Handle to background object e.g. Rect(0,0,width,height).

**barLabelArray** explicit array of bar label values, must match size of data if present.

**barLabelCallOut** Callout function(label) label.\_callOutInfo = (self,g,rowNo,colNo,x,y,width,height,x00,y00,x0,y0)

**barLabelFormat** Formatting string or function used for bar labels.

**barLabels** Handle to the list of bar labels.

**barRecord** callable(bar,label=labelText,value=value,\*\*kwds) to record bar information

**barSpacing** Width between individual bars.

**barWidth** The width of an individual bar.

**bars** Handle of the individual bars.

**categoryAxis** Handle of the category axis.

**categoryLabelBarOrder** where any label bar should appear first/last

**categoryLabelBarSize** width to leave for a category label to go between categories.

**categoryNALabel** Label to use for a group of N/A values.

**data** Data to be plotted, list of (lists of) numbers.

**debug** Used only for debugging.

**fillColor** Color of the plot area interior.

**groupSpacing** Width between groups of bars.

**height** Height of the chart.

**naLabel** Label to use for N/A values.

**reversePlotOrder** If true, reverse common category plot order.

**strokeColor** Color of the plot area border.

**strokeWidth** Width plot area border.

**theta\_x** dx/dz

**theta\_y** dy/dz

**useAbsolute** Flag to use absolute spacing values; use string of gsb for finer control (g=groupSpacing,s=barSpacing,b=barWidth).

**valueAxis** Handle of the value axis.

**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**y** Y position of the lower-left corner of the chart.



**zDepth** depth of an individual series

**zIndexOverrides** None (the default ie use old z ordering scheme) or a ',' separated list of key=value (int/float) for new zIndex ordering. If used defaults are background=0, categoryAxis=1, valueAxis=2, bars=3, barLabels=4, categoryAxisGrid=5, valueAxisGrid=6, annotations=7

**zSpace** z gap around series

## HorizontalBarChart (BarChart)

Horizontal bar chart with multiple side-by-side bars.

### Public Attributes

**annotations** list of callables, will be called with self, xscale, yscale.

**background** Handle to background object e.g. Rect(0,0,width,height).

**barLabelArray** explicit array of bar label values, must match size of data if present.

**barLabelCallOut** Callout function(label) label.\_callOutInfo = (self,g,rowNo,colNo,x,y,width,height,x00,y00,x0,y0)

**barLabelFormat** Formatting string or function used for bar labels.

**barLabels** Handle to the list of bar labels.

**barRecord** callable(bar,label=labelText,value=value,\*\*kwds) to record bar information

**barSpacing** Width between individual bars.

**barWidth** The width of an individual bar.

**bars** Handle of the individual bars.

**categoryAxis** Handle of the category axis.

**categoryLabelBarOrder** where any label bar should appear first/last

**categoryLabelBarSize** width to leave for a category label to go between categories.

**categoryNALabel** Label to use for a group of N/A values.

**data** Data to be plotted, list of (lists of) numbers.

**debug** Used only for debugging.

**fillColor** Color of the plot area interior.

**groupSpacing** Width between groups of bars.

**height** Height of the chart.

**naLabel** Label to use for N/A values.

**reversePlotOrder** If true, reverse common category plot order.

**strokeColor** Color of the plot area border.

**strokeWidth** Width plot area border.

**useAbsolute** Flag to use absolute spacing values; use string of gsb for finer control (g=groupSpacing,s=barSpacing,b=barWidth).

**valueAxis** Handle of the value axis.

**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**y** Y position of the lower-left corner of the chart.

**zIndexOverrides** None (the default ie use old z ordering scheme) or a ',' separated list of key=value (int/float) for new zIndex ordering. If used defaults are background=0, categoryAxis=1, valueAxis=2, bars=3, barLabels=4, categoryAxisGrid=5, valueAxisGrid=6, annotations=7

#### Example

```
def demo(self):
    """Shows basic use of a bar chart"""
    if self.__class__.__name__=='BarChart':
        raise NotImplementedError, 'Abstract Class BarChart has no demo'
    drawing = Drawing(200, 100)
    bc = self.__class__()
    drawing.add(bc)
    return drawing
```

#### Properties of Example Widget

```
background = None
barLabelArray = None
barLabelFormat = None
barLabels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x2af4c68>
barSpacing = 0
barWidth = 10
bars = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x2af4cf8>
categoryAxis.categoryNames = None
categoryAxis.drawGridLast = False
categoryAxis.gridEnd = None
categoryAxis.gridStart = None
categoryAxis.gridStrokeColor = Color(0,0,0,1)
categoryAxis.gridStrokeDashArray = None
categoryAxis.gridStrokeLineCap = 0
categoryAxis.gridStrokeLineJoin = 0
categoryAxis.gridStrokeMiterLimit = 10
categoryAxis.gridStrokeWidth = 0.25
categoryAxis.hiLLen = 0
categoryAxis.hiPad = 0
categoryAxis.joinAxis = None
categoryAxis.joinAxisMode = None
categoryAxis.joinAxisPos = None
categoryAxis.labelAxisMode = 'axis'
categoryAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x2af4b00>
categoryAxis.loLLen = 0
categoryAxis.loPad = 0
categoryAxis.reverseDirection = 0
categoryAxis.strokeColor = Color(0,0,0,1)
categoryAxis.strokeDashArray = None
categoryAxis.strokeLineCap = 0
categoryAxis.strokeLineJoin = 0
categoryAxis.strokeMiterLimit = 10
categoryAxis.strokeWidth = 1
categoryAxis.style = 'parallel'
categoryAxis.tickLeft = 5
categoryAxis.tickRight = 0
categoryAxis.tickShift = 0
categoryAxis.visible = 1
categoryAxis.visibleAxis = 1
categoryAxis.visibleGrid = 0
categoryAxis.visibleLabels = 1
categoryAxis.visibleTicks = 1
categoryNALabel = None
data = [(100, 110, 120, 130), (70, 80, 85, 90)]
debug = 0
fillColor = None
groupSpacing = 5
height = 85
naLabel = None
reversePlotOrder = 0
strokeColor = None
strokeWidth = 1
useAbsolute = 0
```

```
valueAxis.abf_ignore_zero = False
valueAxis.avoidBoundFrac = None
valueAxis.avoidBoundSpace = None
valueAxis.drawGridLast = False
valueAxis.forceZero = 0
valueAxis.gridEnd = None
valueAxis.gridStart = None
valueAxis.gridStrokeColor = Color(0,0,0,1)
valueAxis.gridStrokeDashArray = None
valueAxis.gridStrokeLineCap = 0
valueAxis.gridStrokeLineJoin = 0
valueAxis.gridStrokeMiterLimit = 10
valueAxis.gridStrokeWidth = 0.25
valueAxis.hiLLen = 0
valueAxis.joinAxis = None
valueAxis.joinAxisMode = None
valueAxis.joinAxisPos = None
valueAxis.keepTickLabelsInside = 0
valueAxis.labelAxisMode = 'axis'
valueAxis.labelTextFormat = None
valueAxis.labelTextPostFormat = None
valueAxis.labelTextScale = None
valueAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x2af4bd8>
valueAxis.loLLen = 0
valueAxis.maximumTicks = 7
valueAxis.minimumTickSpacing = 10
valueAxis.origShiftIPC = None
valueAxis.origShiftMin = None
valueAxis.origShiftSpecialValue = None
valueAxis.rangeRound = 'none'
valueAxis.requiredRange = 0
valueAxis.reverseDirection = 0
valueAxis.skipEndL = 'none'
valueAxis.strokeColor = Color(0,0,0,1)
valueAxis.strokeDashArray = None
valueAxis.strokeLineCap = 0
valueAxis.strokeLineJoin = 0
valueAxis.strokeMiterLimit = 10
valueAxis.strokeWidth = 1
valueAxis.style = 'normal'
valueAxis.subGridEnd = None
valueAxis.subGridStart = None
valueAxis.subGridStrokeColor = Color(0,0,0,1)
valueAxis.subGridStrokeDashArray = None
valueAxis.subGridStrokeLineCap = 0
valueAxis.subGridStrokeLineJoin = 0
valueAxis.subGridStrokeMiterLimit = 10
valueAxis.subGridStrokeWidth = 0.25
valueAxis.subTickHi = 0
valueAxis.subTickLo = 0
valueAxis.subTickNum = 0
valueAxis.tickAxisMode = 'axis'
valueAxis.tickDown = 5
valueAxis.tickUp = 0
valueAxis.valueMax = None
valueAxis.valueMin = None
valueAxis.valueStep = None
valueAxis.visible = 1
valueAxis.visibleAxis = 1
valueAxis.visibleGrid = 0
valueAxis.visibleLabels = 1
valueAxis.visibleSubGrid = 0
valueAxis.visibleSubTicks = 0
valueAxis.visibleTicks = 1
valueAxis.zrangePref = 0
width = 180
x = 20
y = 10
zIndexOverrides = None
```

## **HorizontalBarChart3D(BarChart3D, HorizontalBarChart)**

### **Public Attributes**

**annotations** list of callables, will be called with self, xscale, yscale.

**background** Handle to background object e.g. Rect(0,0,width,height).

**barLabelArray** explicit array of bar label values, must match size of data if present.

**barLabelCallOut** Callout function(label) label.\_callOutInfo =  
(self,g,rowNo,colNo,x,y,width,height,x00,y00,x0,y0)

**barLabelFormat** Formatting string or function used for bar labels.

**barLabels** Handle to the list of bar labels.

**barRecord** callable(bar,label=labelText,value=value,\*\*kwds) to record bar information

**barSpacing** Width between individual bars.

**barWidth** The width of an individual bar.

**bars** Handle of the individual bars.

**categoryAxis** Handle of the category axis.

**categoryLabelBarOrder** where any label bar should appear first/last

**categoryLabelBarSize** width to leave for a category label to go between categories.

**categoryNALabel** Label to use for a group of N/A values.

**data** Data to be plotted, list of (lists of) numbers.

**debug** Used only for debugging.

**fillColor** Color of the plot area interior.

**groupSpacing** Width between groups of bars.

**height** Height of the chart.

**naLabel** Label to use for N/A values.

**reversePlotOrder** If true, reverse common category plot order.

**strokeColor** Color of the plot area border.

**strokeWidth** Width plot area border.

**theta\_x** dx/dz

**theta\_y** dy/dz

**useAbsolute** Flag to use absolute spacing values; use string of gsb for finer control  
(g=groupSpacing,s=barSpacing,b=barWidth).

**valueAxis** Handle of the value axis.

**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**y** Y position of the lower-left corner of the chart.

**zDepth** depth of an individual series

**zIndexOverrides** None (the default ie use old z ordering scheme) or a ',' separated list of  
key=value (int/float) for new zIndex ordering. If used defaults are background=0,  
categoryAxis=1, valueAxis=2, bars=3, barLabels=4, categoryAxisGrid=5, valueAxisGrid=6,

annotations=7

**zSpace** z gap around series

### Example

```
def demo(self):
    """Shows basic use of a bar chart"""
    if self.__class__.__name__=='BarChart':
        raise NotImplementedError, 'Abstract Class BarChart has no demo'
    drawing = Drawing(200, 100)
    bc = self.__class__()
    drawing.add(bc)
    return drawing
```

### Properties of Example Widget

```
background = None
barLabelArray = None
barLabelFormat = None
barLabels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x2b06f80>
barSpacing = 0
barWidth = 10
bars = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x2b08050>
categoryAxis.categoryNames = None
categoryAxis.drawGridLast = False
categoryAxis.gridEnd = None
categoryAxis.gridStart = None
categoryAxis.gridStrokeColor = Color(0,0,0,1)
categoryAxis.gridStrokeDashArray = None
categoryAxis.gridStrokeLineCap = 0
categoryAxis.gridStrokeLineJoin = 0
categoryAxis.gridStrokeMiterLimit = 10
categoryAxis.gridStrokeWidth = 0.25
categoryAxis.hiLLen = 0
categoryAxis.hiPad = 0
categoryAxis.joinAxis = None
categoryAxis.joinAxisMode = None
categoryAxis.joinAxisPos = None
categoryAxis.labelAxisMode = 'axis'
categoryAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x2b06e18>
categoryAxis.loLLen = 0
categoryAxis.loPad = 0
categoryAxis.reverseDirection = 0
categoryAxis.strokeColor = Color(0,0,0,1)
categoryAxis.strokeDashArray = None
categoryAxis.strokeLineCap = 0
categoryAxis.strokeLineJoin = 0
categoryAxis.strokeMiterLimit = 10
categoryAxis.strokeWidth = 1
categoryAxis.style = 'parallel'
categoryAxis.tickLeft = 5
categoryAxis.tickRight = 0
categoryAxis.tickShift = 0
categoryAxis.visible = 1
categoryAxis.visibleAxis = 1
categoryAxis.visibleGrid = 0
categoryAxis.visibleLabels = 1
categoryAxis.visibleTicks = 1
categoryNALabel = None
data = [(100, 110, 120, 130), (70, 80, 85, 90)]
debug = 0
fillColor = None
groupSpacing = 5
height = 85
naLabel = None
reversePlotOrder = 0
strokeColor = None
strokeWidth = 1
useAbsolute = 0
valueAxis.abf_ignore_zero = False
valueAxis.avoidBoundFrac = None
valueAxis.avoidBoundSpace = None
valueAxis.drawGridLast = False
```

```
valueAxis.forceZero = 0
valueAxis.gridEnd = None
valueAxis.gridStart = None
valueAxis.gridStrokeColor = Color(0,0,0,1)
valueAxis.gridStrokeDashArray = None
valueAxis.gridStrokeLineCap = 0
valueAxis.gridStrokeLineJoin = 0
valueAxis.gridStrokeMiterLimit = 10
valueAxis.gridStrokeWidth = 0.25
valueAxis.hiLLen = 0
valueAxis.joinAxis = None
valueAxis.joinAxisMode = None
valueAxis.joinAxisPos = None
valueAxis.keepTickLabelsInside = 0
valueAxis.labelAxisMode = 'axis'
valueAxis.labelTextFormat = None
valueAxis.labelTextPostFormat = None
valueAxis.labelTextScale = None
valueAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x2b06ef0>
valueAxis.loLLen = 0
valueAxis.maximumTicks = 7
valueAxis.minimumTickSpacing = 10
valueAxis.origShiftIPC = None
valueAxis.origShiftMin = None
valueAxis.origShiftSpecialValue = None
valueAxis.rangeRound = 'none'
valueAxis.requiredRange = 0
valueAxis.reverseDirection = 0
valueAxis.skipEndL = 'none'
valueAxis.strokeColor = Color(0,0,0,1)
valueAxis.strokeDashArray = None
valueAxis.strokeLineCap = 0
valueAxis.strokeLineJoin = 0
valueAxis.strokeMiterLimit = 10
valueAxis.strokeWidth = 1
valueAxis.style = 'normal'
valueAxis.subGridEnd = None
valueAxis.subGridStart = None
valueAxis.subGridStrokeColor = Color(0,0,0,1)
valueAxis.subGridStrokeDashArray = None
valueAxis.subGridStrokeLineCap = 0
valueAxis.subGridStrokeLineJoin = 0
valueAxis.subGridStrokeMiterLimit = 10
valueAxis.subGridStrokeWidth = 0.25
valueAxis.subTickHi = 0
valueAxis.subTickLo = 0
valueAxis.subTickNum = 0
valueAxis.tickAxisMode = 'axis'
valueAxis.tickDown = 5
valueAxis.tickUp = 0
valueAxis.valueMax = None
valueAxis.valueMin = None
valueAxis.valueStep = None
valueAxis.visible = 1
valueAxis.visibleAxis = 1
valueAxis.visibleGrid = 0
valueAxis.visibleLabels = 1
valueAxis.visibleSubGrid = 0
valueAxis.visibleSubTicks = 0
valueAxis.visibleTicks = 1
valueAxis.zrangePref = 0
width = 180
x = 20
y = 10
zIndexOverrides = None
```

## SampleH5c4 (Drawing)

Simple bar chart with absolute spacing.

### Example

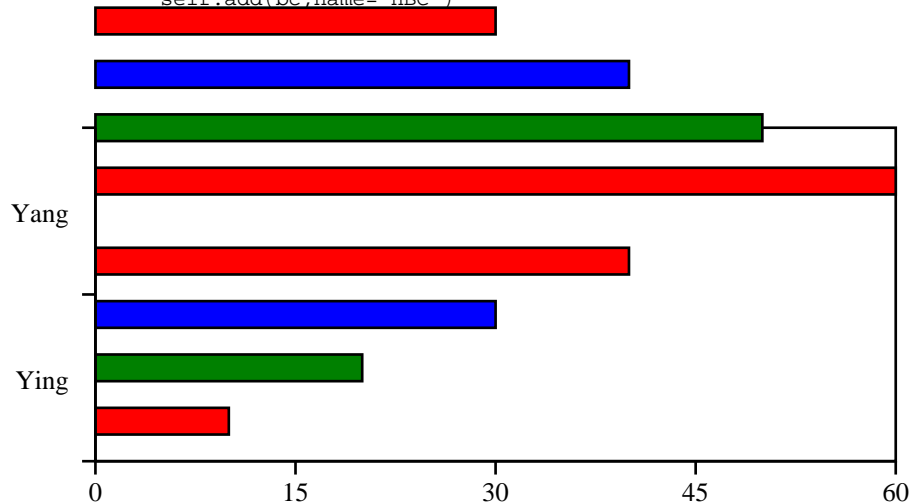
```
def __init__(self,width=400,height=200,*args,**kw):
    Drawing.__init__(self,width,height,*args,**kw)
    bc = HorizontalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = dataSample5
    bc.strokeColor = colors.black

    bc.useAbsolute = 1
    bc.barWidth = 10
    bc.groupSpacing = 20
    bc.barSpacing = 10

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'e'
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    self.add(bc,name='HBC')
```



## VerticalBarChart (BarChart)

Vertical bar chart with multiple side-by-side bars.

### Public Attributes

**annotations** list of callables, will be called with self, xscale, yscale.

**background** Handle to background object e.g. Rect(0,0,width,height).

**barLabelArray** explicit array of bar label values, must match size of data if present.

**barLabelCallOut** Callout function(label) label.\_callOutInfo = (self,g,rowNo,colNo,x,y,width,height,x0,y0,x0,y0)

**barLabelFormat** Formatting string or function used for bar labels.

**barLabels** Handle to the list of bar labels.

**barRecord** callable(bar,label=labelText,value=value,\*\*kwds) to record bar information

**barSpacing** Width between individual bars.

**barWidth** The width of an individual bar.

**bars** Handle of the individual bars.

**categoryAxis** Handle of the category axis.

**categoryLabelBarOrder** where any label bar should appear first/last

**categoryLabelBarSize** width to leave for a category label to go between categories.

**categoryNALabel** Label to use for a group of N/A values.

**data** Data to be plotted, list of (lists of) numbers.

**debug** Used only for debugging.

**fillColor** Color of the plot area interior.

**groupSpacing** Width between groups of bars.

**height** Height of the chart.

**naLabel** Label to use for N/A values.

**reversePlotOrder** If true, reverse common category plot order.

**strokeColor** Color of the plot area border.

**strokeWidth** Width plot area border.

**useAbsolute** Flag to use absolute spacing values; use string of gsb for finer control (g=groupSpacing,s=barSpacing,b=barWidth).

**valueAxis** Handle of the value axis.

**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**y** Y position of the lower-left corner of the chart.

**zIndexOverrides** None (the default ie use old z ordering scheme) or a ',' separated list of key=value (int/float) for new zIndex ordering. If used defaults are background=0, categoryAxis=1, valueAxis=2, bars=3, barLabels=4, categoryAxisGrid=5, valueAxisGrid=6, annotations=7

### *Example*

```
def demo(self):
    """Shows basic use of a bar chart"""
    if self.__class__.__name__=='BarChart':
        raise NotImplementedError, 'Abstract Class BarChart has no demo'
    drawing = Drawing(200, 100)
    bc = self.__class__()
    drawing.add(bc)
    return drawing
```

### *Properties of Example Widget*

```
background = None
barLabelArray = None
barLabelFormat = None
barLabels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x2bc5f80>
```



```
barSpacing = 0
barWidth = 10
bars = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x2bc7050>
categoryAxis.categoryNames = None
categoryAxis.drawGridLast = False
categoryAxis.gridEnd = None
categoryAxis.gridStart = None
categoryAxis.gridStrokeColor = Color(0,0,0,1)
categoryAxis.gridStrokeDashArray = None
categoryAxis.gridStrokeLineCap = 0
categoryAxis.gridStrokeLineJoin = 0
categoryAxis.gridStrokeMiterLimit = 10
categoryAxis.gridStrokeWidth = 0.25
categoryAxis.hiLLen = 0
categoryAxis.hiPad = 0
categoryAxis.joinAxis = None
categoryAxis.joinAxisMode = None
categoryAxis.joinAxisPos = None
categoryAxis.labelAxisMode = 'axis'
categoryAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x2bc5e18>
categoryAxis.loLLen = 0
categoryAxis.loPad = 0
categoryAxis.reverseDirection = 0
categoryAxis.strokeColor = Color(0,0,0,1)
categoryAxis.strokeDashArray = None
categoryAxis.strokeLineCap = 0
categoryAxis.strokeLineJoin = 0
categoryAxis.strokeMiterLimit = 10
categoryAxis.strokeWidth = 1
categoryAxis.style = 'parallel'
categoryAxis.tickDown = 5
categoryAxis.tickShift = 0
categoryAxis.tickUp = 0
categoryAxis.visible = 1
categoryAxis.visibleAxis = 1
categoryAxis.visibleGrid = 0
categoryAxis.visibleLabels = 1
categoryAxis.visibleTicks = 1
categoryNALabel = None
data = [(100, 110, 120, 130), (70, 80, 85, 90)]
debug = 0
fillColor = None
groupSpacing = 5
height = 85
naLabel = None
reversePlotOrder = 0
strokeColor = None
strokeWidth = 1
useAbsolute = 0
valueAxis.abf_ignore_zero = False
valueAxis.avoidBoundFrac = None
valueAxis.avoidBoundSpace = None
valueAxis.drawGridLast = False
valueAxis.forceZero = 0
valueAxis.gridEnd = None
valueAxis.gridStart = None
valueAxis.gridStrokeColor = Color(0,0,0,1)
valueAxis.gridStrokeDashArray = None
valueAxis.gridStrokeLineCap = 0
valueAxis.gridStrokeLineJoin = 0
valueAxis.gridStrokeMiterLimit = 10
valueAxis.gridStrokeWidth = 0.25
valueAxis.hiLLen = 0
valueAxis.joinAxis = None
valueAxis.joinAxisMode = None
valueAxis.joinAxisPos = None
valueAxis.keepTickLabelsInside = 0
valueAxis.labelAxisMode = 'axis'
valueAxis.labelTextFormat = None
valueAxis.labelTextPostFormat = None
valueAxis.labelTextScale = None
valueAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x2bc5ef0>
valueAxis.loLLen = 0
valueAxis.maximumTicks = 7
valueAxis.minimumTickSpacing = 10
valueAxis.origShiftIPC = None
valueAxis.origShiftMin = None
valueAxis.origShiftSpecialValue = None
valueAxis.rangeRound = 'none'
```

```
valueAxis.requiredRange = 0
valueAxis.reverseDirection = 0
valueAxis.skipEndL = 'none'
valueAxis.strokeColor = Color(0,0,0,1)
valueAxis.strokeDashArray = None
valueAxis.strokeLineCap = 0
valueAxis.strokeLineJoin = 0
valueAxis.strokeMiterLimit = 10
valueAxis.strokeWidth = 1
valueAxis.style = 'normal'
valueAxis.subGridEnd = None
valueAxis.subGridStart = None
valueAxis.subGridStrokeColor = Color(0,0,0,1)
valueAxis.subGridStrokeDashArray = None
valueAxis.subGridStrokeLineCap = 0
valueAxis.subGridStrokeLineJoin = 0
valueAxis.subGridStrokeMiterLimit = 10
valueAxis.subGridStrokeWidth = 0.25
valueAxis.subTickHi = 0
valueAxis.subTickLo = 0
valueAxis.subTickNum = 0
valueAxis.tickAxisMode = 'axis'
valueAxis.tickLeft = 5
valueAxis.tickRight = 0
valueAxis.valueMax = None
valueAxis.valueMin = None
valueAxis.valueStep = None
valueAxis.visible = 1
valueAxis.visibleAxis = 1
valueAxis.visibleGrid = 0
valueAxis.visibleLabels = 1
valueAxis.visibleSubGrid = 0
valueAxis.visibleSubTicks = 0
valueAxis.visibleTicks = 1
valueAxis.zrangePref = 0
width = 180
x = 20
y = 10
zIndexOverrides = None
```

## VerticalBarChart3D(BarChart3D, VerticalBarChart)

### Public Attributes

**annotations** list of callables, will be called with self, xscale, yscale.

**background** Handle to background object e.g. Rect(0,0,width,height).

**barLabelArray** explicit array of bar label values, must match size of data if present.

**barLabelCallOut** Callout function(label) label.\_callOutInfo = (self,g,rowNo,colNo,x,y,width,height,x00,y00,x0,y0)

**barLabelFormat** Formatting string or function used for bar labels.

**barLabels** Handle to the list of bar labels.

**barRecord** callable(bar,label=labelText,value=value,\*\*kwds) to record bar information

**barSpacing** Width between individual bars.

**barWidth** The width of an individual bar.

**bars** Handle of the individual bars.

**categoryAxis** Handle of the category axis.

**categoryLabelBarOrder** where any label bar should appear first/last

**categoryLabelBarSize** width to leave for a category label to go between categories.

**categoryNALabel** Label to use for a group of N/A values.

**data** Data to be plotted, list of (lists of) numbers.

**debug** Used only for debugging.

**fillColor** Color of the plot area interior.

**groupSpacing** Width between groups of bars.

**height** Height of the chart.

**naLabel** Label to use for N/A values.

**reversePlotOrder** If true, reverse common category plot order.

**strokeColor** Color of the plot area border.

**strokeWidth** Width plot area border.

**theta\_x** dx/dz

**theta\_y** dy/dz

**useAbsolute** Flag to use absolute spacing values; use string of gsb for finer control (g=groupSpacing,s=barSpacing,b=barWidth).

**valueAxis** Handle of the value axis.

**width** Width of the chart.

**x** X position of the lower-left corner of the chart.

**y** Y position of the lower-left corner of the chart.

**zDepth** depth of an individual series

**zIndexOverrides** None (the default ie use old z ordering scheme) or a ',' separated list of key=value (int/float) for new zIndex ordering. If used defaults are background=0, categoryAxis=1, valueAxis=2, bars=3, barLabels=4, categoryAxisGrid=5, valueAxisGrid=6, annotations=7

**zSpace z gap around series***Example*

```
def demo(self):
    """Shows basic use of a bar chart"""
    if self.__class__.__name__=='BarChart':
        raise NotImplementedError, 'Abstract Class BarChart has no demo'
    drawing = Drawing(200, 100)
    bc = self.__class__()
    drawing.add(bc)
    return drawing
```

*Properties of Example Widget*

```
background = None
barLabelArray = None
barLabelFormat = None
barLabels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x2bdb2d8>
barSpacing = 0
barWidth = 10
bars = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x2bdb368>
categoryAxis.categoryNames = None
categoryAxis.drawGridLast = False
categoryAxis.gridEnd = None
categoryAxis.gridStart = None
categoryAxis.gridStrokeColor = Color(0,0,0,1)
categoryAxis.gridStrokeDashArray = None
categoryAxis.gridStrokeLineCap = 0
categoryAxis.gridStrokeLineJoin = 0
categoryAxis.gridStrokeMiterLimit = 10
categoryAxis.gridStrokeWidth = 0.25
categoryAxis.hiLen = 0
categoryAxis.hiPad = 0
categoryAxis.joinAxis = None
categoryAxis.joinAxisMode = None
categoryAxis.joinAxisPos = None
categoryAxis.labelAxisMode = 'axis'
categoryAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x2bdb170>
categoryAxis.loLen = 0
categoryAxis.loPad = 0
categoryAxis.reverseDirection = 0
categoryAxis.strokeColor = Color(0,0,0,1)
categoryAxis.strokeDashArray = None
categoryAxis.strokeLineCap = 0
categoryAxis.strokeLineJoin = 0
categoryAxis.strokeMiterLimit = 10
categoryAxis.strokeWidth = 1
categoryAxis.style = 'parallel'
categoryAxis.tickDown = 5
categoryAxis.tickShift = 0
categoryAxis.tickUp = 0
categoryAxis.visible = 1
categoryAxis.visibleAxis = 1
categoryAxis.visibleGrid = 0
categoryAxis.visibleLabels = 1
categoryAxis.visibleTicks = 1
categoryNALabel = None
data = [(100, 110, 120, 130), (70, 80, 85, 90)]
debug = 0
fillColor = None
groupSpacing = 5
height = 85
naLabel = None
reversePlotOrder = 0
strokeColor = None
strokeWidth = 1
useAbsolute = 0
valueAxis.abf_ignore_zero = False
valueAxis.avoidBoundFrac = None
valueAxis.avoidBoundSpace = None
valueAxis.drawGridLast = False
valueAxis.forceZero = 0
valueAxis.gridEnd = None
```

```
valueAxis.gridStart = None
valueAxis.gridStrokeColor = Color(0,0,0,1)
valueAxis.gridStrokeDashArray = None
valueAxis.gridStrokeLineCap = 0
valueAxis.gridStrokeLineJoin = 0
valueAxis.gridStrokeMiterLimit = 10
valueAxis.gridStrokeWidth = 0.25
valueAxis.hiLLen = 0
valueAxis.joinAxis = None
valueAxis.joinAxisMode = None
valueAxis.joinAxisPos = None
valueAxis.keepTickLabelsInside = 0
valueAxis.labelAxisMode = 'axis'
valueAxis.labelTextFormat = None
valueAxis.labelTextPostFormat = None
valueAxis.labelTextScale = None
valueAxis.labels = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x2bdb248>
valueAxis.loLLen = 0
valueAxis.maximumTicks = 7
valueAxis.minimumTickSpacing = 10
valueAxis.origShiftIPC = None
valueAxis.origShiftMin = None
valueAxis.origShiftSpecialValue = None
valueAxis.rangeRound = 'none'
valueAxis.requiredRange = 0
valueAxis.reverseDirection = 0
valueAxis.skipEndL = 'none'
valueAxis.strokeColor = Color(0,0,0,1)
valueAxis.strokeDashArray = None
valueAxis.strokeLineCap = 0
valueAxis.strokeLineJoin = 0
valueAxis.strokeMiterLimit = 10
valueAxis.strokeWidth = 1
valueAxis.style = 'normal'
valueAxis.subGridEnd = None
valueAxis.subGridStart = None
valueAxis.subGridStrokeColor = Color(0,0,0,1)
valueAxis.subGridStrokeDashArray = None
valueAxis.subGridStrokeLineCap = 0
valueAxis.subGridStrokeLineJoin = 0
valueAxis.subGridStrokeMiterLimit = 10
valueAxis.subGridStrokeWidth = 0.25
valueAxis.subTickHi = 0
valueAxis.subTickLo = 0
valueAxis.subTickNum = 0
valueAxis.tickAxisMode = 'axis'
valueAxis.tickLeft = 5
valueAxis.tickRight = 0
valueAxis.valueMax = None
valueAxis.valueMin = None
valueAxis.valueStep = None
valueAxis.visible = 1
valueAxis.visibleAxis = 1
valueAxis.visibleGrid = 0
valueAxis.visibleLabels = 1
valueAxis.visibleSubGrid = 0
valueAxis.visibleSubTicks = 0
valueAxis.visibleTicks = 1
valueAxis.zrangePref = 0
width = 180
x = 20
y = 10
zIndexOverrides = None
```

## Functions

### `sampleH0a( ... )`

Make a slightly pathologic bar chart with only TWO data items.

#### *Example*

```
def sampleH0a():
    "Make a slightly pathologic bar chart with only TWO data items."

    drawing = Drawing(400, 200)

    data = [(13, 20)]

    bc = HorizontalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data

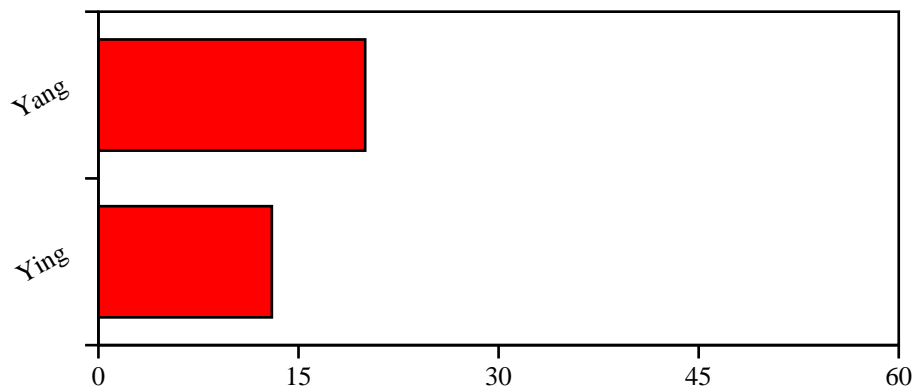
    bc.strokeColor = colors.black

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'se'
    bc.categoryAxis.labels.angle = 30
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    drawing.add(bc)

    return drawing
```



**sampleH0b( ... )**

Make a pathologic bar chart with only ONE data item.

*Example*

```
def sampleH0b():
    "Make a pathologic bar chart with only ONE data item."

    drawing = Drawing(400, 200)

    data = [(42,)]

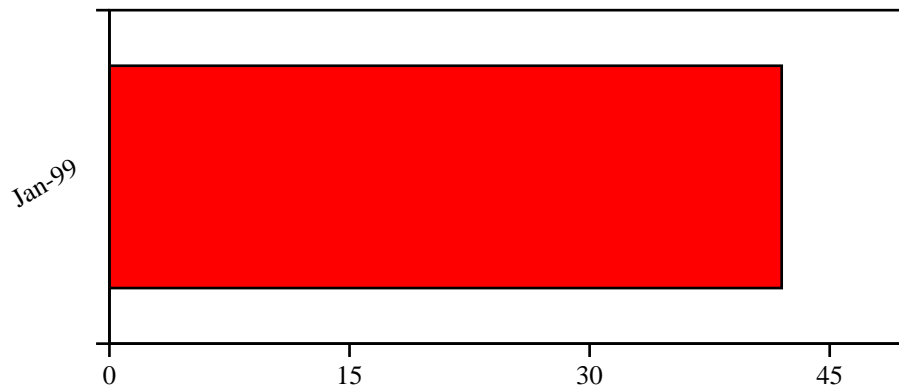
    bc = HorizontalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data
    bc.strokeColor = colors.black

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 50
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'se'
    bc.categoryAxis.labels.angle = 30
    bc.categoryAxis.categoryNames = ['Jan-99']

    drawing.add(bc)

    return drawing
```



**sampleH0c( ... )**

Make a really pathologic bar chart with NO data items at all!

*Example*

```
def sampleH0c():
    "Make a really pathologic bar chart with NO data items at all!"

    drawing = Drawing(400, 200)

    data = [()]

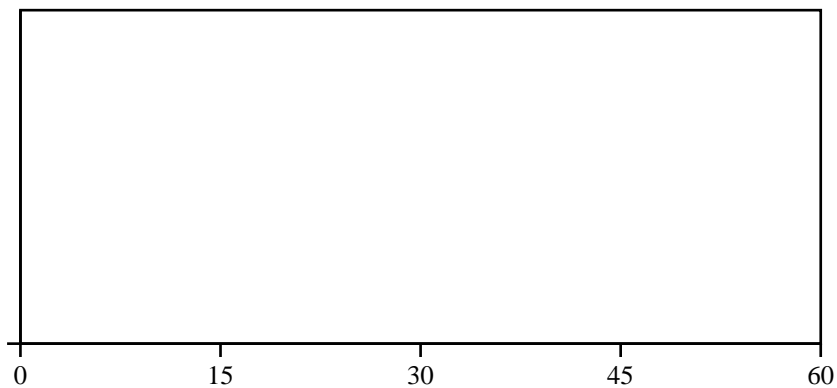
    bc = HorizontalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data
    bc.strokeColor = colors.black

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'se'
    bc.categoryAxis.labels.angle = 30
    bc.categoryAxis.categoryNames = []

    drawing.add(bc)

    return drawing
```





**sampleH1( ... )**

Sample of multi-series bar chart.

*Example*

```
def sampleH1():
    "Sample of multi-series bar chart."

    drawing = Drawing(400, 200)

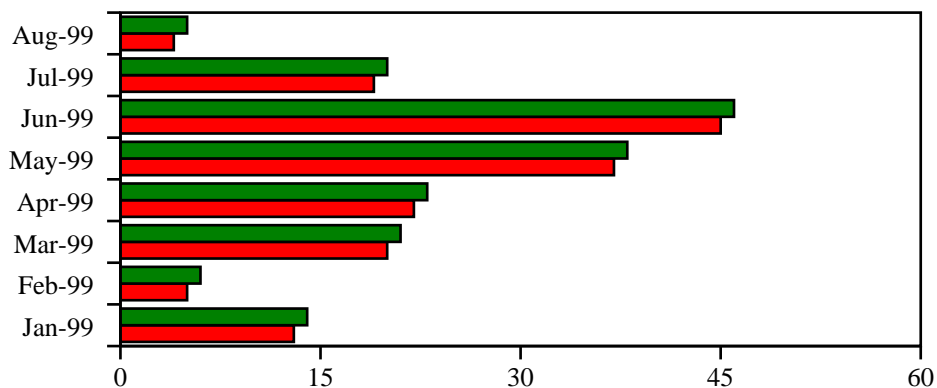
    data = [
        (13, 5, 20, 22, 37, 45, 19, 4),
        (14, 6, 21, 23, 38, 46, 20, 5)
    ]

    bc = HorizontalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data
    bc.strokeColor = colors.black

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'e'
    catNames = 'Jan Feb Mar Apr May Jun Jul Aug'.split(' ')
    catNames = map(lambda n:n+'-99', catNames)
    bc.categoryAxis.categoryNames = catNames
    drawing.add(bc, 'barchart')

    return drawing
```



**sampleH2a( ... )**

Sample of multi-series bar chart.

*Example*

```
def sampleH2a():
    "Sample of multi-series bar chart."

    data = [(2.4, -5.7, 2, 5, 9.2),
            (0.6, -4.9, -3, 4, 6.8)
            ]

    labels = ("Q3 2000", "Year to Date", "12 months",
              "Annualised\n3 years", "Since 07.10.99")

    drawing = Drawing(400, 200)

    bc = HorizontalBarChart()
    bc.x = 80
    bc.y = 50
    bc.height = 120
    bc.width = 300
    bc.data = data

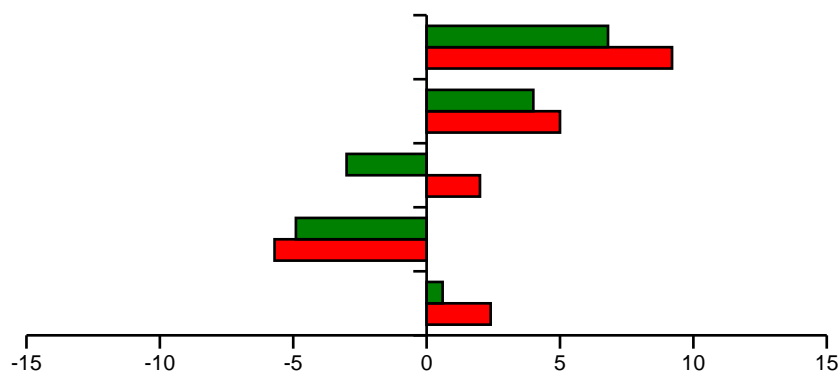
    bc.barSpacing = 0
    bc.groupSpacing = 10
    bc.barWidth = 10

    bc.valueAxis.valueMin = -15
    bc.valueAxis.valueMax = +15
    bc.valueAxis.valueStep = 5
    bc.valueAxis.labels.fontName = 'Helvetica'
    bc.valueAxis.labels.fontSize = 8
    bc.valueAxis.labels.boxAnchor = 'n' # irrelevant (becomes 'c')
    bc.valueAxis.labels.textAnchor = 'middle'
    bc.valueAxis.configure(bc.data)

    bc.categoryAxis.categoryNames = labels
    bc.categoryAxis.labels.fontName = 'Helvetica'
    bc.categoryAxis.labels.fontSize = 8
    bc.categoryAxis.labels.dx = -150

    drawing.add(bc)

    return drawing
```



**sampleH2b( ... )**

Sample of multi-series bar chart.

*Example*

```
def sampleH2b():
    "Sample of multi-series bar chart."

    data = [(2.4, -5.7, 2, 5, 9.2),
            (0.6, -4.9, -3, 4, 6.8)]

    labels = ("Q3 2000", "Year to Date", "12 months",
              "Annualised\n3 years", "Since 07.10.99")

    drawing = Drawing(400, 200)

    bc = HorizontalBarChart()
    bc.x = 80
    bc.y = 50
    bc.height = 120
    bc.width = 300
    bc.data = data

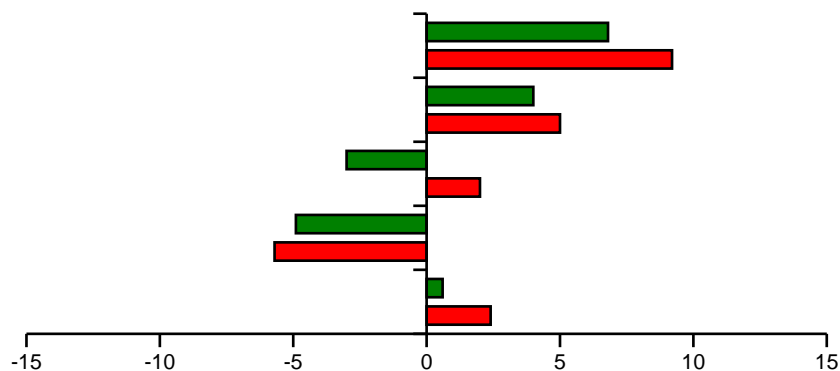
    bc.barSpacing = 5
    bc.groupSpacing = 10
    bc.barWidth = 10

    bc.valueAxis.valueMin = -15
    bc.valueAxis.valueMax = +15
    bc.valueAxis.valueStep = 5
    bc.valueAxis.labels.fontName = 'Helvetica'
    bc.valueAxis.labels.fontSize = 8
    bc.valueAxis.labels.boxAnchor = 'n' # irrelevant (becomes 'c')
    bc.valueAxis.labels.textAnchor = 'middle'

    bc.categoryAxis.categoryNames = labels
    bc.categoryAxis.labels.fontName = 'Helvetica'
    bc.categoryAxis.labels.fontSize = 8
    bc.categoryAxis.labels.dx = -150

    drawing.add(bc)

    return drawing
```



**sampleH2c( ... )**

Sample of multi-series bar chart.

*Example*

```
def sampleH2c():
    "Sample of multi-series bar chart."

    data = [(2.4, -5.7, 2, 5, 9.99),
            (0.6, -4.9, -3, 4, 9.99)
            ]

    labels = ("Q3 2000", "Year to Date", "12 months",
              "Annualised\n3 years", "Since 07.10.99")

    drawing = Drawing(400, 200)

    bc = HorizontalBarChart()
    bc.x = 80
    bc.y = 50
    bc.height = 120
    bc.width = 300
    bc.data = data

    bc.barSpacing = 2
    bc.groupSpacing = 10
    bc.barWidth = 10

    bc.valueAxis.valueMin = -15
    bc.valueAxis.valueMax = +15
    bc.valueAxis.valueStep = 5
    bc.valueAxis.labels.fontName = 'Helvetica'
    bc.valueAxis.labels.fontSize = 8
    bc.valueAxis.labels.boxAnchor = 'n'
    bc.valueAxis.labels.textAnchor = 'middle'

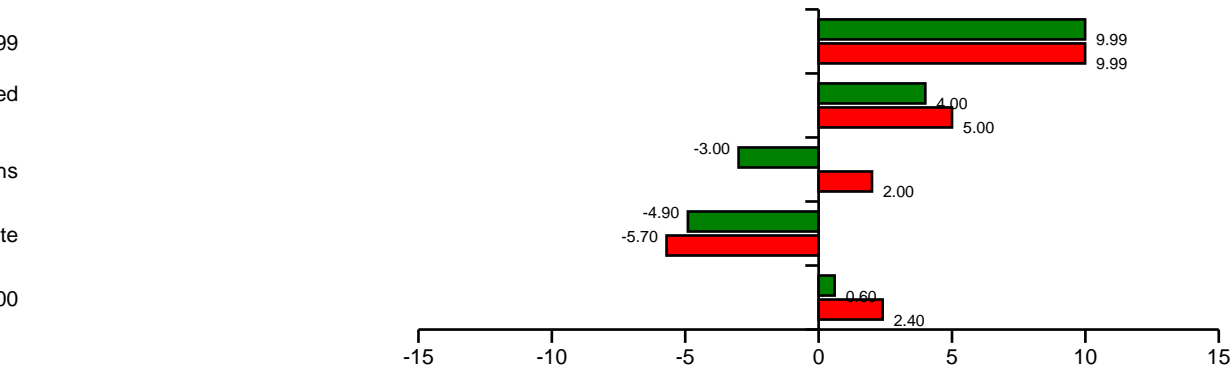
    bc.categoryAxis.categoryNames = labels
    bc.categoryAxis.labels.fontName = 'Helvetica'
    bc.categoryAxis.labels.fontSize = 8
    bc.categoryAxis.labels.dx = -150

    bc.barLabels.nudge = 10

    bc.barLabelFormat = '%0.2f'
    bc.barLabels.dx = 0
    bc.barLabels.dy = 0
    bc.barLabels.boxAnchor = 'n' # irrelevant (becomes 'c')
    bc.barLabels.fontName = 'Helvetica'
    bc.barLabels.fontSize = 6

    drawing.add(bc)

    return drawing
```



**sampleH3( ... )**

A really horizontal bar chart (compared to the equivalent faked one).

*Example*

```
def sampleH3():
    "A really horizontal bar chart (compared to the equivalent faked one)."
```

names = ("UK Equities", "US Equities", "European Equities", "Japanese Equities",  
 "Pacific (ex Japan) Equities", "Emerging Markets Equities",  
 "UK Bonds", "Overseas Bonds", "UK Index-Linked", "Cash")

series1 = (-1.5, 0.3, 0.5, 1.0, 0.8, 0.7, 0.4, 0.1, 1.0, 0.3)  
series2 = (0.0, 0.33, 0.55, 1.1, 0.88, 0.77, 0.44, 0.11, 1.10, 0.33)

assert len(names) == len(series1), "bad data"  
assert len(names) == len(series2), "bad data"

drawing = Drawing(400, 200)

bc = HorizontalBarChart()  
bc.x = 100  
bc.y = 20  
bc.height = 150  
bc.width = 250  
bc.data = (series1,)   
bc.bars.fillColor = colors.green

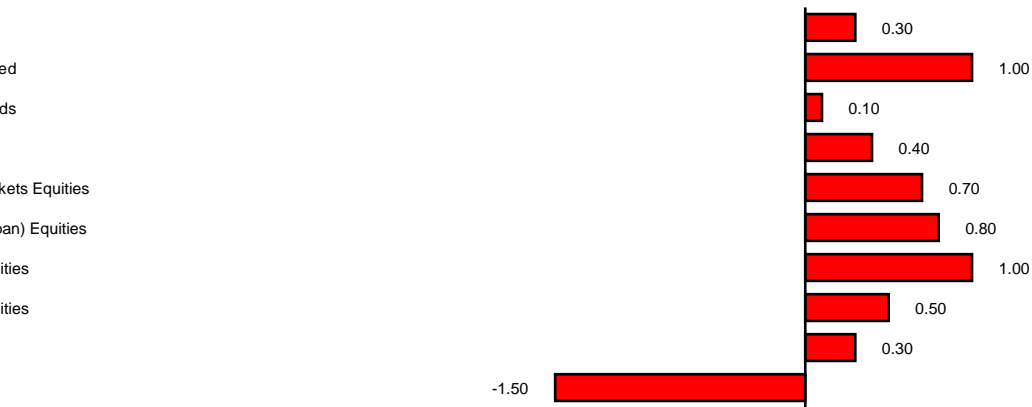
bc.barLabelFormat = '%0.2f'  
bc.barLabels.dx = 0  
bc.barLabels.dy = 0  
bc.barLabels.boxAnchor = 'w' # irrelevant (becomes 'c')  
bc.barLabels.fontName = 'Helvetica'  
bc.barLabels.fontSize = 6  
bc.barLabels.nudge = 10

bc.valueAxis.visible = 0  
bc.valueAxis.valueMin = -2  
bc.valueAxis.valueMax = +2  
bc.valueAxis.valueStep = 1

bc.categoryAxis.tickLeft = 0  
bc.categoryAxis.tickRight = 0  
bc.categoryAxis.categoryNames = names  
bc.categoryAxis.labels.boxAnchor = 'w'  
bc.categoryAxis.labels.dx = -170  
bc.categoryAxis.labels.fontName = 'Helvetica'  
bc.categoryAxis.labels.fontSize = 6

g = Group(bc)  
drawing.add(g)

return drawing



**sampleH4a( ... )**

A bar chart showing value axis region starting at *exactly* zero.

*Example*

```
def sampleH4a():
    "A bar chart showing value axis region starting at exactly zero."

    drawing = Drawing(400, 200)

    data = [(13, 20)]

    bc = HorizontalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data

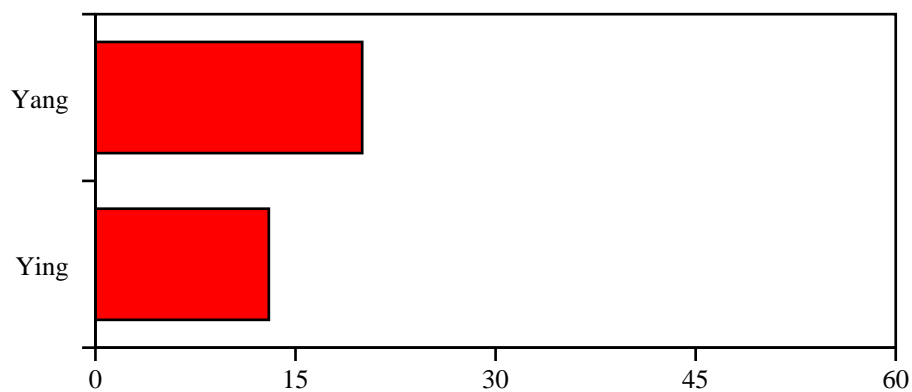
    bc.strokeColor = colors.black

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'e'
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    drawing.add(bc)

    return drawing
```





**sampleH4b( ... )**

A bar chart showing value axis region starting *\*below\** zero.

*Example*

```
def sampleH4b():
    "A bar chart showing value axis region starting *below* zero."

    drawing = Drawing(400, 200)

    data = [(13, 20)]

    bc = HorizontalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data

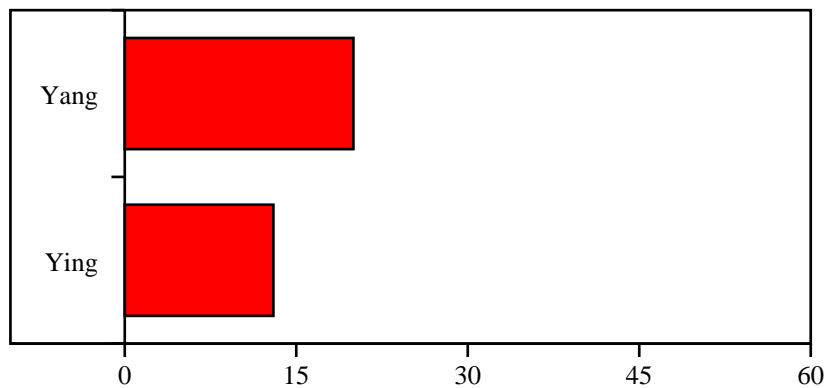
    bc.strokeColor = colors.black

    bc.valueAxis.valueMin = -10
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'e'
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    drawing.add(bc)

    return drawing
```



**sampleH4c( ... )**

A bar chart showing value axis region starting *above* zero.

*Example*

```
def sampleH4c():
    "A bar chart showing value axis region starting above zero."

    drawing = Drawing(400, 200)

    data = [(13, 20)]

    bc = HorizontalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data

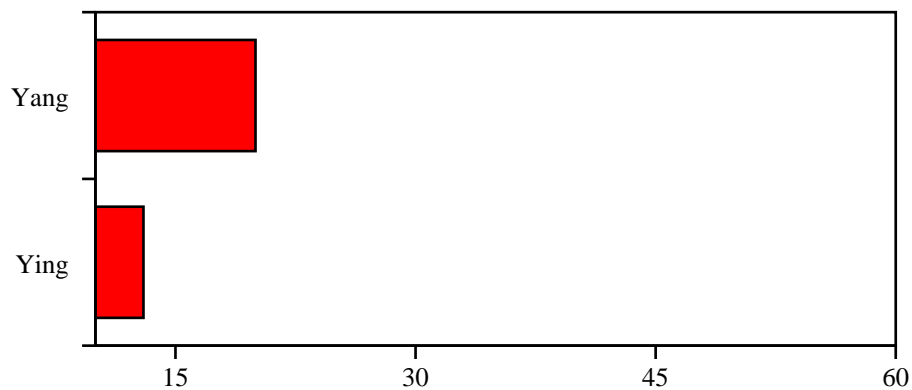
    bc.strokeColor = colors.black

    bc.valueAxis.valueMin = 10
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'e'
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    drawing.add(bc)

    return drawing
```



**sampleH4d( ... )**

A bar chart showing value axis region entirely *\*below\** zero.

*Example*

```
def sampleH4d():
    "A bar chart showing value axis region entirely *below* zero."

    drawing = Drawing(400, 200)

    data = [(-13, -20)]

    bc = HorizontalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data

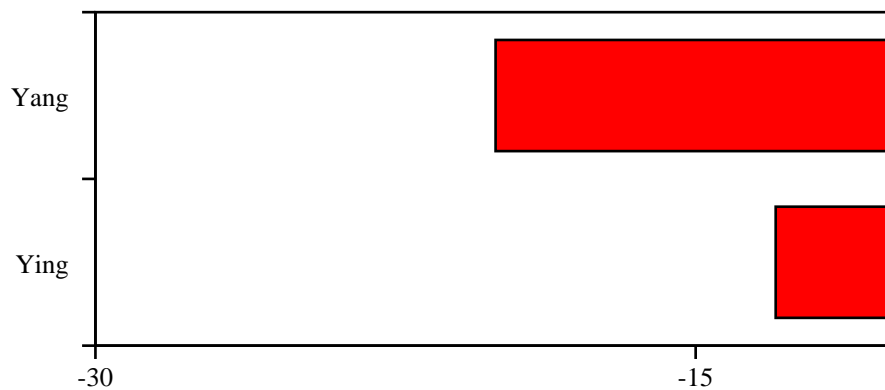
    bc.strokeColor = colors.black

    bc.valueAxis.valueMin = -30
    bc.valueAxis.valueMax = -10
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'e'
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    drawing.add(bc)

    return drawing
```



**sampleH5a( ... )**

A simple bar chart with no expressed spacing attributes.

*Example*

```
def sampleH5a():
    "A simple bar chart with no expressed spacing attributes."

    drawing = Drawing(400, 200)

    data = dataSample5

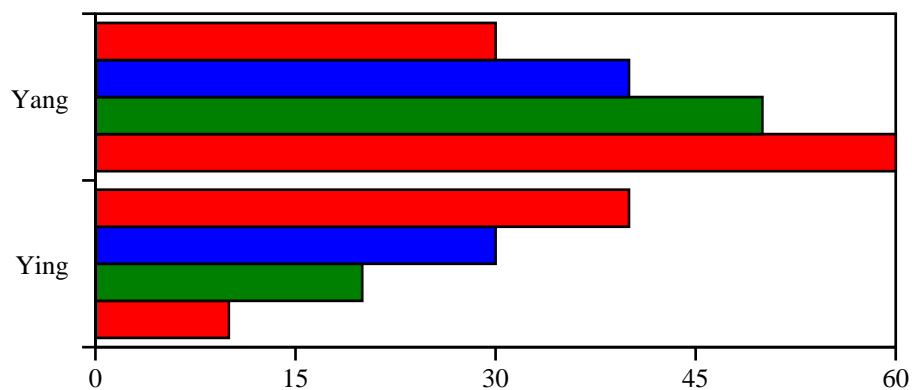
    bc = HorizontalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data
    bc.strokeColor = colors.black

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'e'
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    drawing.add(bc)

    return drawing
```



**sampleH5b( ... )**

A simple bar chart with proportional spacing.

*Example*

```
def sampleH5b():
    "A simple bar chart with proportional spacing."

    drawing = Drawing(400, 200)

    data = dataSample5

    bc = HorizontalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data
    bc.strokeColor = colors.black

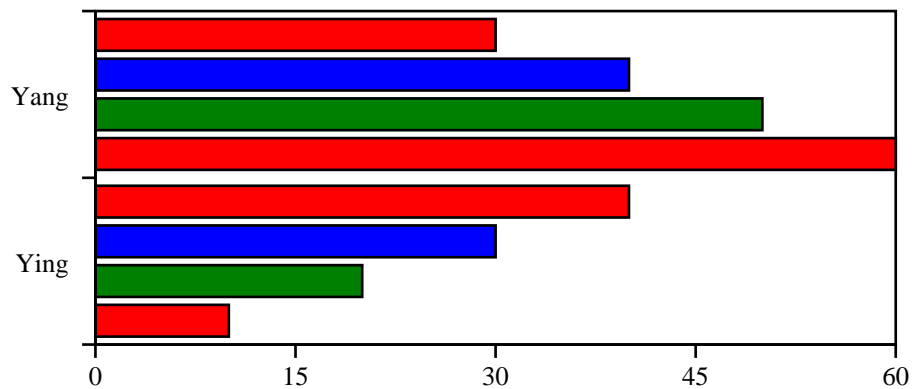
    bc.useAbsolute = 0
    bc.barWidth = 40
    bc.groupSpacing = 20
    bc.barSpacing = 10

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'e'
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    drawing.add(bc)

    return drawing
```



**sampleH5c1( ... )**

A simple bar chart with absolute spacing.

*Example*

```
def sampleH5c1():
    "A simple bar chart with absolute spacing."

    drawing = Drawing(400, 200)

    data = dataSample5

    bc = HorizontalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data
    bc.strokeColor = colors.black

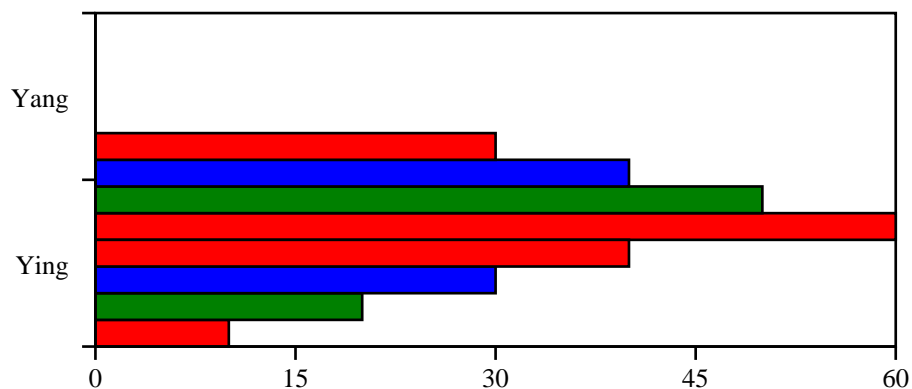
    bc.useAbsolute = 1
    bc.barWidth = 10
    bc.groupSpacing = 0
    bc.barSpacing = 0

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'e'
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    drawing.add(bc)

    return drawing
```



**sampleH5c2( ... )**

Simple bar chart with absolute spacing.

*Example*

```
def sampleH5c2():
    "Simple bar chart with absolute spacing."

    drawing = Drawing(400, 200)

    data = dataSample5

    bc = HorizontalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data
    bc.strokeColor = colors.black

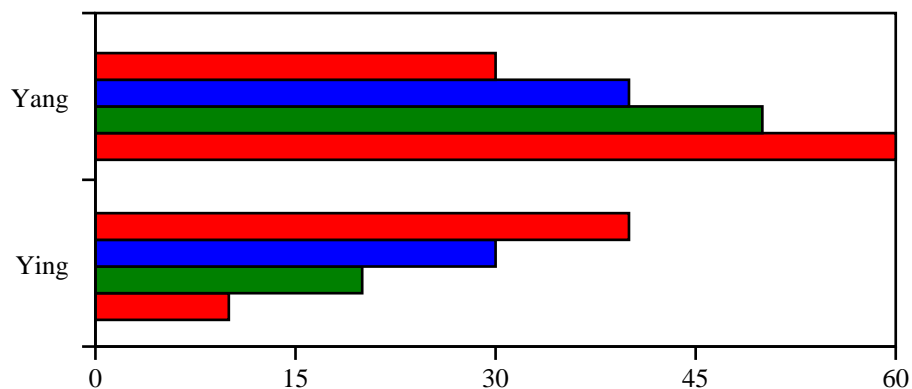
    bc.useAbsolute = 1
    bc.barWidth = 10
    bc.groupSpacing = 20
    bc.barSpacing = 0

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'e'
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    drawing.add(bc)

    return drawing
```



**sampleH5c3( ... )**

Simple bar chart with absolute spacing.

*Example*

```
def sampleH5c3():
    "Simple bar chart with absolute spacing."

    drawing = Drawing(400, 200)

    data = dataSample5

    bc = HorizontalBarChart()
    bc.x = 50
    bc.y = 20
    bc.height = 155
    bc.width = 300
    bc.data = data
    bc.strokeColor = colors.black

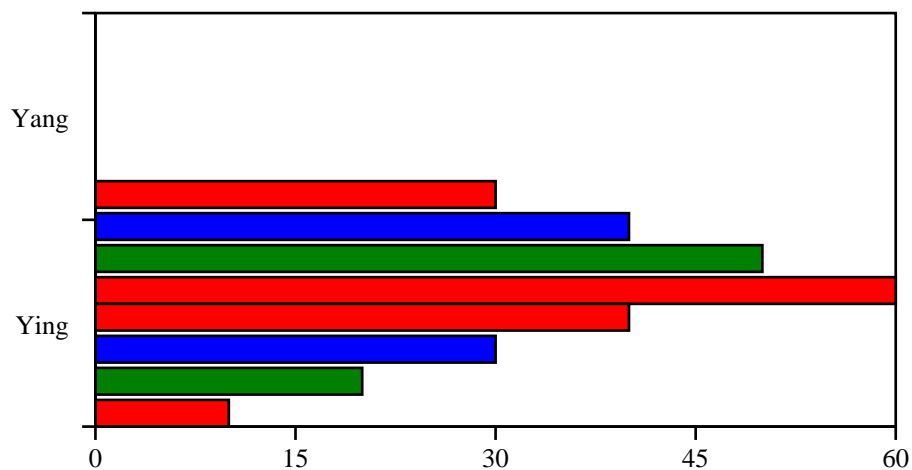
    bc.useAbsolute = 1
    bc.barWidth = 10
    bc.groupSpacing = 0
    bc.barSpacing = 2

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'e'
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    drawing.add(bc)

    return drawing
```





**sampleH5c4( ... )**

Simple bar chart with absolute spacing.

*Example*

```
def sampleH5c4():
    "Simple bar chart with absolute spacing."

    drawing = Drawing(400, 200)

    data = dataSample5

    bc = HorizontalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data
    bc.strokeColor = colors.black

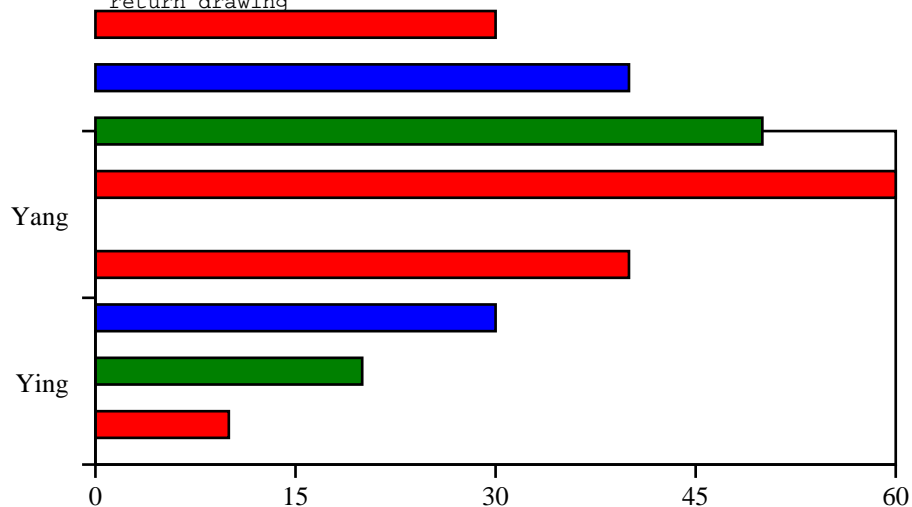
    bc.useAbsolute = 1
    bc.barWidth = 10
    bc.groupSpacing = 20
    bc.barSpacing = 10

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'e'
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    drawing.add(bc)

    return drawing
```



**sampleStacked1( ... )**

Simple bar chart using symbol attribute.

*Example*

```
def sampleStacked1():
    "Simple bar chart using symbol attribute."

    drawing = Drawing(400, 200)

    data = dataSample5

    bc = VerticalBarChart()
    bc.categoryAxis.style = 'stacked'
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data
    bc.strokeColor = colors.black

    bc.barWidth = 10
    bc.groupSpacing = 15
    bc.valueAxis.valueMin = 0

    bc.categoryAxis.labels.boxAnchor = 'e'
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    from reportlab.graphics.widgets.grids import ShadedRect
    bc.bars.symbol = ShadedRect()
    bc.bars.symbol.fillColorStart = colors.red
    bc.bars.symbol.fillColorEnd = colors.white
    bc.bars.symbol.orientation = 'vertical'
    bc.bars.symbol.cylinderMode = 1
    bc.bars.symbol.strokeWidth = 0

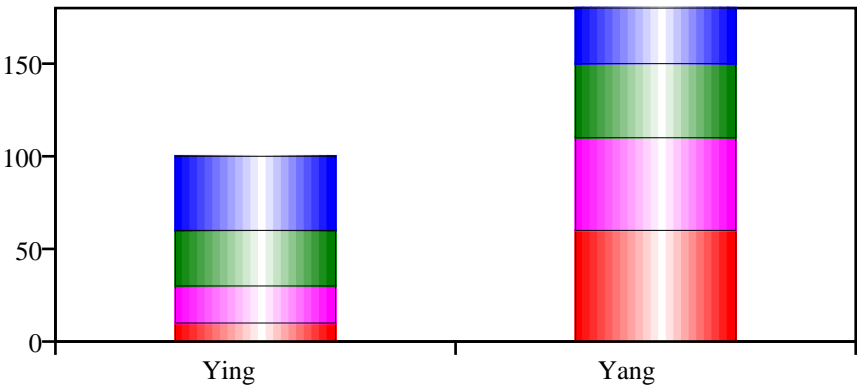
    bc.bars[1].symbol = ShadedRect()
    bc.bars[1].symbol.fillColorStart = colors.magenta
    bc.bars[1].symbol.fillColorEnd = colors.white
    bc.bars[1].symbol.orientation = 'vertical'
    bc.bars[1].symbol.cylinderMode = 1
    bc.bars[1].symbol.strokeWidth = 0

    bc.bars[2].symbol = ShadedRect()
    bc.bars[2].symbol.fillColorStart = colors.green
    bc.bars[2].symbol.fillColorEnd = colors.white
    bc.bars[2].symbol.orientation = 'vertical'
    bc.bars[2].symbol.cylinderMode = 1
    bc.bars[2].symbol.strokeWidth = 0

    bc.bars[3].symbol = ShadedRect()
    bc.bars[3].symbol.fillColorStart = colors.blue
    bc.bars[3].symbol.fillColorEnd = colors.white
    bc.bars[3].symbol.orientation = 'vertical'
    bc.bars[3].symbol.cylinderMode = 1
    bc.bars[3].symbol.strokeWidth = 0

    drawing.add(bc)

    return drawing
```



**sampleSymbol1( ... )**

Simple bar chart using symbol attribute.

*Example*

```
def sampleSymbol1():
    "Simple bar chart using symbol attribute."

    drawing = Drawing(400, 200)

    data = dataSample5

    bc = VerticalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data
    bc.strokeColor = colors.black

    bc.barWidth = 10
    bc.groupSpacing = 15
    bc.barSpacing = 3

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'e'
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    from reportlab.graphics.widgets.grids import ShadedRect
    sym1 = ShadedRect()
    sym1.fillColorStart = colors.black
    sym1.fillColorEnd = colors.blue
    sym1.orientation = 'horizontal'
    sym1.strokeWidth = 0

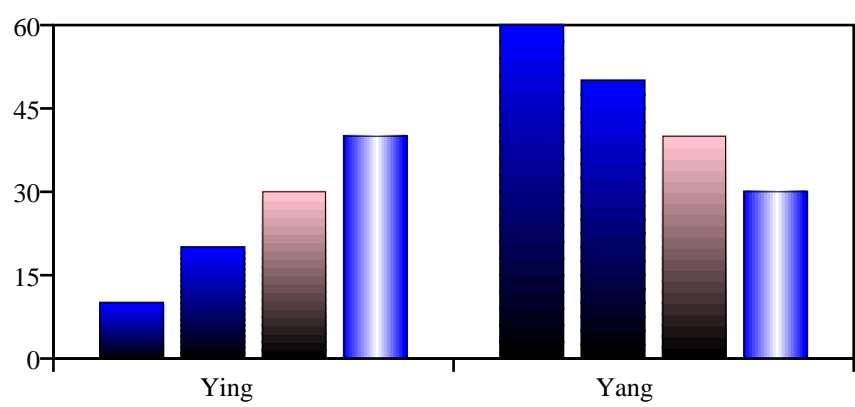
    sym2 = ShadedRect()
    sym2.fillColorStart = colors.black
    sym2.fillColorEnd = colors.pink
    sym2.orientation = 'horizontal'
    sym2.strokeWidth = 0

    sym3 = ShadedRect()
    sym3.fillColorStart = colors.blue
    sym3.fillColorEnd = colors.white
    sym3.orientation = 'vertical'
    sym3.cylinderMode = 1
    sym3.strokeWidth = 0

    bc.bars.symbol = sym1
    bc.bars[2].symbol = sym2
    bc.bars[3].symbol = sym3

    drawing.add(bc)

    return drawing
```



**sampleV0a( ... )**

A slightly pathologic bar chart with only TWO data items.

*Example*

```
def sampleV0a():
    "A slightly pathologic bar chart with only TWO data items."

    drawing = Drawing(400, 200)

    data = [(13, 20)]

    bc = VerticalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data

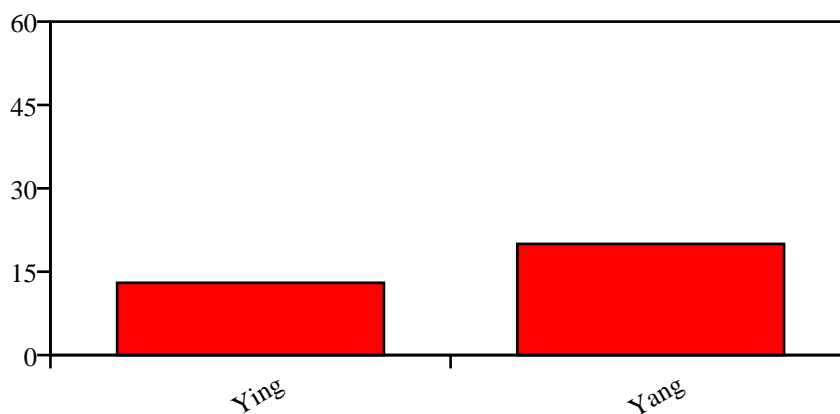
    bc.strokeColor = colors.black

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'ne'
    bc.categoryAxis.labels.dx = 8
    bc.categoryAxis.labels.dy = -2
    bc.categoryAxis.labels.angle = 30
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    drawing.add(bc)

    return drawing
```



**sampleV0b( ... )**

A pathologic bar chart with only ONE data item.

*Example*

```
def sampleV0b():
    "A pathologic bar chart with only ONE data item."

    drawing = Drawing(400, 200)

    data = [(42,)]

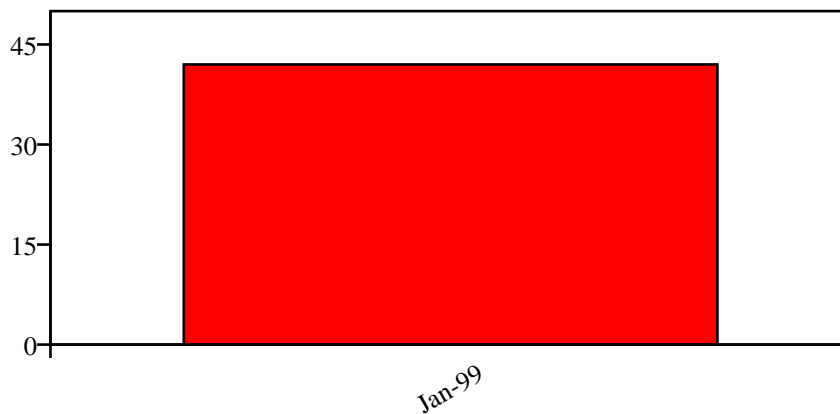
    bc = VerticalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data
    bc.strokeColor = colors.black

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 50
    bc.valueAxis.valueStep = 15

    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']

    drawing.add(bc)

    return drawing
```



**sampleV0c( ... )**

A really pathologic bar chart with NO data items at all!

*Example*

```
def sampleV0c():
    "A really pathologic bar chart with NO data items at all!"

    drawing = Drawing(400, 200)

    data = [()]

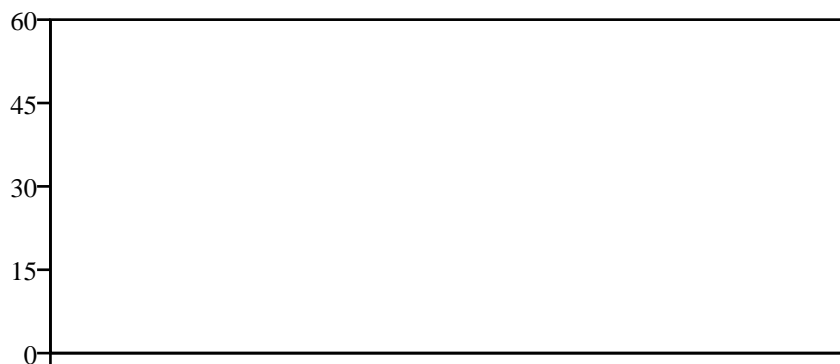
    bc = VerticalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data
    bc.strokeColor = colors.black

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'ne'
    bc.categoryAxis.labels.dx = 8
    bc.categoryAxis.labels.dy = -2
    bc.categoryAxis.categoryNames = []

    drawing.add(bc)

    return drawing
```





**sampleV1( ... )**

Sample of multi-series bar chart.

*Example*

```
def sampleV1():
    "Sample of multi-series bar chart."

    drawing = Drawing(400, 200)

    data = [
        (13, 5, 20, 22, 37, 45, 19, 4),
        (14, 6, 21, 23, 38, 46, 20, 5)
    ]

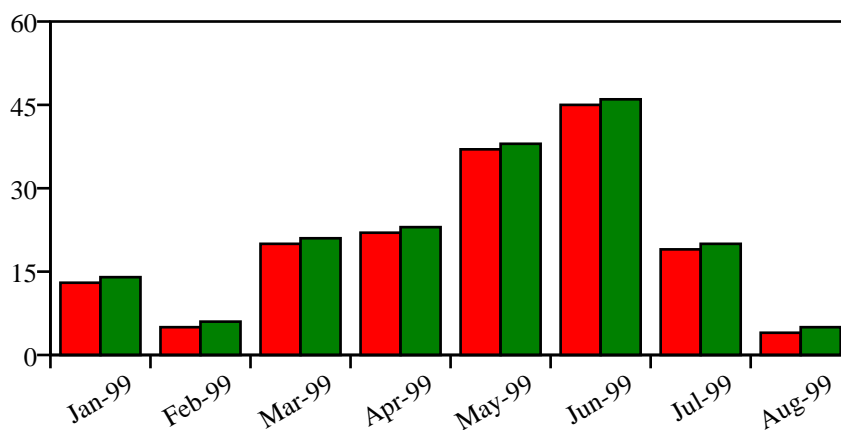
    bc = VerticalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data
    bc.strokeColor = colors.black

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'ne'
    bc.categoryAxis.labels.dx = 8
    bc.categoryAxis.labels.dy = -2
    bc.categoryAxis.labels.angle = 30

    catNames = 'Jan Feb Mar Apr May Jun Jul Aug'.split(' ')
    catNames = map(lambda n:n+'-99', catNames)
    bc.categoryAxis.categoryNames = catNames
    drawing.add(bc)

    return drawing
```



**sampleV2a( ... )**

Sample of multi-series bar chart.

*Example*

```
def sampleV2a():
    "Sample of multi-series bar chart."

    data = [(2.4, -5.7, 2, 5, 9.2),
            (0.6, -4.9, -3, 4, 6.8)]

    labels = ("Q3 2000", "Year to Date", "12 months",
              "Annualised\n3 years", "Since 07.10.99")

    drawing = Drawing(400, 200)

    bc = VerticalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 120
    bc.width = 300
    bc.data = data

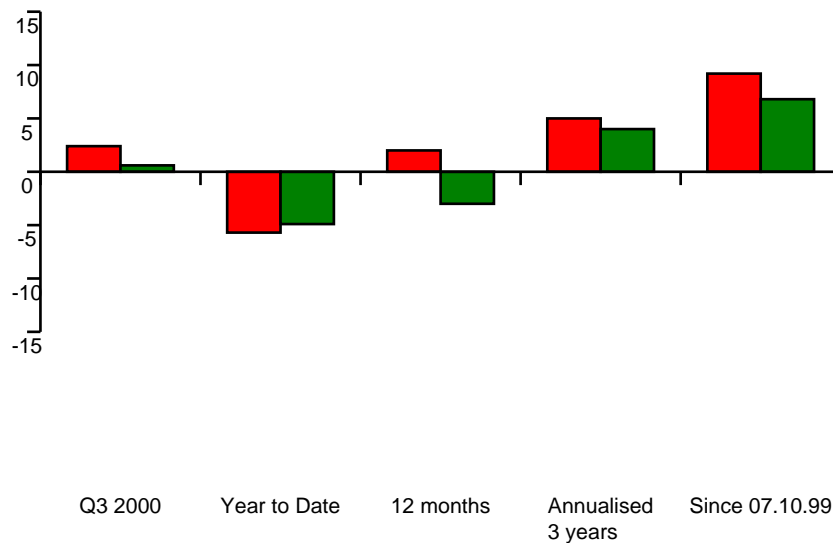
    bc.barSpacing = 0
    bc.groupSpacing = 10
    bc.barWidth = 10

    bc.valueAxis.valueMin = -15
    bc.valueAxis.valueMax = +15
    bc.valueAxis.valueStep = 5
    bc.valueAxis.labels.fontName = 'Helvetica'
    bc.valueAxis.labels.fontSize = 8
    bc.valueAxis.labels.boxAnchor = 'n' # irrelevant (becomes 'c')
    bc.valueAxis.labels.textAnchor = 'middle'

    bc.categoryAxis.categoryNames = labels
    bc.categoryAxis.labels.fontName = 'Helvetica'
    bc.categoryAxis.labels.fontSize = 8
    bc.categoryAxis.labels.dy = -60

    drawing.add(bc)

    return drawing
```



**sampleV2b( ... )**

Sample of multi-series bar chart.

*Example*

```
def sampleV2b():
    "Sample of multi-series bar chart."

    data = [(2.4, -5.7, 2, 5, 9.2),
            (0.6, -4.9, -3, 4, 6.8)]

    labels = ("Q3 2000", "Year to Date", "12 months",
              "Annualised\n3 years", "Since 07.10.99")

    drawing = Drawing(400, 200)

    bc = VerticalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 120
    bc.width = 300
    bc.data = data

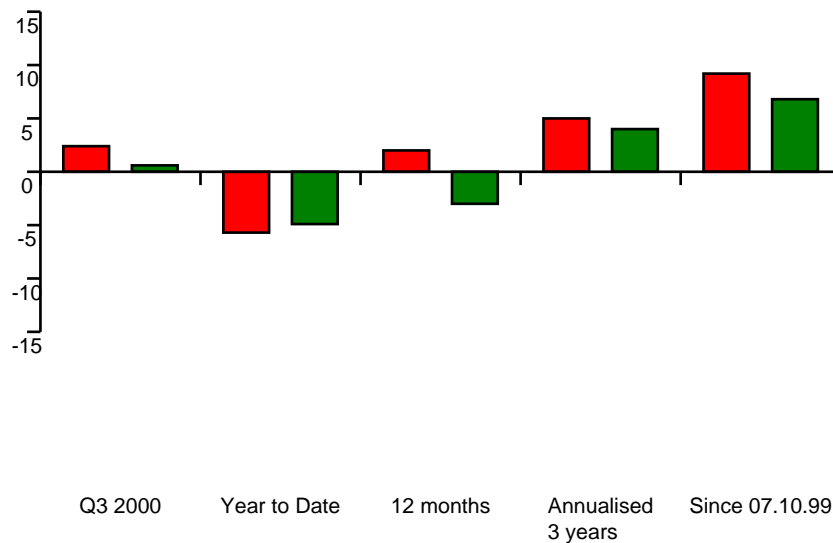
    bc.barSpacing = 5
    bc.groupSpacing = 10
    bc.barWidth = 10

    bc.valueAxis.valueMin = -15
    bc.valueAxis.valueMax = +15
    bc.valueAxis.valueStep = 5
    bc.valueAxis.labels.fontName = 'Helvetica'
    bc.valueAxis.labels.fontSize = 8
    bc.valueAxis.labels.boxAnchor = 'n' # irrelevant (becomes 'c')
    bc.valueAxis.labels.textAnchor = 'middle'

    bc.categoryAxis.categoryNames = labels
    bc.categoryAxis.labels.fontName = 'Helvetica'
    bc.categoryAxis.labels.fontSize = 8
    bc.categoryAxis.labels.dy = -60

    drawing.add(bc)

    return drawing
```



**sampleV2c( ... )**

Sample of multi-series bar chart.

*Example*

```
def sampleV2c():
    "Sample of multi-series bar chart."

    data = [(2.4, -5.7, 2, 5, 9.99),
            (0.6, -4.9, -3, 4, 9.99)
            ]

    labels = ("Q3 2000", "Year to Date", "12 months",
             "Annualised\n3 years", "Since 07.10.99")

    drawing = Drawing(400, 200)

    bc = VerticalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 120
    bc.width = 300
    bc.data = data

    bc.barSpacing = 2
    bc.groupSpacing = 10
    bc.barWidth = 10

    bc.valueAxis.valueMin = -15
    bc.valueAxis.valueMax = +15
    bc.valueAxis.valueStep = 5
    bc.valueAxis.labels.fontName = 'Helvetica'
    bc.valueAxis.labels.fontSize = 8

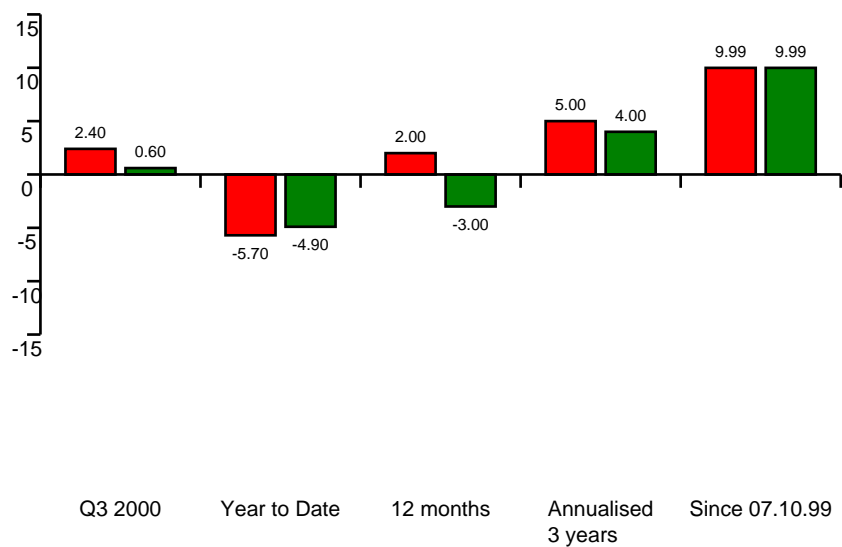
    bc.categoryAxis.categoryNames = labels
    bc.categoryAxis.labels.fontName = 'Helvetica'
    bc.categoryAxis.labels.fontSize = 8
    bc.valueAxis.labels.boxAnchor = 'n'
    bc.valueAxis.labels.textAnchor = 'middle'
    bc.categoryAxis.labels.dy = -60

    bc.barLabels.nudge = 10

    bc.barLabelFormat = '%0.2f'
    bc.barLabels.dx = 0
    bc.barLabels.dy = 0
    bc.barLabels.boxAnchor = 'n' # irrelevant (becomes 'c')
    bc.barLabels.fontName = 'Helvetica'
    bc.barLabels.fontSize = 6

    drawing.add(bc)

    return drawing
```



**sampleV3( ... )**

Faked horizontal bar chart using a vertical real one (deprecated).

*Example*

```
def sampleV3():
    "Faked horizontal bar chart using a vertical real one (deprecated)."
```

```
    names = ("UK Equities", "US Equities", "European Equities", "Japanese Equities",
             "Pacific (ex Japan) Equities", "Emerging Markets Equities",
             "UK Bonds", "Overseas Bonds", "UK Index-Linked", "Cash")

    series1 = (-1.5, 0.3, 0.5, 1.0, 0.8, 0.7, 0.4, 0.1, 1.0, 0.3)
    series2 = (0.0, 0.33, 0.55, 1.1, 0.88, 0.77, 0.44, 0.11, 1.10, 0.33)

    assert len(names) == len(series1), "bad data"
    assert len(names) == len(series2), "bad data"

    drawing = Drawing(400, 200)

    bc = VerticalBarChart()
    bc.x = 0
    bc.y = 0
    bc.height = 100
    bc.width = 150
    bc.data = (series1,)
    bc.bars.fillColor = colors.green

    bc.barLabelFormat = '%0.2f'
    bc.barLabels.dx = 0
    bc.barLabels.dy = 0
    bc.barLabels.boxAnchor = 'w' # irrelevant (becomes 'c')
    bc.barLabels.angle = 90
    bc.barLabels.fontName = 'Helvetica'
    bc.barLabels.fontSize = 6
    bc.barLabels.nudge = 10

    bc.valueAxis.visible = 0
    bc.valueAxis.valueMin = -2
    bc.valueAxis.valueMax = +2
    bc.valueAxis.valueStep = 1

    bc.categoryAxis.tickUp = 0
    bc.categoryAxis.tickDown = 0
    bc.categoryAxis.categoryNames = names
    bc.categoryAxis.labels.angle = 90
    bc.categoryAxis.labels.boxAnchor = 'w'
    bc.categoryAxis.labels.dx = 0
    bc.categoryAxis.labels.dy = -125
    bc.categoryAxis.labels.fontName = 'Helvetica'
    bc.categoryAxis.labels.fontSize = 6

    g = Group(bc)
    g.translate(100, 175)
    g.rotate(-90)

    drawing.add(g)

    return drawing
```



**sampleV4a( ... )**

A bar chart showing value axis region starting at *exactly* zero.

*Example*

```
def sampleV4a():
    "A bar chart showing value axis region starting at exactly zero."

    drawing = Drawing(400, 200)

    data = [(13, 20)]

    bc = VerticalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data

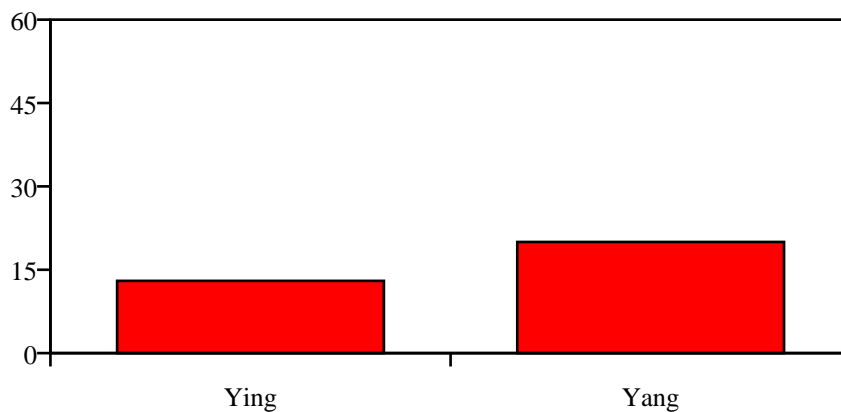
    bc.strokeColor = colors.black

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'n'
    bc.categoryAxis.labels.dy = -5
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    drawing.add(bc)

    return drawing
```





**sampleV4b( ... )**

A bar chart showing value axis region starting \*below\* zero.

*Example*

```
def sampleV4b():
    "A bar chart showing value axis region starting *below* zero."

    drawing = Drawing(400, 200)

    data = [(13, 20)]

    bc = VerticalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data

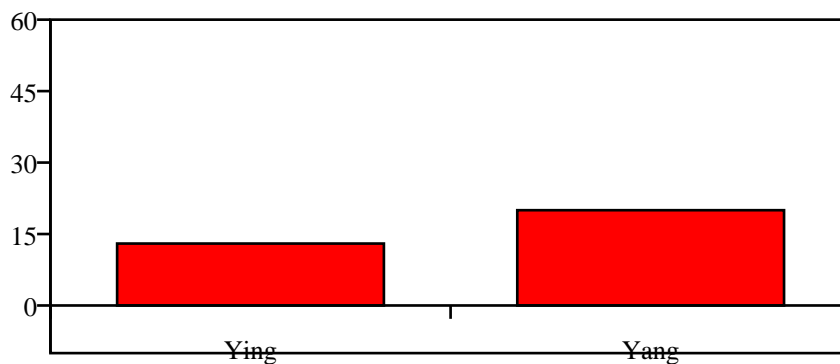
    bc.strokeColor = colors.black

    bc.valueAxis.valueMin = -10
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'n'
    bc.categoryAxis.labels.dy = -5
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    drawing.add(bc)

    return drawing
```



**sampleV4c( ... )**

A bar chart showing value axis region starting \*above\* zero.

*Example*

```
def sampleV4c():
    "A bar chart showing value axis region starting *above* zero."

    drawing = Drawing(400, 200)

    data = [(13, 20)]

    bc = VerticalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data

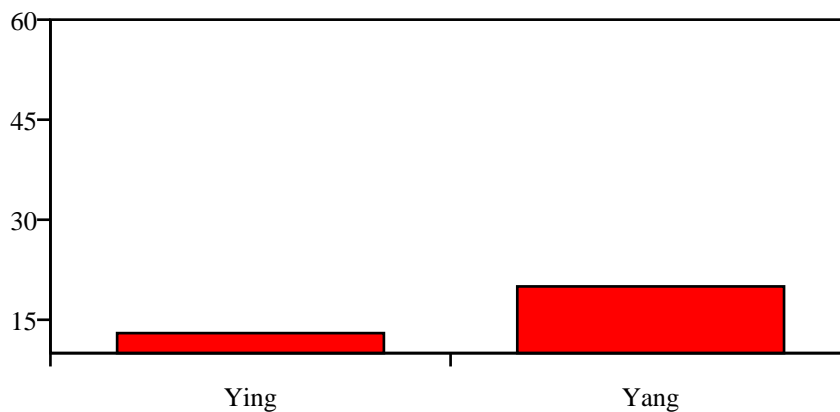
    bc.strokeColor = colors.black

    bc.valueAxis.valueMin = 10
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'n'
    bc.categoryAxis.labels.dy = -5
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    drawing.add(bc)

    return drawing
```



**sampleV4d( ... )**

A bar chart showing value axis region entirely *\*below\** zero.

*Example*

```
def sampleV4d():
    "A bar chart showing value axis region entirely *below* zero."

    drawing = Drawing(400, 200)

    data = [(-13, -20)]

    bc = VerticalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data

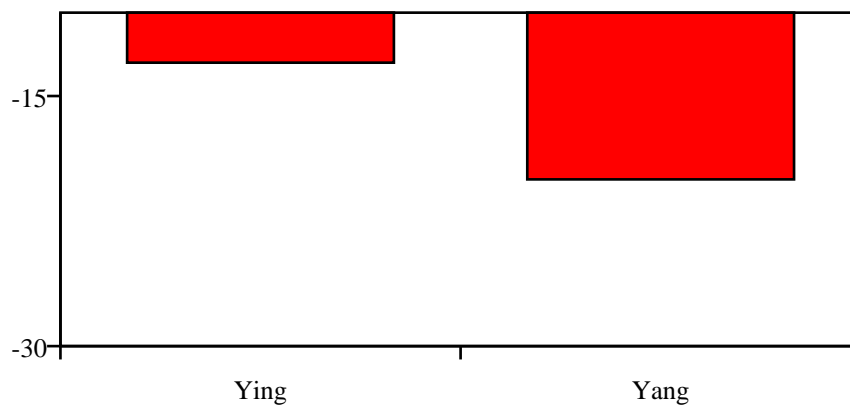
    bc.strokeColor = colors.black

    bc.valueAxis.valueMin = -30
    bc.valueAxis.valueMax = -10
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'n'
    bc.categoryAxis.labels.dy = -5
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    drawing.add(bc)

    return drawing
```



**sampleV5a( ... )**

A simple bar chart with no expressed spacing attributes.

*Example*

```
def sampleV5a():
    "A simple bar chart with no expressed spacing attributes."

    drawing = Drawing(400, 200)

    data = dataSample5

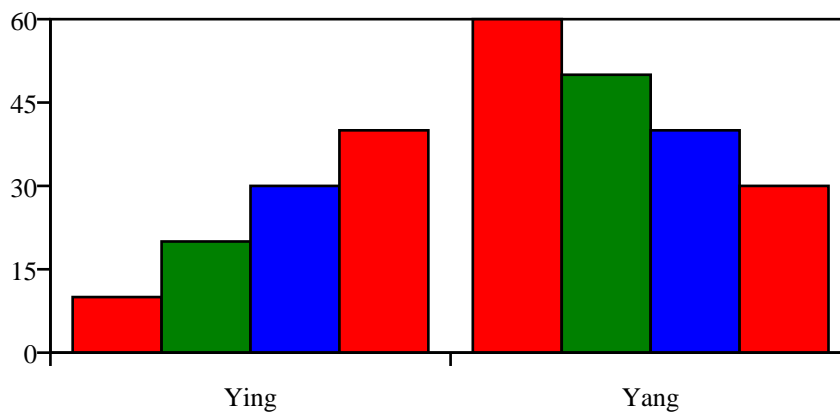
    bc = VerticalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data
    bc.strokeColor = colors.black

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'n'
    bc.categoryAxis.labels.dy = -5
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    drawing.add(bc)

    return drawing
```



**sampleV5b( ... )**

A simple bar chart with proportional spacing.

*Example*

```
def sampleV5b():
    "A simple bar chart with proportional spacing."

    drawing = Drawing(400, 200)

    data = dataSample5

    bc = VerticalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data
    bc.strokeColor = colors.black

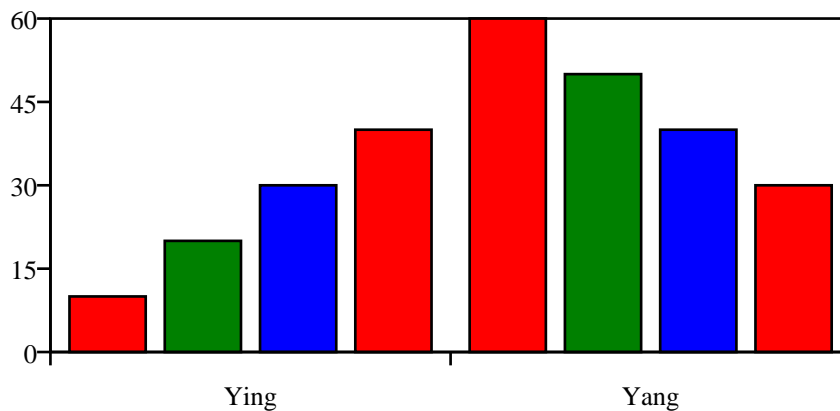
    bc.useAbsolute = 0
    bc.barWidth = 40
    bc.groupSpacing = 20
    bc.barSpacing = 10

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'n'
    bc.categoryAxis.labels.dy = -5
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    drawing.add(bc)

    return drawing
```



**sampleV5c1( ... )**

Make sampe simple bar chart but with absolute spacing.

*Example*

```
def sampleV5c1():
    "Make sampe simple bar chart but with absolute spacing."

    drawing = Drawing(400, 200)

    data = dataSample5

    bc = VerticalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data
    bc.strokeColor = colors.black

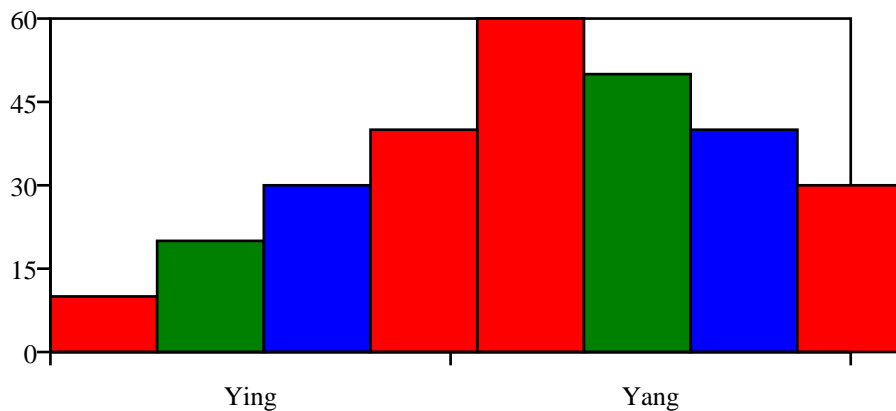
    bc.useAbsolute = 1
    bc.barWidth = 40
    bc.groupSpacing = 0
    bc.barSpacing = 0

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'n'
    bc.categoryAxis.labels.dy = -5
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    drawing.add(bc)

    return drawing
```



**sampleV5c2( ... )**

Make sampe simple bar chart but with absolute spacing.

*Example*

```
def sampleV5c2():
    "Make sampe simple bar chart but with absolute spacing."

    drawing = Drawing(400, 200)

    data = dataSample5

    bc = VerticalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data
    bc.strokeColor = colors.black

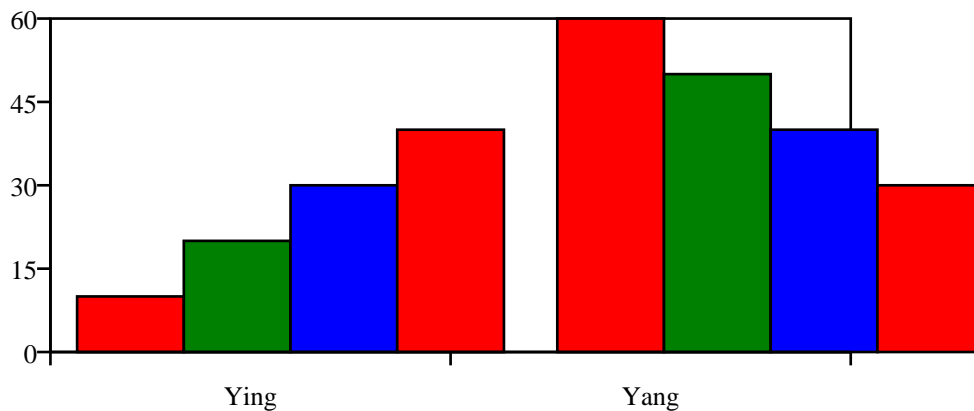
    bc.useAbsolute = 1
    bc.barWidth = 40
    bc.groupSpacing = 20
    bc.barSpacing = 0

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'n'
    bc.categoryAxis.labels.dy = -5
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    drawing.add(bc)

    return drawing
```



**sampleV5c3( ... )**

Make sampe simple bar chart but with absolute spacing.

*Example*

```
def sampleV5c3():
    "Make sampe simple bar chart but with absolute spacing."

    drawing = Drawing(400, 200)

    data = dataSample5

    bc = VerticalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data
    bc.strokeColor = colors.black

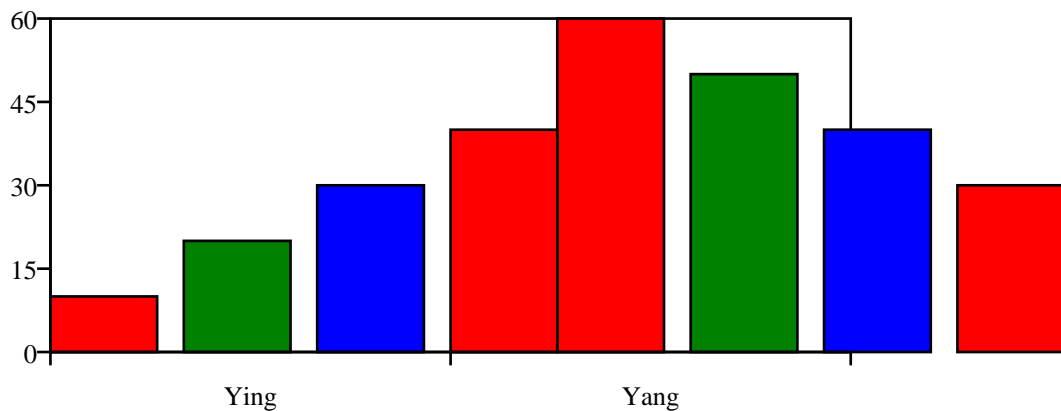
    bc.useAbsolute = 1
    bc.barWidth = 40
    bc.groupSpacing = 0
    bc.barSpacing = 10

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'n'
    bc.categoryAxis.labels.dy = -5
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    drawing.add(bc)

    return drawing
```





**sampleV5c4( ... )**

Make sampe simple bar chart but with absolute spacing.

*Example*

```
def sampleV5c4():
    "Make sampe simple bar chart but with absolute spacing."

    drawing = Drawing(400, 200)

    data = dataSample5

    bc = VerticalBarChart()
    bc.x = 50
    bc.y = 50
    bc.height = 125
    bc.width = 300
    bc.data = data
    bc.strokeColor = colors.black

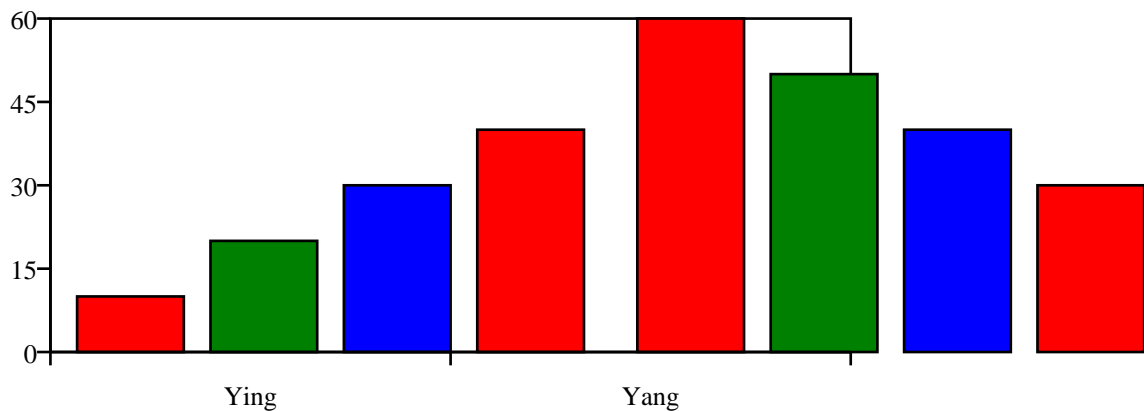
    bc.useAbsolute = 1
    bc.barWidth = 40
    bc.groupSpacing = 20
    bc.barSpacing = 10

    bc.valueAxis.valueMin = 0
    bc.valueAxis.valueMax = 60
    bc.valueAxis.valueStep = 15

    bc.categoryAxis.labels.boxAnchor = 'n'
    bc.categoryAxis.labels.dy = -5
    bc.categoryAxis.categoryNames = ['Ying', 'Yang']

    drawing.add(bc)

    return drawing
```



## legends

This will be a collection of legends to be used with charts.

## Classes

### Legend(Widget)

A simple legend containing rectangular swatches and strings.

The swatches are filled rectangles whenever the respective color object in 'colorNamePairs' is a subclass of Color in reportlab.lib.colors. Otherwise the object passed instead is assumed to have 'x', 'y', 'width' and 'height' attributes. A legend then tries to set them or catches any error. This lets you plug-in any widget you like as a replacement for the default rectangular swatches.

Strings can be nicely aligned left or right to the swatches.

### Public Attributes

**alignment** Alignment of text with respect to swatches

**autoXPadding** x Padding between columns if deltax=None

**autoYPadding** y Padding between rows if deltay=None

**boxAnchor** Anchor point for the legend area

**callout** a user callout(self,g,x,y,(color,text))

**colEndCallout** a user callout(self,g, x, xt, y,width, lWidth)

**colorNamePairs** List of color/name tuples (color can also be widget)

**columnMaximum** Max. number of items per column

**deltax** x-distance between neighbouring swatches

**deltay** y-distance between neighbouring swatches

**dividerColor** dividerLines color

**dividerDashArray** Dash array for dividerLines.

**dividerLines** If 1 we have dividers between the rows | 2 for extra top | 4 for bottom

**dividerOffsX** divider lines X offsets

**dividerOffsY** dividerLines Y offset

**dividerWidth** dividerLines width

**dx** Width of swatch rectangle

**dxTextSpace** Distance between swatch rectangle and text

**dy** Height of swatch rectangle

**fillColor** swatches filling color

**fontName** Font name of the strings

**fontSize** Font size of the strings

**strokeColor** Border color of the swatches

**strokeWidth** Width of the border color of the swatches

**subCols** subColumn properties

**swatchCallout** a user swatch callout(self,g,x,y,i,(col,name),swatch)

**swatchMarker** None, Auto() or makeMarker('Diamond') ...

**swdx** x position adjustment for the swatch

**swdy** y position adjustment for the swatch

**variColumn** If true column widths may vary (default is false)

**x** x-coordinate of upper-left reference point

**y** y-coordinate of upper-left reference point

**yGap** Additional gap between rows

### *Example*

```
def demo(self):
    "Make sample legend."

    d = Drawing(200, 100)

    legend = Legend()
    legend.alignment = 'left'
    legend.x = 0
    legend.y = 100
    legend.dxTextSpace = 5
    items = 'red green blue yellow pink black white'.split()
    items = map(lambda i:(getattr(colors, i), i), items)
    legend.colorNamePairs = items

    d.add(legend, 'legend')

    return d
```

### *Properties of Example Widget*

```
alignment = 'left'
autoXPadding = 5
autoYPadding = 2
boxAnchor = 'nw'
colEndCallout = None
colorNamePairs = [(Color(1,0,0,1), 'red'),
                  (Color(0,0,1,1), 'blue'),
                  (Color(0,.501961,0,1), 'green'),
                  (Color(1,.752941,.796078,1), 'pink'),
                  (Color(1,1,0,1), 'yellow')]
columnMaximum = 3
deltax = 75
deltay = 20
dividerColor = Color(0,0,0,1)
dividerDashArray = None
dividerLines = 0
dividerOffsX = (0, 0)
dividerOffsY = 0
dividerWidth = 0.5
dx = 10
dxTextSpace = 10
dy = 10
fillColor = Color(0,0,0,1)
fontName = 'Times-Roman'
fontSize = 10
strokeColor = Color(0,0,0,1)
strokeWidth = 1
subCols = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x1535b48>
swatchMarker = None
swdx = 0
swdy = 0
variColumn = 0
```

```
x = 0  
y = 0  
yGap = 0
```

## LineLegend ( Legend )

A subclass of Legend for drawing legends with lines as the swatches rather than rectangles. Useful for lineCharts and linePlots. Should be similar in all other ways the the standard Legend class.

### Public Attributes

**alignment** Alignment of text with respect to swatches  
**autoXPadding** x Padding between columns if deltax=None  
**autoYPadding** y Padding between rows if deltax=None  
**boxAnchor** Anchor point for the legend area  
**callout** a user callout(self,g,x,y,(color,text))  
**colEndCallout** a user callout(self,g, x, xt, y,width, lWidth)  
**colorNamePairs** List of color/name tuples (color can also be widget)  
**columnMaximum** Max. number of items per column  
**deltax** x-distance between neighbouring swatches  
**deltay** y-distance between neighbouring swatches  
**dividerColor** dividerLines color  
**dividerDashArray** Dash array for dividerLines.  
**dividerLines** If 1 we have dividers between the rows | 2 for extra top | 4 for bottom  
**dividerOffsX** divider lines X offsets  
**dividerOffsY** dividerLines Y offset  
**dividerWidth** dividerLines width  
**dx** Width of swatch rectangle  
**dxTextSpace** Distance between swatch rectangle and text  
**dy** Height of swatch rectangle  
**fillColor** swatches filling color  
**fontName** Font name of the strings  
**fontSize** Font size of the strings  
**strokeColor** Border color of the swatches  
**strokeWidth** Width of the border color of the swatches  
**subCols** subColumn properties  
**swatchCallout** a user swatch callout(self,g,x,y,i,(col,name),swatch)  
**swatchMarker** None, Auto() or makeMarker('Diamond') ...  
**swdx** x position adjustment for the swatch  
**swdy** y position adjustment for the swatch  
**variColumn** If true column widths may vary (default is false)  
**x** x-coordinate of upper-left reference point  
**y** y-coordinate of upper-left reference point

**yGap** Additional gap between rows*Example*

```
def demo(self):
    "Make sample legend."

    d = Drawing(200, 100)

    legend = Legend()
    legend.alignment = 'left'
    legend.x = 0
    legend.y = 100
    legend.dxTextSpace = 5
    items = 'red green blue yellow pink black white'.split()
    items = map(lambda i:(getattr(colors, i), i), items)
    legend.colorNamePairs = items

    d.add(legend, 'legend')

    return d
```

*Properties of Example Widget*

```
alignment = 'left'
autoXPadding = 5
autoYPadding = 2
boxAnchor = 'nw'
colEndCallout = None
colorNamePairs = [(Color(1,0,0,1), 'red'),
                  (Color(0,0,1,1), 'blue'),
                  (Color(0,.501961,0,1), 'green'),
                  (Color(1,.752941,.796078,1), 'pink'),
                  (Color(1,1,0,1), 'yellow')]
columnMaximum = 3
deltax = 75
deltay = 20
dividerColor = Color(0,0,0,1)
dividerDashArray = None
dividerLines = 0
dividerOffsX = (0, 0)
dividerOffsY = 0
dividerWidth = 0.5
dx = 10
dxTextSpace = 10
dy = 2
fillColor = Color(0,0,0,1)
fontName = 'Times-Roman'
fontSize = 10
strokeColor = Color(0,0,0,1)
strokeWidth = 1
subCols = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x15474d0>
swatchMarker = None
swdx = 0
swdy = 0
variColumn = 0
x = 0
y = 0
yGap = 0
```

## LineSwatch(Widget)

basically a Line with properties added so it can be used in a LineLegend

### Public Attributes

**height** used for line strokeWidth

**strokeColor** color of swatch line

**strokeDashArray** dash array for swatch line

**width** length of swatch line

**x** x-coordinate for swatch line start point

**y** y-coordinate for swatch line start point

### Example

```
def demo(self):
    msg = "demo() must be implemented for each Widget!"
    raise shapes.NotImplementedError, msg
```

### Properties of Example Widget

```
height = 1
strokeColor = Color(1,0,0,1)
strokeDashArray = None
width = 20
x = 0
y = 0
```

## Functions

### `sample1c( ... )`

Make sample legend.

#### *Example*








```
def sample1c():
    "Make sample legend."

    d = Drawing(200, 100)

    legend = Legend()
    legend.alignment = 'right'
    legend.x = 0
    legend.y = 100
    legend.dxTextSpace = 5
    items = 'red green blue yellow pink black white'.split()
    items = map(lambda i:(getattr(colors, i), i), items)
    legend.colorNamePairs = items

    d.add(legend, 'legend')

    return d
```

 red	 yellow	 white
 green	 pink	
 blue	 black	



**sample2c( ... )**

Make sample legend.

*Example*








```
def sample2c():
    "Make sample legend."

    d = Drawing(200, 100)

    legend = Legend()
    legend.alignment = 'right'
    legend.x = 20
    legend.y = 90
    legend.deltax = 60
    legend.dxTextSpace = 10
    legend.columnMaximum = 4
    items = 'red green blue yellow pink black white'.split()
    items = map(lambda i:(getattr(colors, i), i), items)
    legend.colorNamePairs = items

    d.add(legend, 'legend')

    return d
```

	red		pink
	green		black
	blue		white
	yellow		

**sample3( ... )**

Make sample legend with line swatches.







*Example*

```
def sample3():
    "Make sample legend with line swatches."

    d = Drawing(200, 100)

    legend = LineLegend()
    legend.alignment = 'right'
    legend.x = 20
    legend.y = 90
    legend.deltax = 60
    legend.dxTextSpace = 10
    legend.columnMaximum = 4
    items = 'red green blue yellow pink black white'.split()
    items = map(lambda i:(getattr(colors, i), i), items)
    legend.colorNamePairs = items
    d.add(legend, 'legend')

    return d
```

	red		pink
	green		black
	blue		white
	yellow		

**sample3a( ... )**

Make sample legend with line swatches and dasharrays on the lines.

*Example*

```
def sample3a():
    "Make sample legend with line swatches and dasharrays on the lines."

    d = Drawing(200, 100)

    legend = LineLegend()
    legend.alignment = 'right'
    legend.x = 20
    legend.y = 90
    legend.deltax = 60
    legend.dxTextSpace = 10
    legend.columnMaximum = 4
    items = 'red green blue yellow pink black white'.split()
    darrays = ([2,1], [2,5], [2,2,5,5], [1,2,3,4], [4,2,3,4], [1,2,3,4,5,6], [1])
    cnp = []
    for i in range(0, len(items)):
        l = LineSwatch()
        l.strokeColor = getattr(colors, items[i])
        l.strokeDashArray = darrays[i]
        cnp.append((l, items[i]))
    legend.colorNamePairs = cnp
    d.add(legend, 'legend')

    return d
```

...	red	- -	pink
■ ■	green	· ·	black
■ ■	blue		white
■ ■	yellow		

## doughnut

Doughnut chart

Produces a circular chart like the doughnut charts produced by Excel.  
Can handle multiple series (which produce concentric 'rings' in the chart).

## Classes

### Doughnut (AbstractPieChart)

#### Public Attributes

**checkLabelOverlap** If true check and attempt to fix standard label overlaps(default off)

**data** list of numbers defining sector sizes; need not sum to 1

**direction** 'clockwise' or 'anticlockwise'

**height** height of doughnut bounding box. Need not be same as height.

**labels** optional list of labels to use for each data point

**sideLabels** If true attempt to make chart with labels along side and pointers

**simpleLabels** If true(default) use String not super duper WedgeLabel

**slices** collection of sector descriptor objects

**startAngle** angle of first slice; like the compass, 0 is due North

**width** width of doughnut bounding box. Need not be same as width.

**x** X position of the chart within its container.

**y** Y position of the chart within its container.

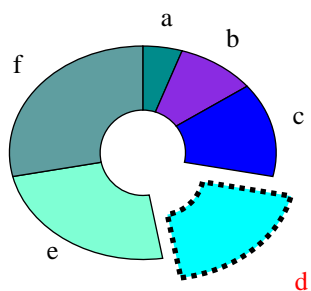
#### Example

```
def demo(self):
    d = Drawing(200, 100)

    dn = Doughnut()
    dn.x = 50
    dn.y = 10
    dn.width = 100
    dn.height = 80
    dn.data = [10,20,30,40,50,60]
    dn.labels = ['a','b','c','d','e','f']

    dn.slices.strokeWidth=0.5
    dn.slices[3].popout = 10
    dn.slices[3].strokeWidth = 2
    dn.slices[3].strokeDashArray = [2,2]
    dn.slices[3].labelRadius = 1.75
    dn.slices[3].fontColor = colors.red
    dn.slices[0].fillColor = colors.darkcyan
    dn.slices[1].fillColor = colors.blueviolet
    dn.slices[2].fillColor = colors.blue
    dn.slices[3].fillColor = colors.cyan
    dn.slices[4].fillColor = colors.aquamarine
    dn.slices[5].fillColor = colors.cadetblue
    dn.slices[6].fillColor = colors.lightcoral

    d.add(dn)
    return d
```



### *Properties of Example Widget*

```
checkLabelOverlap = 0
data = [1, 1]
direction = 'clockwise'
height = 100
labels = None
sideLabels = 0
simpleLabels = 1
slices = <reportlab.graphics.widgetbase.TypedPropertyCollection instance at 0x2f00ea8>
startAngle = 90
width = 100
x = 0
y = 0
```

## Functions

### **sample1( ... )**

Make up something from the individual Sectors

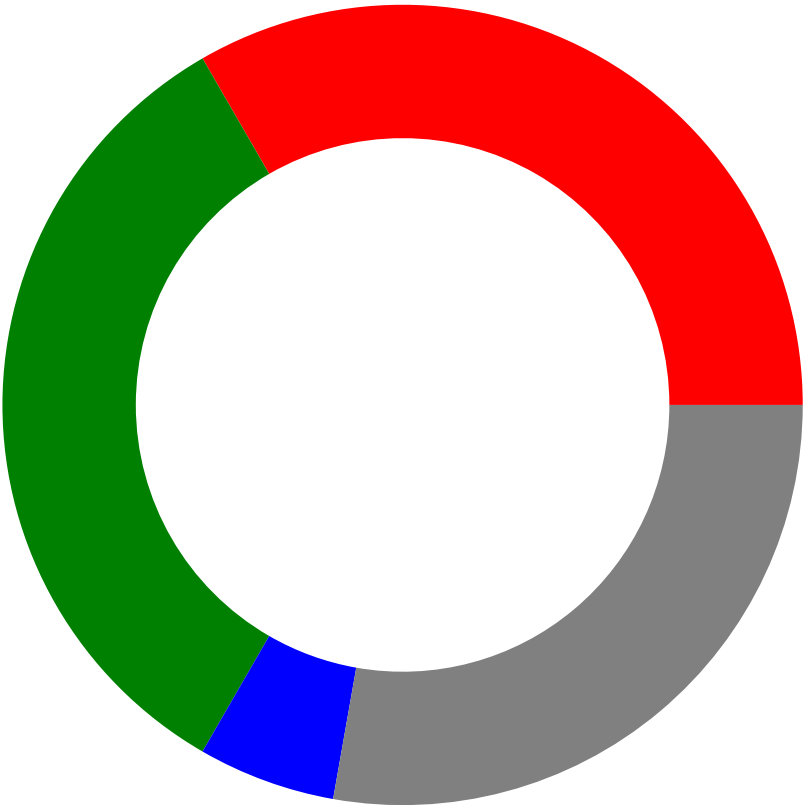
#### *Example*

```
def sample1():
    "Make up something from the individual Sectors"

    d = Drawing(400, 400)
    g = Group()

    s1 = Wedge(centerx=200, centery=200, radius=150, startangleddegrees=0, endangleddegrees=120, radius=150)
    s1.fillColor=colors.red
    s1.strokeColor=None
    d.add(s1)
    s2 = Wedge(centerx=200, centery=200, radius=150, startangleddegrees=120, endangleddegrees=240, radius=150)
    s2.fillColor=colors.green
    s2.strokeColor=None
    d.add(s2)
    s3 = Wedge(centerx=200, centery=200, radius=150, startangleddegrees=240, endangleddegrees=260, radius=150)
    s3.fillColor=colors.blue
    s3.strokeColor=None
    d.add(s3)
    s4 = Wedge(centerx=200, centery=200, radius=150, startangleddegrees=260, endangleddegrees=360, radius=150)
    s4.fillColor=colors.gray
    s4.strokeColor=None
    d.add(s4)

    return d
```

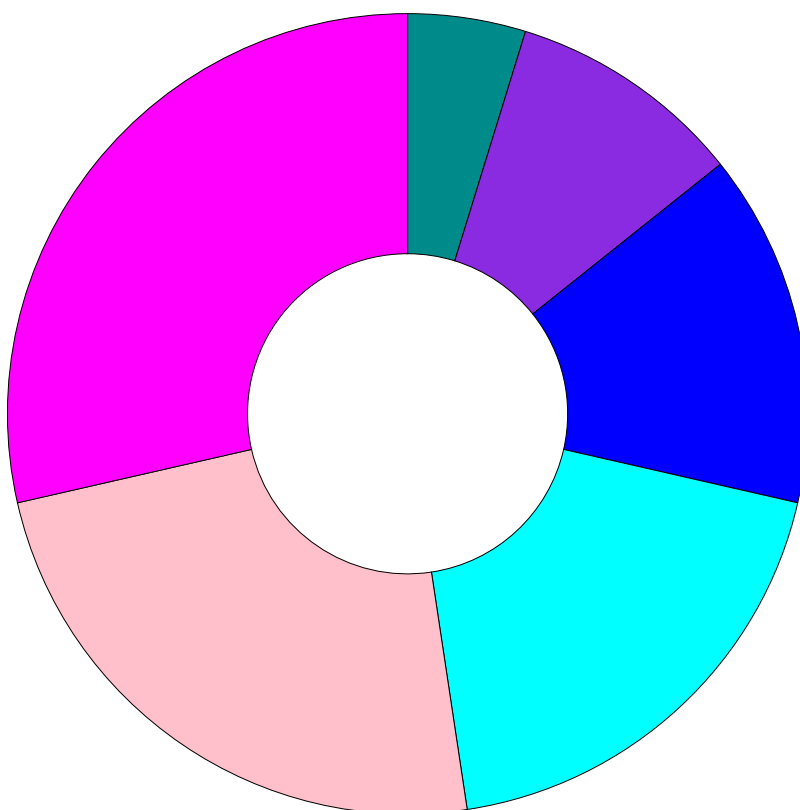


**sample2( ... )**

Make a simple demo

*Example*

```
def sample2():  
    "Make a simple demo"  
  
    d = Drawing(400, 400)  
  
    dn = Doughnut()  
    dn.x = 50  
    dn.y = 50  
    dn.width = 300  
    dn.height = 300  
    dn.data = [10,20,30,40,50,60]  
  
    d.add(dn)  
  
    return d
```



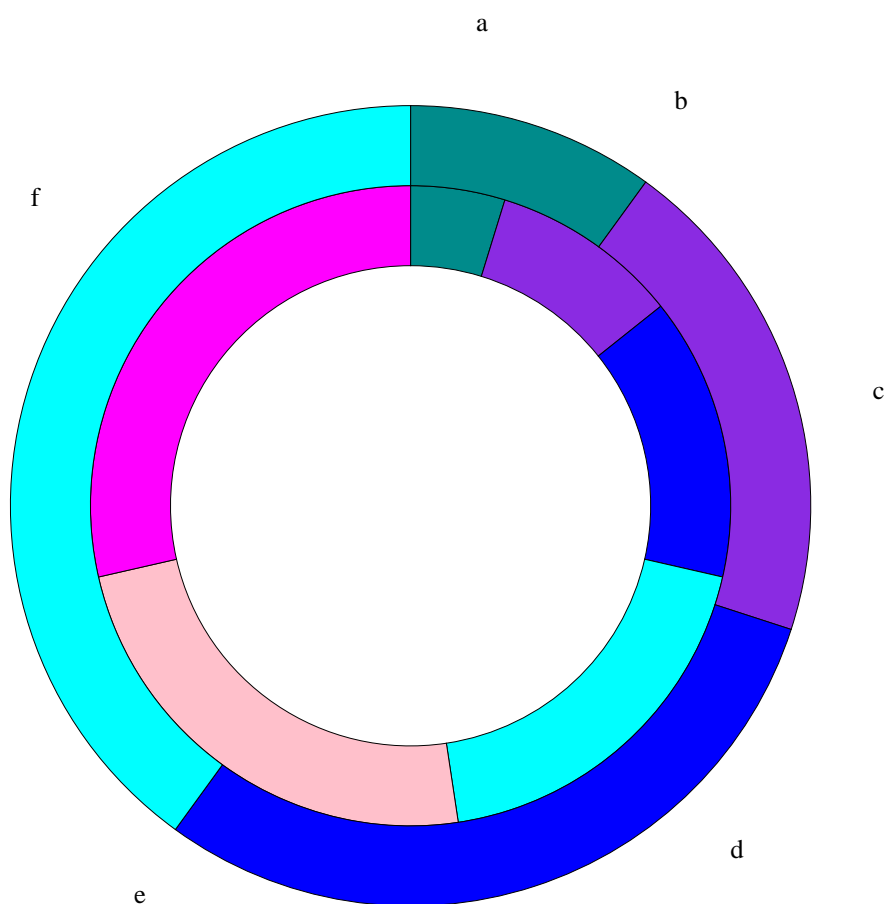


**sample3( ... )**

Make a more complex demo

*Example*

```
def sample3():  
    "Make a more complex demo"  
  
    d = Drawing(400, 400)  
    dn = Doughnut()  
    dn.x = 50  
    dn.y = 50  
    dn.width = 300  
    dn.height = 300  
    dn.data = [[10,20,30,40,50,60], [10,20,30,40]]  
    dn.labels = ['a','b','c','d','e','f']  
  
    d.add(dn)  
  
    return d
```



**sample4( ... )**

Make a more complex demo with Label Overlap fixing

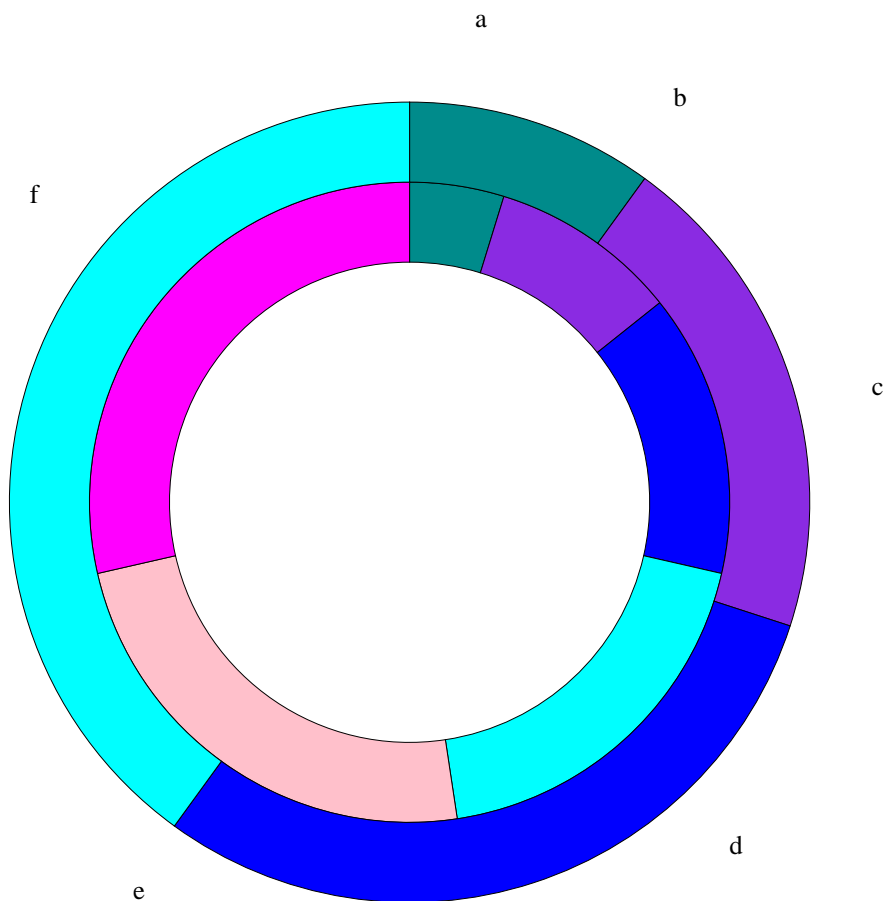
*Example*

```
def sample4():
    "Make a more complex demo with Label Overlap fixing"

    d = Drawing(400, 400)
    dn = Doughnut()
    dn.x = 50
    dn.y = 50
    dn.width = 300
    dn.height = 300
    dn.data = [[10,20,30,40,50,60], [10,20,30,40]]
    dn.labels = ['a','b','c','d','e','f']
    dn.checkLabelOverlap = True

    d.add(dn)

    return d
```



## linechart\_with\_markers

#Autogenerated by ReportLab guiedit do not edit

### Classes

#### LineChartWithMarkers(\_DrawingEditorMixin, Drawing)

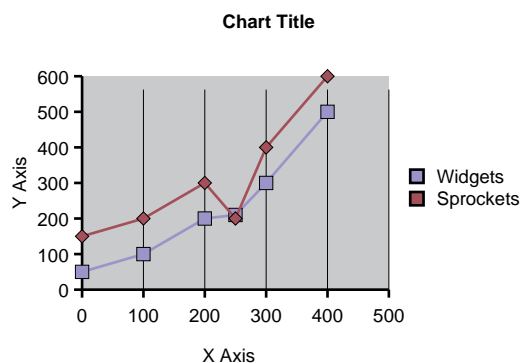
##### Example

```
def __init__(self,width=200,height=150,*args,**kw):
    Drawing.__init__(self,width,height,*args,**kw)
    self._add(self,LinePlot(),name='chart',validate=None,desc="The main chart")
    self.chart.width = 115
    self.chart.height = 80
    self.chart.x = 30
    self.chart.y = 40
    self.chart.lines[0].strokeColor = color01
    self.chart.lines[1].strokeColor = color02
    self.chart.lines[2].strokeColor = color03
    self.chart.lines[3].strokeColor = color04
    self.chart.lines[4].strokeColor = color05
    self.chart.lines[5].strokeColor = color06
    self.chart.lines[6].strokeColor = color07
    self.chart.lines[7].strokeColor = color08
    self.chart.lines[8].strokeColor = color09
    self.chart.lines[9].strokeColor = color10
    self.chart.lines[0].symbol = makeMarker('FilledSquare')
    self.chart.lines[1].symbol = makeMarker('FilledDiamond')
    self.chart.lines[2].symbol = makeMarker('FilledStarFive')
    self.chart.lines[3].symbol = makeMarker('FilledTriangle')
    self.chart.lines[4].symbol = makeMarker('FilledCircle')
    self.chart.lines[5].symbol = makeMarker('FilledPentagon')
    self.chart.lines[6].symbol = makeMarker('FilledStarSix')
    self.chart.lines[7].symbol = makeMarker('FilledHeptagon')
    self.chart.lines[8].symbol = makeMarker('FilledOctagon')
    self.chart.lines[9].symbol = makeMarker('FilledCross')
    self.chart.fillColor = backgroundGrey
    self.chart.lineLabels.fontName = 'Helvetica'
    self.chart.xValueAxis.labels.fontName = 'Helvetica'
    self.chart.xValueAxis.labels.fontSize = 7
    self.chart.xValueAxis.forceZero = 0
    self.chart.data = [(0, 50), (100,100), (200,200), (250,210), (300,300), (400,500)]
    self.chart.xValueAxis.avoidBoundFrac = 1
    self.chart.xValueAxis.gridEnd = 115
    self.chart.xValueAxis.tickDown = 3
    self.chart.xValueAxis.visibleGrid = 1
    self.chart.yValueAxis.tickLeft = 3
    self.chart.yValueAxis.labels.fontName = 'Helvetica'
    self.chart.yValueAxis.labels.fontSize = 7
    self._add(self,Label(),name='Title',validate=None,desc="The title at the top of the chart")
    self.Title.fontName = 'Helvetica-Bold'
    self.Title.fontSize = 7
    self.Title.x = 100
    self.Title.y = 135
    self.Title._text = 'Chart Title'
    self.Title.maxWidth = 180
    self.Title.height = 20
    self.Title.textAnchor = 'middle'
    self._add(self,Legend(),name='Legend',validate=None,desc="The legend or key for the chart")
    self.Legend.colorNamePairs = [(color01, 'Widgets'), (color02, 'Sprockets')]
    self.Legend.fontName = 'Helvetica'
    self.Legend.fontSize = 7
    self.Legend.x = 153
    self.Legend.y = 85
    self.Legend.dxTextSpace = 5
    self.Legend.dy = 5
    self.Legend.dx = 5
    self.Legend.deltay = 5
    self.Legend.alignment = 'right'
    self._add(self,Label(),name='XLabel',validate=None,desc="The label on the horizontal axis")
```

```

self.XLabel.fontName      = 'Helvetica'
self.XLabel.fontSize      = 7
self.XLabel.x             = 85
self.XLabel.y             = 10
self.XLabel.textAnchor    = 'middle'
self.XLabel.maxWidth      = 100
self.XLabel.height        = 20
self.XLabel._text         = "X Axis"
self._add(self,Label(),name='YLabel',validate=None,desc="The label on the vertical axis")
self.YLabel.fontName      = 'Helvetica'
self.YLabel.fontSize      = 7
self.YLabel.x             = 12
self.YLabel.y             = 80
self.YLabel.angle         = 90
self.YLabel.textAnchor    = 'middle'
self.YLabel.maxWidth      = 100
self.YLabel.height        = 20
self.YLabel._text         = "Y Axis"
self.chart.yValueAxis.forceZero = 1
self.chart.xValueAxis.forceZero = 1
self._add(self,0,name='preview',validate=None,desc=None)

```



## line\_chart

#Autogenerated by ReportLab guiedit do not edit

## Classes

### LineChart(\_DrawingEditorMixin, Drawing)

#### Example

```

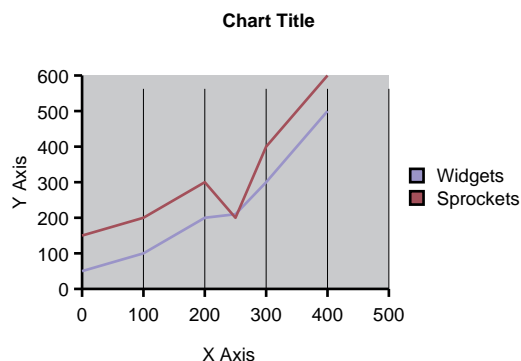
def __init__(self,width=200,height=150,*args,**kw):
    Drawing.__init__(self,width,height,*args,**kw)
    self._add(self,LinePlot(),name='chart',validate=None,desc="The main chart")
    self.chart.width      = 115
    self.chart.height     = 80
    self.chart.x          = 30
    self.chart.y          = 40
    self.chart.lines[0].strokeColor = color01
    self.chart.lines[1].strokeColor = color02
    self.chart.lines[2].strokeColor = color03
    self.chart.lines[3].strokeColor = color04
    self.chart.lines[4].strokeColor = color05
    self.chart.lines[5].strokeColor = color06
    self.chart.lines[6].strokeColor = color07
    self.chart.lines[7].strokeColor = color08
    self.chart.lines[8].strokeColor = color09
    self.chart.lines[9].strokeColor = color10
    self.chart.fillColor    = backgroundGrey
    self.chart.lineLabels.fontName = 'Helvetica'

```

```

self.chart.xValueAxis.labels.fontName      = 'Helvetica'
self.chart.xValueAxis.labels.fontSize      = 7
self.chart.xValueAxis.forceZero            = 0
self.chart.data                            = [(0, 50), (100,100), (200,200), (250,210), (300,300), (400,500)]
self.chart.xValueAxis.avoidBoundFrac       = 1
self.chart.xValueAxis.gridEnd              = 115
self.chart.xValueAxis.tickDown             = 3
self.chart.xValueAxis.visibleGrid         = 1
self.chart.yValueAxis.tickLeft             = 3
self.chart.yValueAxis.labels.fontName      = 'Helvetica'
self.chart.yValueAxis.labels.fontSize      = 7
self._add(self,Label(),name='Title',validate=None,desc="The title at the top of the chart")
self.Title.fontName                        = 'Helvetica-Bold'
self.Title.fontSize                        = 7
self.Title.x                              = 100
self.Title.y                              = 135
self.Title._text                           = 'Chart Title'
self.Title.maxWidth                        = 180
self.Title.height                         = 20
self.Title.textAnchor                      = 'middle'
self._add(self,Legend(),name='Legend',validate=None,desc="The legend or key for the chart")
self.Legend.colorNamePairs                = [(color01, 'Widgets'), (color02, 'Sprockets')]
self.Legend.fontName                      = 'Helvetica'
self.Legend.fontSize                      = 7
self.Legend.x                             = 153
self.Legend.y                             = 85
self.Legend.dxTextSpace                   = 5
self.Legend.dy                           = 5
self.Legend.dx                           = 5
self.Legend.deltay                       = 5
self.Legend.alignment                     = 'right'
self._add(self,Label(),name='XLabel',validate=None,desc="The label on the horizontal axis")
self.XLabel.fontName                      = 'Helvetica'
self.XLabel.fontSize                      = 7
self.XLabel.x                             = 85
self.XLabel.y                             = 10
self.XLabel.textAnchor                    = 'middle'
self.XLabel.maxWidth                      = 100
self.XLabel.height                       = 20
self.XLabel._text                        = "X Axis"
self._add(self,Label(),name='YLabel',validate=None,desc="The label on the vertical axis")
self.YLabel.fontName                      = 'Helvetica'
self.YLabel.fontSize                      = 7
self.YLabel.x                             = 12
self.YLabel.y                             = 80
self.YLabel.angle                         = 90
self.YLabel.textAnchor                    = 'middle'
self.YLabel.maxWidth                      = 100
self.YLabel.height                       = 20
self.YLabel._text                        = "Y Axis"
self.chart.yValueAxis.forceZero           = 1
self.chart.xValueAxis.forceZero           = 1
self._add(self,0,name='preview',validate=None,desc=None)

```



## exploded\_pie

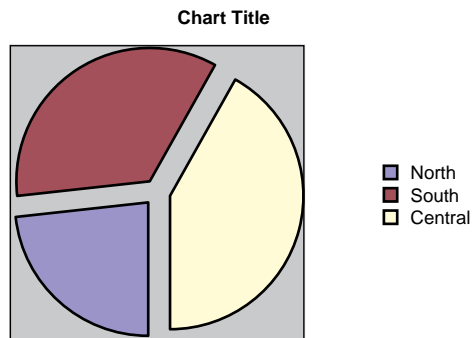
#Autogenerated by ReportLab guiedit do not edit

## Classes

### ExplodedPie(\_DrawingEditorMixin, Drawing)

#### Example

```
def __init__(self,width=200,height=150,*args,**kw):
    Drawing.__init__(self,width,height,*args,**kw)
    self._add(self,Pie(),name='chart',validate=None,desc="The main chart")
    self.chart.width = 100
    self.chart.height = 100
    self.chart.x = 25
    self.chart.y = 25
    self.chart.slices[0].fillColor = color01
    self.chart.slices[1].fillColor = color02
    self.chart.slices[2].fillColor = color03
    self.chart.slices[3].fillColor = color04
    self.chart.slices[4].fillColor = color05
    self.chart.slices[5].fillColor = color06
    self.chart.slices[6].fillColor = color07
    self.chart.slices[7].fillColor = color08
    self.chart.slices[8].fillColor = color09
    self.chart.slices[9].fillColor = color10
    self.chart.data = (100, 150, 180)
    self.chart.startAngle = -90
    self._add(self,Label(),name='Title',validate=None,desc="The title at the top of the chart")
    self.Title.fontName = 'Helvetica-Bold'
    self.Title.fontSize = 7
    self.Title.x = 100
    self.Title.y = 135
    self.Title._text = 'Chart Title'
    self.Title.maxWidth = 180
    self.Title.height = 20
    self.Title.textAnchor = 'middle'
    self._add(self,Legend(),name='Legend',validate=None,desc="The legend or key for the chart")
    self.Legend.colorNamePairs = [(color01, 'North'), (color02, 'South'), (color03, 'Central')]
    self.Legend.fontName = 'Helvetica'
    self.Legend.fontSize = 7
    self.Legend.x = 160
    self.Legend.y = 85
    self.Legend.dxTextSpace = 5
    self.Legend.dy = 5
    self.Legend.dx = 5
    self.Legend.deltay = 5
    self.Legend.alignment = 'right'
    self.Legend.columnMaximum = 10
    self.chart.slices.strokeWidth = 1
    self.chart.slices.fontName = 'Helvetica'
    self.background = ShadedRect()
    self.background.fillColorStart = backgroundGrey
    self.background.fillColorEnd = backgroundGrey
    self.background.numShades = 1
    self.background.strokeWidth = 0.5
    self.background.x = 20
    self.background.y = 20
    self.chart.slices.popout = 5
    self.background.height = 110
    self.background.width = 110
    self._add(self,0,name='preview',validate=None,desc=None)
```



## radar

#Autogenerated by ReportLab guiedit do not edit

## Classes

### RadarChart(\_DrawingEditorMixin, Drawing)

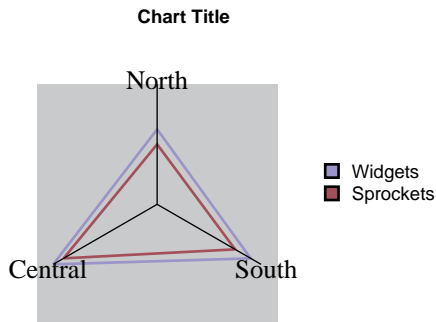
#### Example

```
def __init__(self,width=200,height=150,*args,**kw):
    Drawing.__init__(self,width,height,*args,**kw)
    self._add(self,SpiderChart(),name='chart',validate=None,desc="The main chart")
    self.chart.width = 90
    self.chart.height = 90
    self.chart.x = 45
    self.chart.y = 25
    self.chart.strands[0].strokeColor= color01
    self.chart.strands[1].strokeColor= color02
    self.chart.strands[2].strokeColor= color03
    self.chart.strands[3].strokeColor= color04
    self.chart.strands[4].strokeColor= color05
    self.chart.strands[5].strokeColor= color06
    self.chart.strands[6].strokeColor= color07
    self.chart.strands[7].strokeColor= color08
    self.chart.strands[8].strokeColor= color09
    self.chart.strands[9].strokeColor= color10
    self.chart.strands[0].fillColor = None
    self.chart.strands[1].fillColor = None
    self.chart.strands[2].fillColor = None
    self.chart.strands[3].fillColor = None
    self.chart.strands[4].fillColor = None
    self.chart.strands[5].fillColor = None
    self.chart.strands[6].fillColor = None
    self.chart.strands[7].fillColor = None
    self.chart.strands[8].fillColor = None
    self.chart.strands[9].fillColor = None
    self.chart.strands.strokeWidth = 1
    self.chart.strandLabels.fontName = 'Helvetica'
    self.chart.strandLabels.fontSize = 6
    self.chart.fillColor = backgroundGrey
    self.chart.data = [(125, 180, 200), (100, 150, 180)]
    self.chart.labels = ['North', 'South', 'Central']
    self._add(self,Label(),name='Title',validate=None,desc="The title at the top of the chart")
    self.Title.fontName = 'Helvetica-Bold'
    self.Title.fontSize = 7
    self.Title.x = 100
    self.Title.y = 135
    self.Title._text = 'Chart Title'
    self.Title.maxWidth = 180
    self.Title.height = 20
    self.Title.textAnchor = 'middle'
    self._add(self,Legend(),name='Legend',validate=None,desc="The legend or key for the chart")
```

```

self.Legend.colorNamePairs = [(color01, 'Widgets'), (color02, 'Sprockets')]
self.Legend.fontName       = 'Helvetica'
self.Legend.fontSize       = 7
self.Legend.x               = 153
self.Legend.y               = 85
self.Legend.dxTextSpace    = 5
self.Legend.dy              = 5
self.Legend.dx              = 5
self.Legend.deltay         = 5
self.Legend.alignment       = 'right'
self.chart.strands.strokeWidth = 1
self._add(self,0,name='preview',validate=None,desc=None)

```



## stacked\_column

#Autogenerated by ReportLab guiedit do not edit

## Classes

### StackedColumn(\_DrawingEditorMixin, Drawing)

#### Example

```

def __init__(self,width=200,height=150,*args,**kw):
    Drawing.__init__(self,width,height,*args,**kw)
    self._add(self,VerticalBarChart(),name='chart',validate=None,desc="The main chart")
    self.chart.width      = 115
    self.chart.height     = 80
    self.chart.x           = 30
    self.chart.y           = 40
    self.chart.bars[0].fillColor = color01
    self.chart.bars[1].fillColor = color02
    self.chart.bars[2].fillColor = color03
    self.chart.bars[3].fillColor = color04
    self.chart.bars[4].fillColor = color05
    self.chart.bars[5].fillColor = color06
    self.chart.bars[6].fillColor = color07
    self.chart.bars[7].fillColor = color08
    self.chart.bars[8].fillColor = color09
    self.chart.bars[9].fillColor = color10
    self.chart.fillColor    = backgroundGrey
    self.chart.barLabels.fontName = 'Helvetica'
    self.chart.valueAxis.labels.fontName = 'Helvetica'
    self.chart.valueAxis.labels.fontSize = 7
    self.chart.valueAxis.forceZero = 1
    self.chart.data          = [(100, 150, 180), (125, 180, 200)]
    self.chart.groupSpacing = 15
    self.chart.valueAxis.avoidBoundFrac = 1
    self.chart.valueAxis.gridEnd = 115
    self.chart.valueAxis.tickLeft = 3
    self.chart.valueAxis.visibleGrid = 1

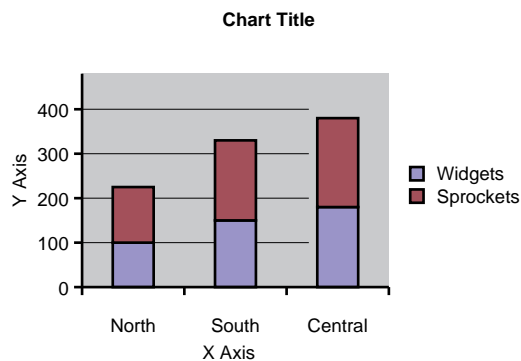
```



```

self.chart.categoryAxis.categoryNames = ['North', 'South', 'Central']
self.chart.categoryAxis.tickDown = 3
self.chart.categoryAxis.labels.fontName = 'Helvetica'
self.chart.categoryAxis.labels.fontSize = 7
self._add(self,Label(),name='Title',validate=None,desc="The title at the top of the chart")
self.Title.fontName = 'Helvetica-Bold'
self.Title.fontSize = 7
self.Title.x = 100
self.Title.y = 135
self.Title._text = 'Chart Title'
self.Title.maxWidth = 180
self.Title.height = 20
self.Title.textAnchor = 'middle'
self._add(self,Legend(),name='Legend',validate=None,desc="The legend or key for the chart")
self.Legend.colorNamePairs = [(color01, 'Widgets'), (color02, 'Sprockets')]
self.Legend.fontName = 'Helvetica'
self.Legend.fontSize = 7
self.Legend.x = 153
self.Legend.y = 85
self.Legend.dxTextSpace = 5
self.Legend.dy = 5
self.Legend.dx = 5
self.Legend.deltay = 5
self.Legend.alignment = 'right'
self._add(self,Label(),name='XLabel',validate=None,desc="The label on the horizontal axis")
self.XLabel.fontName = 'Helvetica'
self.XLabel.fontSize = 7
self.XLabel.x = 85
self.XLabel.y = 10
self.XLabel.textAnchor = 'middle'
self.XLabel.maxWidth = 100
self.XLabel.height = 20
self.XLabel._text = "X Axis"
self._add(self,Label(),name='YLabel',validate=None,desc="The label on the vertical axis")
self.YLabel.fontName = 'Helvetica'
self.YLabel.fontSize = 7
self.YLabel.x = 12
self.YLabel.y = 80
self.YLabel.angle = 90
self.YLabel.textAnchor = 'middle'
self.YLabel.maxWidth = 100
self.YLabel.height = 20
self.YLabel._text = "Y Axis"
self.chart.categoryAxis.style='stacked'
self._add(self,0,name='preview',validate=None,desc=None)

```



## bubble

#Autogenerated by ReportLab guiedit do not edit

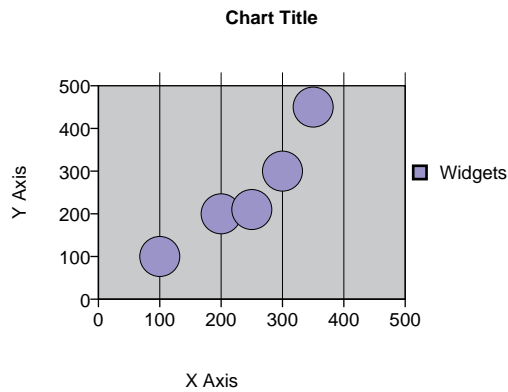
## Classes

**Bubble(\_DrawingEditorMixin, Drawing)**

*Example*

```
def __init__(self,width=200,height=150,*args,**kw):
    Drawing.__init__(self,width,height,*args,**kw)
    self._add(self,ScatterPlot(),name='chart',validate=None,desc="The main chart")
    self.chart.width = 115
    self.chart.height = 80
    self.chart.x = 30
    self.chart.y = 40
    self.chart.lines[0].strokeColor = color01
    self.chart.lines[1].strokeColor = color02
    self.chart.lines[2].strokeColor = color03
    self.chart.lines[3].strokeColor = color04
    self.chart.lines[4].strokeColor = color05
    self.chart.lines[5].strokeColor = color06
    self.chart.lines[6].strokeColor = color07
    self.chart.lines[7].strokeColor = color08
    self.chart.lines[8].strokeColor = color09
    self.chart.lines[9].strokeColor = color10
    self.chart.lines.symbol.kind = 'Circle'
    self.chart.lines.symbol.size = 15
    self.chart.fillColor = backgroundGrey
    self.chart.lineLabels.fontName = 'Helvetica'
    self.chart.xValueAxis.labels.fontName = 'Helvetica'
    self.chart.xValueAxis.labels.fontSize = 7
    self.chart.xValueAxis.forceZero = 0
    self.chart.data = [((100,100), (200,200), (250,210), (300,300), (350,450))]
    self.chart.xValueAxis.avoidBoundFrac = 1
    self.chart.xValueAxis.gridEnd = 115
    self.chart.xValueAxis.tickDown = 3
    self.chart.xValueAxis.visibleGrid = 1
    self.chart.yValueAxis.tickLeft = 3
    self.chart.yValueAxis.labels.fontName = 'Helvetica'
    self.chart.yValueAxis.labels.fontSize = 7
    self._add(self,Label(),name='Title',validate=None,desc="The title at the top of the chart")
    self.Title.fontName = 'Helvetica-Bold'
    self.Title.fontSize = 7
    self.Title.x = 100
    self.Title.y = 135
    self.Title._text = 'Chart Title'
    self.Title.maxWidth = 180
    self.Title.height = 20
    self.Title.textAnchor = 'middle'
    self._add(self,Legend(),name='Legend',validate=None,desc="The legend or key for the chart")
    self.Legend.colorNamePairs = [(color01, 'Widgets')]
    self.Legend.fontName = 'Helvetica'
    self.Legend.fontSize = 7
    self.Legend.x = 153
    self.Legend.y = 85
    self.Legend.dxTextSpace = 5
    self.Legend.dy = 5
    self.Legend.dx = 5
    self.Legend.deltay = 5
    self.Legend.alignment = 'right'
    self.chart.lineLabelFormat = None
    self.chart.xLabel = 'X Axis'
    self.chart.y = 30
    self.chart.yLabel = 'Y Axis'
    self.chart.yValueAxis.labelTextFormat = '%d'
    self.chart.yValueAxis.forceZero = 1
    self.chart.xValueAxis.forceZero = 1

    self._add(self,0,name='preview',validate=None,desc=None)
```



## scatter\_lines\_markers

#Autogenerated by ReportLab guiedit do not edit

## Classes

### ScatterLinesMarkers(\_DrawingEditorMixin, Drawing)

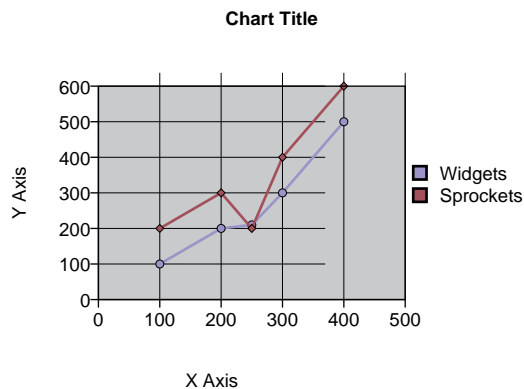
#### Example

```
def __init__(self,width=200,height=150,*args,**kw):
    Drawing.__init__(self,width,height,*args,**kw)
    self._add(self,ScatterPlot(),name='chart',validate=None,desc="The main chart")
    self.chart.width = 115
    self.chart.height = 80
    self.chart.x = 30
    self.chart.y = 40
    self.chart.lines[0].strokeColor = color01
    self.chart.lines[1].strokeColor = color02
    self.chart.lines[2].strokeColor = color03
    self.chart.lines[3].strokeColor = color04
    self.chart.lines[4].strokeColor = color05
    self.chart.lines[5].strokeColor = color06
    self.chart.lines[6].strokeColor = color07
    self.chart.lines[7].strokeColor = color08
    self.chart.lines[8].strokeColor = color09
    self.chart.lines[9].strokeColor = color10
    self.chart.fillColor = backgroundGrey
    self.chart.lineLabels.fontName = 'Helvetica'
    self.chart.xValueAxis.labels.fontName = 'Helvetica'
    self.chart.xValueAxis.labels.fontSize = 7
    self.chart.xValueAxis.forceZero = 0
    self.chart.data = [((100,100), (200,200), (250,210), (300,300), (400,500)), ((100,100), (200,200), (250,210), (300,300), (400,500))]
    self.chart.xValueAxis.avoidBoundFrac = 1
    self.chart.xValueAxis.gridEnd = 115
    self.chart.xValueAxis.tickDown = 3
    self.chart.xValueAxis.visibleGrid = 1
    self.chart.yValueAxis.tickLeft = 3
    self.chart.yValueAxis.labels.fontName = 'Helvetica'
    self.chart.yValueAxis.labels.fontSize = 7
    self._add(self,Label(),name='Title',validate=None,desc="The title at the top of the chart")
    self.Title.fontName = 'Helvetica-Bold'
    self.Title.fontSize = 7
    self.Title.x = 100
    self.Title.y = 135
    self.Title._text = 'Chart Title'
    self.Title.maxWidth = 180
    self.Title.height = 20
    self.Title.textAnchor = 'middle'
    self._add(self,Legend(),name='Legend',validate=None,desc="The legend or key for the chart")
    self.Legend.colorNamePairs = [(color01, 'Widgets'), (color02, 'Sprockets')]
    self.Legend.fontName = 'Helvetica'
    self.Legend.fontSize = 7
```

```

self.Legend.x          = 153
self.Legend.y          = 85
self.Legend.dxTextSpace = 5
self.Legend.dy         = 5
self.Legend.dx         = 5
self.Legend.deltay     = 5
self.Legend.alignment  = 'right'
self.chart.lineLabelFormat = None
self.chart.xLabel      = 'X Axis'
self.chart.y           = 30
self.chart.yLabel      = 'Y Axis'
self.chart.yValueAxis.gridEnd = 115
self.chart.yValueAxis.visibleGrid = 1
self.chart.yValueAxis.labelTextFormat = '%d'
self.chart.yValueAxis.forceZero = 1
self.chart.xValueAxis.forceZero = 1
self.chart.joinedLines = 1
self._add(self,0,name='preview',validate=None,desc=None)

```



## scatter\_lines

#Autogenerated by ReportLab guiedit do not edit

## Classes

### ScatterLines(\_DrawingEditorMixin, Drawing)

#### Example

```

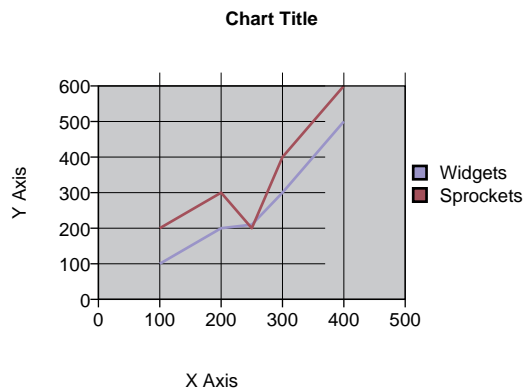
def __init__(self,width=200,height=150,*args,**kw):
    Drawing.__init__(self,width,height,*args,**kw)
    self._add(self,ScatterPlot(),name='chart',validate=None,desc="The main chart")
    self.chart.width = 115
    self.chart.height = 80
    self.chart.x = 30
    self.chart.y = 40
    self.chart.lines[0].strokeColor = color01
    self.chart.lines[1].strokeColor = color02
    self.chart.lines[2].strokeColor = color03
    self.chart.lines[3].strokeColor = color04
    self.chart.lines[4].strokeColor = color05
    self.chart.lines[5].strokeColor = color06
    self.chart.lines[6].strokeColor = color07
    self.chart.lines[7].strokeColor = color08
    self.chart.lines[8].strokeColor = color09
    self.chart.lines[9].strokeColor = color10
    self.chart.lines[0].symbol = None
    self.chart.lines[1].symbol = None
    self.chart.lines[2].symbol = None
    self.chart.lines[3].symbol = None
    self.chart.lines[4].symbol = None

```

```

self.chart.lines[5].symbol = None
self.chart.lines[6].symbol = None
self.chart.lines[7].symbol = None
self.chart.lines[8].symbol = None
self.chart.lines[9].symbol = None
self.chart.fillColor = backgroundGrey
self.chart.lineLabels.fontName = 'Helvetica'
self.chart.xValueAxis.labels.fontName = 'Helvetica'
self.chart.xValueAxis.labels.fontSize = 7
self.chart.xValueAxis.forceZero = 0
self.chart.data = [((100,100), (200,200), (250,210), (300,300), (400,500)), ((100,200), (200,100), (250,110), (300,200), (400,300))]
self.chart.xValueAxis.avoidBoundFrac = 1
self.chart.xValueAxis.gridEnd = 115
self.chart.xValueAxis.tickDown = 3
self.chart.xValueAxis.visibleGrid = 1
self.chart.yValueAxis.tickLeft = 3
self.chart.yValueAxis.labels.fontName = 'Helvetica'
self.chart.yValueAxis.labels.fontSize = 7
self._add(self,Label(),name='Title',validate=None,desc="The title at the top of the chart")
self.Title.fontName = 'Helvetica-Bold'
self.Title.fontSize = 7
self.Title.x = 100
self.Title.y = 135
self.Title._text = 'Chart Title'
self.Title.maxWidth = 180
self.Title.height = 20
self.Title.textAnchor = 'middle'
self._add(self,Legend(),name='Legend',validate=None,desc="The legend or key for the chart")
self.Legend.colorNamePairs = [(color01, 'Widgets'), (color02, 'Sprockets')]
self.Legend.fontName = 'Helvetica'
self.Legend.fontSize = 7
self.Legend.x = 153
self.Legend.y = 85
self.Legend.dxTextSpace = 5
self.Legend.dy = 5
self.Legend.dx = 5
self.Legend.deltay = 5
self.Legend.alignment = 'right'
self.chart.lineLabelFormat = None
self.chart.xLabel = 'X Axis'
self.chart.y = 30
self.chart.yLabel = 'Y Axis'
self.chart.yValueAxis.gridEnd = 115
self.chart.yValueAxis.visibleGrid = 1
self.chart.yValueAxis.labelTextFormat = '%d'
self.chart.yValueAxis.forceZero = 1
self.chart.xValueAxis.forceZero = 1
self.chart.joinedLines = 1
self._add(self,0,name='preview',validate=None,desc=None)

```



## stacked\_bar

#Autogenerated by ReportLab guiedit do not edit

## Classes

### StackedBar(\_DrawingEditorMixin, Drawing)

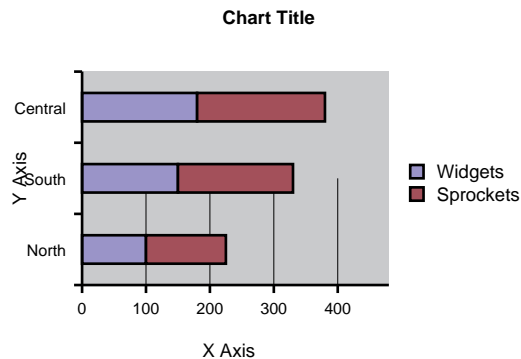
#### Example

```
def __init__(self,width=200,height=150,*args,**kw):
    Drawing.__init__(self,width,height,*args,**kw)
    self._add(self,HorizontalBarChart(),name='chart',validate=None,desc="The main chart")
    self.chart.width = 115
    self.chart.height = 80
    self.chart.x = 30
    self.chart.y = 40
    self.chart.bars[0].fillColor = color01
    self.chart.bars[1].fillColor = color02
    self.chart.bars[2].fillColor = color03
    self.chart.bars[3].fillColor = color04
    self.chart.bars[4].fillColor = color05
    self.chart.bars[5].fillColor = color06
    self.chart.bars[6].fillColor = color07
    self.chart.bars[7].fillColor = color08
    self.chart.bars[8].fillColor = color09
    self.chart.bars[9].fillColor = color10
    self.chart.fillColor = backgroundGrey
    self.chart.barLabels.fontName = 'Helvetica'
    self.chart.valueAxis.labels.fontName = 'Helvetica'
    self.chart.valueAxis.labels.fontSize = 6
    self.chart.valueAxis.forceZero = 1
    self.chart.data = [(100, 150, 180), (125, 180, 200)]
    self.chart.groupSpacing = 15
    self.chart.valueAxis.avoidBoundFrac = 1
    self.chart.valueAxis.gridEnd = 80
    self.chart.valueAxis.tickDown = 3
    self.chart.valueAxis.visibleGrid = 1
    self.chart.categoryAxis.categoryNames = ['North', 'South', 'Central']
    self.chart.categoryAxis.tickLeft = 3
    self.chart.categoryAxis.labels.fontName = 'Helvetica'
    self.chart.categoryAxis.labels.fontSize = 6
    self.chart.categoryAxis.labels.dx = -3
    self._add(self,Label(),name='Title',validate=None,desc="The title at the top of the chart")
    self.Title.fontName = 'Helvetica-Bold'
    self.Title.fontSize = 7
    self.Title.x = 100
    self.Title.y = 135
    self.Title._text = 'Chart Title'
    self.Title.maxWidth = 180
    self.Title.height = 20
    self.Title.textAnchor = 'middle'
    self._add(self,Legend(),name='Legend',validate=None,desc="The legend or key for the chart")
    self.Legend.colorNamePairs = [(color01, 'Widgets'), (color02, 'Sprockets')]
    self.Legend.fontName = 'Helvetica'
    self.Legend.fontSize = 7
    self.Legend.x = 153
    self.Legend.y = 85
    self.Legend.dxTextSpace = 5
    self.Legend.dy = 5
    self.Legend.dx = 5
    self.Legend.deltay = 5
    self.Legend.alignment = 'right'
    self._add(self,Label(),name='XLabel',validate=None,desc="The label on the horizontal axis")
    self.XLabel.fontName = 'Helvetica'
    self.XLabel.fontSize = 7
    self.XLabel.x = 85
    self.XLabel.y = 10
    self.XLabel.textAnchor = 'middle'
    self.XLabel.maxWidth = 100
    self.XLabel.height = 20
    self.XLabel._text = "X Axis"
    self._add(self,Label(),name='YLabel',validate=None,desc="The label on the vertical axis")
    self.YLabel.fontName = 'Helvetica'
    self.YLabel.fontSize = 7
    self.YLabel.x = 12
    self.YLabel.y = 80
    self.YLabel.angle = 90
    self.YLabel.textAnchor = 'middle'
```

```

self.YLabel.maxWidth      = 100
self.YLabel.height        = 20
self.YLabel._text         = "Y Axis"
self.chart.categoryAxis.style='stacked'
self._add(self,0,name='preview',validate=None,desc=None)

```



## clustered\_bar

#Autogenerated by ReportLab guiedit do not edit

## Classes

### ClusteredBar(\_DrawingEditorMixin, Drawing)

#### Example

```

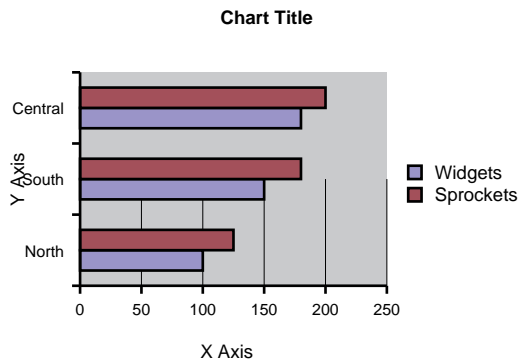
def __init__(self,width=200,height=150,*args,**kw):
    Drawing.__init__(self,width,height,*args,**kw)
    self._add(self,HorizontalBarChart(),name='chart',validate=None,desc="The main chart")
    self.chart.width      = 115
    self.chart.height     = 80
    self.chart.x          = 30
    self.chart.y          = 40
    self.chart.bars[0].fillColor = color01
    self.chart.bars[1].fillColor = color02
    self.chart.bars[2].fillColor = color03
    self.chart.bars[3].fillColor = color04
    self.chart.bars[4].fillColor = color05
    self.chart.bars[5].fillColor = color06
    self.chart.bars[6].fillColor = color07
    self.chart.bars[7].fillColor = color08
    self.chart.bars[8].fillColor = color09
    self.chart.bars[9].fillColor = color10
    self.chart.fillColor    = backgroundGrey
    self.chart.barLabels.fontName      = 'Helvetica'
    self.chart.valueAxis.labels.fontName = 'Helvetica'
    self.chart.valueAxis.labels.fontSize = 6
    self.chart.valueAxis.forceZero     = 1
    self.chart.data                    = [(100, 150, 180), (125, 180, 200)]
    self.chart.groupSpacing            = 15
    self.chart.valueAxis.avoidBoundFrac = 1
    self.chart.valueAxis.gridEnd       = 80
    self.chart.valueAxis.tickDown      = 3
    self.chart.valueAxis.visibleGrid   = 1
    self.chart.categoryAxis.categoryNames = ['North', 'South', 'Central']
    self.chart.categoryAxis.tickLeft    = 3
    self.chart.categoryAxis.labels.fontName = 'Helvetica'
    self.chart.categoryAxis.labels.fontSize = 6
    self.chart.categoryAxis.labels.dx    = -3
    self._add(self,Label(),name='Title',validate=None,desc="The title at the top of the chart")
    self.Title.fontName = 'Helvetica-Bold'

```

```

self.Title.fontSize      = 7
self.Title.x             = 100
self.Title.y             = 135
self.Title._text         = 'Chart Title'
self.Title.maxWidth      = 180
self.Title.height        = 20
self.Title.textAnchor    = 'middle'
self._add(self,Legend(),name='Legend',validate=None,desc="The legend or key for the chart")
self.Legend.colorNamePairs = [(color01, 'Widgets'), (color02, 'Sprockets')]
self.Legend.fontName     = 'Helvetica'
self.Legend.fontSize     = 7
self.Legend.x            = 153
self.Legend.y            = 85
self.Legend.dxTextSpace = 5
self.Legend.dy           = 5
self.Legend.dx           = 5
self.Legend.deltay       = 5
self.Legend.alignment    = 'right'
self._add(self,Label(),name='XLabel',validate=None,desc="The label on the horizontal axis")
self.XLabel.fontName     = 'Helvetica'
self.XLabel.fontSize     = 7
self.XLabel.x            = 85
self.XLabel.y            = 10
self.XLabel.textAnchor   = 'middle'
self.XLabel.maxWidth     = 100
self.XLabel.height       = 20
self.XLabel._text       = "X Axis"
self._add(self,Label(),name='YLabel',validate=None,desc="The label on the vertical axis")
self.YLabel.fontName     = 'Helvetica'
self.YLabel.fontSize     = 7
self.YLabel.x            = 12
self.YLabel.y            = 80
self.YLabel.angle        = 90
self.YLabel.textAnchor   = 'middle'
self.YLabel.maxWidth     = 100
self.YLabel.height       = 20
self.YLabel._text       = "Y Axis"
self._add(self,0,name='preview',validate=None,desc=None)

```



## simple\_pie

#Autogenerated by ReportLab guiedit do not edit

## Classes

### SimplePie(\_DrawingEditorMixin, Drawing)

#### Example

```

def __init__(self,width=200,height=150,*args,**kw):
    Drawing.__init__(self,width,height,*args,**kw)

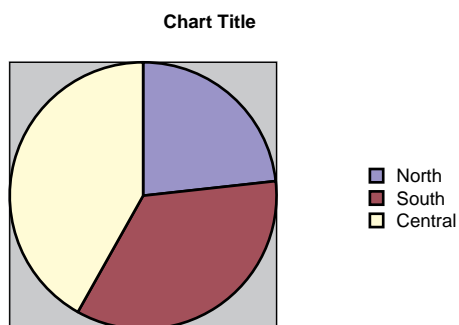
```



```

self._add(self,Pie(),name='chart',validate=None,desc="The main chart")
self.chart.width = 100
self.chart.height = 100
self.chart.x = 25
self.chart.y = 25
self.chart.slices[0].fillColor = color01
self.chart.slices[1].fillColor = color02
self.chart.slices[2].fillColor = color03
self.chart.slices[3].fillColor = color04
self.chart.slices[4].fillColor = color05
self.chart.slices[5].fillColor = color06
self.chart.slices[6].fillColor = color07
self.chart.slices[7].fillColor = color08
self.chart.slices[8].fillColor = color09
self.chart.slices[9].fillColor = color10
self.chart.data = (100, 150, 180)
self._add(self,Label(),name='Title',validate=None,desc="The title at the top of the chart")
self.Title.fontName = 'Helvetica-Bold'
self.Title.fontSize = 7
self.Title.x = 100
self.Title.y = 135
self.Title._text = 'Chart Title'
self.Title.maxWidth = 180
self.Title.height = 20
self.Title.textAnchor = 'middle'
self._add(self,Legend(),name='Legend',validate=None,desc="The legend or key for the chart")
self.Legend.colorNamePairs = [(color01, 'North'), (color02, 'South'),(color03, 'Central')]
self.Legend.fontName = 'Helvetica'
self.Legend.fontSize = 7
self.Legend.x = 160
self.Legend.y = 85
self.Legend.dxTextSpace = 5
self.Legend.dy = 5
self.Legend.dx = 5
self.Legend.deltay = 5
self.Legend.alignment = 'right'
self.chart.slices.strokeWidth = 1
self.chart.slices.fontName = 'Helvetica'
self.background = ShadedRect()
self.background.fillColorStart = backgroundGrey
self.background.fillColorEnd = backgroundGrey
self.background.numShades = 1
self.background.strokeWidth = 0.5
self.background.x = 25
self.background.y = 25
self.Legend.columnMaximum = 10
self._add(self,0,name='preview',validate=None,desc=None)

```



## filled\_radar

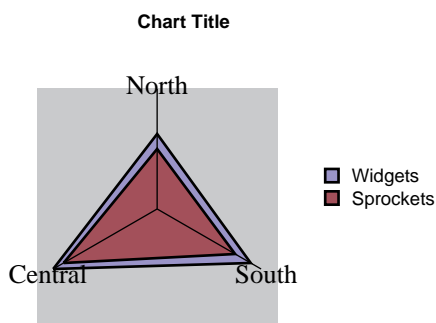
#Autogenerated by ReportLab guiedit do not edit

## Classes

## FilledRadarChart(\_DrawingEditorMixin, Drawing)

### Example

```
def __init__(self,width=200,height=150,*args,**kw):
    Drawing.__init__(self,width,height,*args,**kw)
    self._add(self,SpiderChart(),name='chart',validate=None,desc="The main chart")
    self.chart.width = 90
    self.chart.height = 90
    self.chart.x = 45
    self.chart.y = 25
    self.chart.strands[0].fillColor = color01
    self.chart.strands[1].fillColor = color02
    self.chart.strands[2].fillColor = color03
    self.chart.strands[3].fillColor = color04
    self.chart.strands[4].fillColor = color05
    self.chart.strands[5].fillColor = color06
    self.chart.strands[6].fillColor = color07
    self.chart.strands[7].fillColor = color08
    self.chart.strands[8].fillColor = color09
    self.chart.strands[9].fillColor = color10
    self.chart.strandLabels.fontName = 'Helvetica'
    self.chart.strandLabels.fontSize = 6
    self.chart.fillColor = backgroundGrey
    self.chart.data = [(125, 180, 200), (100, 150, 180)]
    self.chart.labels = ['North', 'South', 'Central']
    self._add(self,Label(),name='Title',validate=None,desc="The title at the top of the chart")
    self.Title.fontName = 'Helvetica-Bold'
    self.Title.fontSize = 7
    self.Title.x = 100
    self.Title.y = 135
    self.Title._text = 'Chart Title'
    self.Title.maxWidth = 180
    self.Title.height = 20
    self.Title.textAnchor = 'middle'
    self._add(self,Legend(),name='Legend',validate=None,desc="The legend or key for the chart")
    self.Legend.colorNamePairs = [(color01, 'Widgets'), (color02, 'Sprockets')]
    self.Legend.fontName = 'Helvetica'
    self.Legend.fontSize = 7
    self.Legend.x = 153
    self.Legend.y = 85
    self.Legend.dxTextSpace = 5
    self.Legend.dy = 5
    self.Legend.dx = 5
    self.Legend.deltay = 5
    self.Legend.alignment = 'right'
    self._add(self,0,name='preview',validate=None,desc=None)
```



### scatter

#Autogenerated by ReportLab guiedit do not edit

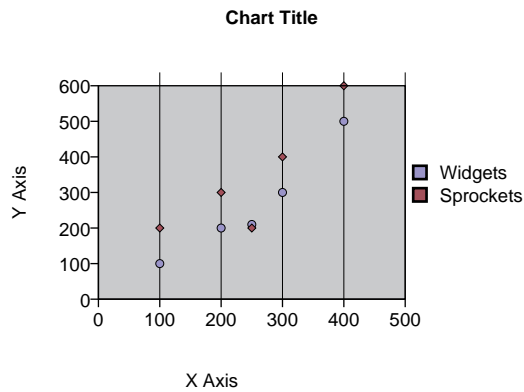
## Classes

### Scatter(\_DrawingEditorMixin, Drawing)

#### Example

```
def __init__(self,width=200,height=150,*args,**kw):
    Drawing.__init__(self,width,height,*args,**kw)
    self._add(self,ScatterPlot(),name='chart',validate=None,desc="The main chart")
    self.chart.width = 115
    self.chart.height = 80
    self.chart.x = 30
    self.chart.y = 40
    self.chart.lines[0].strokeColor = color01
    self.chart.lines[1].strokeColor = color02
    self.chart.lines[2].strokeColor = color03
    self.chart.lines[3].strokeColor = color04
    self.chart.lines[4].strokeColor = color05
    self.chart.lines[5].strokeColor = color06
    self.chart.lines[6].strokeColor = color07
    self.chart.lines[7].strokeColor = color08
    self.chart.lines[8].strokeColor = color09
    self.chart.lines[9].strokeColor = color10
    self.chart.fillColor = backgroundGrey
    self.chart.lineLabels.fontName = 'Helvetica'
    self.chart.xValueAxis.labels.fontName = 'Helvetica'
    self.chart.xValueAxis.labels.fontSize = 7
    self.chart.xValueAxis.forceZero = 0
    self.chart.data = [((100,100), (200,200), (250,210), (300,300), (400,500)), ((100,
    self.chart.xValueAxis.avoidBoundFrac = 1
    self.chart.xValueAxis.gridEnd = 115
    self.chart.xValueAxis.tickDown = 3
    self.chart.xValueAxis.visibleGrid = 1
    self.chart.yValueAxis.tickLeft = 3
    self.chart.yValueAxis.labels.fontName = 'Helvetica'
    self.chart.yValueAxis.labels.fontSize = 7
    self._add(self,Label(),name='Title',validate=None,desc="The title at the top of the chart")
    self.Title.fontName = 'Helvetica-Bold'
    self.Title.fontSize = 7
    self.Title.x = 100
    self.Title.y = 135
    self.Title._text = 'Chart Title'
    self.Title.maxWidth = 180
    self.Title.height = 20
    self.Title.textAnchor = 'middle'
    self._add(self,Legend(),name='Legend',validate=None,desc="The legend or key for the chart")
    self.Legend.colorNamePairs = [(color01, 'Widgets'), (color02, 'Sprockets')]
    self.Legend.fontName = 'Helvetica'
    self.Legend.fontSize = 7
    self.Legend.x = 153
    self.Legend.y = 85
    self.Legend.dxTextSpace = 5
    self.Legend.dy = 5
    self.Legend.dx = 5
    self.Legend.deltay = 5
    self.Legend.alignment = 'right'
    self.chart.lineLabelFormat = None
    self.chart.xLabel = 'X Axis'
    self.chart.y = 30
    self.chart.yLabel = 'Y Axis'
    self.chart.yValueAxis.labelTextFormat = '%d'
    self.chart.yValueAxis.forceZero = 1
    self.chart.xValueAxis.forceZero = 1

    self._add(self,0,name='preview',validate=None,desc=None)
```



## clustered\_column

#Autogenerated by ReportLab guiedit do not edit

## Classes

### ClusteredColumn(\_DrawingEditorMixin, Drawing)

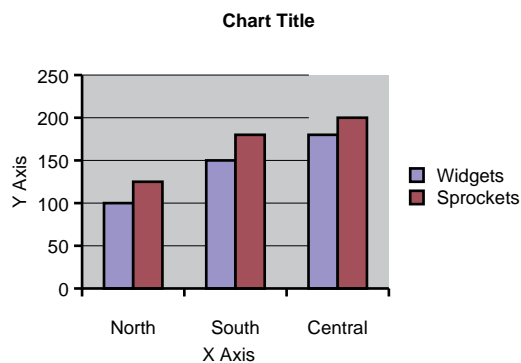
#### Example

```
def __init__(self,width=200,height=150,*args,**kw):
    Drawing.__init__(self,width,height,*args,**kw)
    self._add(self,VerticalBarChart(),name='chart',validate=None,desc="The main chart")
    self.chart.width = 115
    self.chart.height = 80
    self.chart.x = 30
    self.chart.y = 40
    self.chart.bars[0].fillColor = color01
    self.chart.bars[1].fillColor = color02
    self.chart.bars[2].fillColor = color03
    self.chart.bars[3].fillColor = color04
    self.chart.bars[4].fillColor = color05
    self.chart.bars[5].fillColor = color06
    self.chart.bars[6].fillColor = color07
    self.chart.bars[7].fillColor = color08
    self.chart.bars[8].fillColor = color09
    self.chart.bars[9].fillColor = color10
    self.chart.fillColor = backgroundGrey
    self.chart.barLabels.fontName = 'Helvetica'
    self.chart.valueAxis.labels.fontName = 'Helvetica'
    self.chart.valueAxis.labels.fontSize = 7
    self.chart.valueAxis.forceZero = 1
    self.chart.data = [(100, 150, 180), (125, 180, 200)]
    self.chart.groupSpacing = 15
    self.chart.valueAxis.avoidBoundFrac = 1
    self.chart.valueAxis.gridEnd = 115
    self.chart.valueAxis.tickLeft = 3
    self.chart.valueAxis.visibleGrid = 1
    self.chart.categoryAxis.categoryNames = ['North', 'South', 'Central']
    self.chart.categoryAxis.tickDown = 3
    self.chart.categoryAxis.labels.fontName = 'Helvetica'
    self.chart.categoryAxis.labels.fontSize = 7
    self._add(self,Label(),name='Title',validate=None,desc="The title at the top of the chart")
    self.Title.fontName = 'Helvetica-Bold'
    self.Title.fontSize = 7
    self.Title.x = 100
    self.Title.y = 135
    self.Title._text = 'Chart Title'
    self.Title.maxWidth = 180
    self.Title.height = 20
    self.Title.textAnchor = 'middle'
    self._add(self,Legend(),name='Legend',validate=None,desc="The legend or key for the chart")
    self.Legend.colorNamePairs = [(color01, 'Widgets'), (color02, 'Sprockets')]
```

```

self.Legend.fontName      = 'Helvetica'
self.Legend.fontSize      = 7
self.Legend.x             = 153
self.Legend.y             = 85
self.Legend.dxTextSpace  = 5
self.Legend.dy           = 5
self.Legend.dx            = 5
self.Legend.deltay       = 5
self.Legend.alignment     = 'right'
self._add(self,Label(),name='XLabel',validate=None,desc="The label on the horizontal axis")
self.XLabel.fontName      = 'Helvetica'
self.XLabel.fontSize      = 7
self.XLabel.x             = 85
self.XLabel.y             = 10
self.XLabel.textAnchor    = 'middle'
self.XLabel.maxWidth      = 100
self.XLabel.height        = 20
self.XLabel._text         = "X Axis"
self._add(self,Label(),name='YLabel',validate=None,desc="The label on the vertical axis")
self.YLabel.fontName      = 'Helvetica'
self.YLabel.fontSize      = 7
self.YLabel.x             = 12
self.YLabel.y             = 80
self.YLabel.angle         = 90
self.YLabel.textAnchor    = 'middle'
self.YLabel.maxWidth      = 100
self.YLabel.height        = 20
self.YLabel._text         = "Y Axis"
self._add(self,0,name='preview',validate=None,desc=None)

```



## table

#Copyright ReportLab Europe Ltd. 2000-2012

#see license.txt for license details

#history <http://www.reportlab.co.uk/cgi-bin/viewcvs.cgi/public/reportlab/trunk/reportlab/graphics/widgets/grids.py>

## Classes

### TableWidget (Widget)

A two dimensions table of labels

### Public Attributes

**alignment** Alignment of text within cells

**borderStrokeColor** table border color

**borderStrokeWidth** border line width

**boxAnchor** location of the table anchoring point

**data** a list of list of strings to be displayed in the cells

**dividerDashArray** Dash array for dividerLines.

**fillColor** table fill color

**fontColor** font color

**fontName** text font in the table

**fontSize** font size of the table

**height** table height

**horizontalDividerStrokeColor** table inner horizontal lines color

**horizontalDividerStrokeWidth** table inner horizontal lines width

**textAnchor** Alignment of text within cells

**verticalDividerStrokeColor** table inner vertical lines color

**verticalDividerStrokeWidth** table inner vertical lines width

**width** table width

**x** x position of left edge of table

**y** y position of bottom edge of table

#### *Example*

```
def demo(self):
    """ returns a sample of this widget with data
    """
    d = Drawing(400, 200)
    t = TableWidget()
    d.add(t, name='table')
    d.table.dividerDashArray = (1, 3, 2)
    d.table.verticalDividerStrokeColor = None
    d.table.borderStrokeWidth = 0
    d.table.borderStrokeColor = colors.red
    return d
```

#### *Properties of Example Widget*

```
alignment = 'right'
borderStrokeColor = Color(0,0,0,1)
borderStrokeWidth = 0.5
boxAnchor = 'nw'
data = [['North', 'South', 'East', 'West'],
        [100, 110, 120, 130],
        ['A', 'B', 'C', 'D']]
dividerDashArray = None
fillColor = None
fontColor = Color(0,0,0,1)
fontSize = 8
height = 100
horizontalDividerStrokeColor = Color(0,0,0,1)
horizontalDividerStrokeWidth = 0.5
textAnchor = 'start'
verticalDividerStrokeColor = Color(0,0,0,1)
verticalDividerStrokeWidth = 0.25
width = 200
x = 10
y = 10
```

## grids

#Copyright ReportLab Europe Ltd. 2000-2012

#see license.txt for license details

#history <http://www.reportlab.co.uk/cgi-bin/viewcvs.cgi/public/reportlab/trunk/reportlab/graphics/widgets/grids.py>

## Classes

### DoubleGrid(Widget)

This combines two ordinary Grid objects orthogonal to each other.

### Public Attributes

**grid0** The first grid component.

**grid1** The second grid component.

**height** The grid's height.

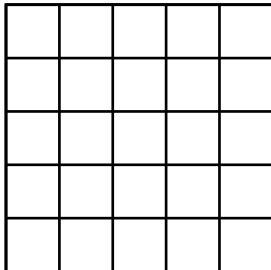
**width** The grid's width.

**x** The grid's lower-left x position.

**y** The grid's lower-left y position.

### Example

```
def demo(self):
    D = Drawing(100, 100)
    g = DoubleGrid()
    D.add(g)
    return D
```



### Properties of Example Widget

```
grid0.delta = 20
grid0.delta0 = 0
grid0.deltaSteps = []
grid0.fillColor = Color(1,1,1,1)
grid0.height = 100
grid0.orientation = 'vertical'
grid0.stripeColors = [Color(1,0,0,1), Color(0,.501961,0,1), Color(0,0,1,1)]
grid0.strokeColor = Color(0,0,0,1)
grid0.strokeWidth = 1
grid0.useLines = 1
grid0.useRects = 0
grid0.width = 100
grid0.x = 0
grid0.y = 0
grid1.delta = 20
grid1.delta0 = 0
grid1.deltaSteps = []
grid1.fillColor = Color(1,1,1,1)
grid1.height = 100
```

```
grid1.orientation = 'horizontal'
grid1.stripeColors = [Color(1,0,0,1), Color(0,.501961,0,1), Color(0,0,1,1)]
grid1.strokeColor = Color(0,0,0,1)
grid1.strokeWidth = 1
grid1.useLines = 1
grid1.useRects = 0
grid1.width = 100
grid1.x = 0
grid1.y = 0
height = 100
width = 100
x = 0
y = 0
```



## Grid(Widget)

This makes a rectangular grid of equidistant stripes.

The grid contains an outer border rectangle, and stripes inside which can be drawn with lines and/or as solid tiles. The drawing order is: outer rectangle, then lines and tiles.

The stripes' width is indicated as 'delta'. The sequence of stripes can have an offset named 'delta0'. Both values need to be positive!

## Public Attributes

**delta** Determines the width/height of the stripes.

**delta0** Determines the stripes initial width/height offset.

**deltaSteps** List of deltas to be used cyclically.

**fillColor** Background color for entire rectangle.

**height** The grid's height.

**orientation** Determines if stripes are vertical or horizontal.

**rectStrokeColor** Color for outer rect stroke.

**rectStrokeWidth** Width for outer rect stroke.

**stripeColors** Colors applied cyclically in the right or upper direction.

**strokeColor** Color used for lines.

**strokeWidth** Width used for lines.

**useLines** Determines if stripes are drawn with lines.

**useRects** Determines if stripes are drawn with solid rectangles.

**width** The grid's width.

**x** The grid's lower-left x position.

**y** The grid's lower-left y position.

### Example

```
def demo(self):
    D = Drawing(100, 100)

    g = Grid()
    D.add(g)

    return D
```

### Properties of Example Widget

```
delta = 20
delta0 = 0
deltaSteps = []
fillColor = Color(1,1,1,1)
height = 100
orientation = 'vertical'
stripeColors = [Color(1,0,0,1), Color(0,.501961,0,1), Color(0,0,1,1)]
strokeColor = Color(0,0,0,1)
strokeWidth = 2
useLines = 0
```

```
useRects = 1  
width = 100  
x = 0  
y = 0
```

## ShadedPolygon(Widget, LineShape)

### Public Attributes

**angle** Shading angle

**cylinderMode** True if shading reverses in middle.

**fillColorEnd** None

**fillColorStart** None

**numShades** The number of interpolating colors.

**overprintMask** overprinting for ordinary CMYK

**points** None

**strokeColor** None

**strokeDashArray** a sequence of numbers represents on and off, e.g. (2,1)

**strokeLineCap** Line cap 0=butt, 1=round 2=square

**strokeLineJoin** Line join 0=miter, 1=round 2=bevel

**strokeMiterLimit** miter limit control miter line joins

**strokeOpacity** The level of transparency of the line, any real number between 0 and 1

**strokeOverprint** Turn on stroke overprinting

**strokeWidth** None

### Example

```
def demo(self):
    msg = "demo() must be implemented for each Widget!"
    raise shapes.NotImplementedError, msg
```

### Properties of Example Widget

```
angle = 90
cylinderMode = 0
fillColorEnd = Color(0,.501961,0,1)
fillColorStart = Color(1,0,0,1)
numShades = 50
points = [-1, -1, 2, 2, 3, -1]
strokeColor = Color(0,0,0,1)
strokeDashArray = None
strokeLineCap = 0
strokeLineJoin = 0
strokeMiterLimit = 0
strokeOpacity = None
strokeWidth = 1
```

## ShadedRect(Widget)

This makes a rectangle with shaded colors between two colors.

Colors are interpolated linearly between 'fillColorStart' and 'fillColorEnd', both of which appear at the margins. If 'numShades' is set to one, though, only 'fillColorStart' is used.

### Public Attributes

**cylinderMode** True if shading reverses in middle.

**fillColorEnd** End value of the color shade.

**fillColorStart** Start value of the color shade.

**height** The grid's height.

**numShades** The number of interpolating colors.

**orientation** Determines if stripes are vertical or horizontal.

**strokeColor** Color used for border line.

**strokeWidth** Width used for lines.

**width** The grid's width.

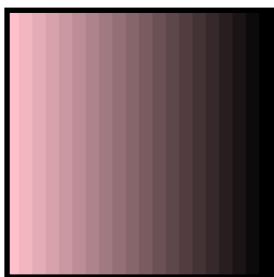
**x** The grid's lower-left x position.

**y** The grid's lower-left y position.

### Example

```
def demo(self):
    D = Drawing(100, 100)
    g = ShadedRect()
    D.add(g)

    return D
```



### Properties of Example Widget

```
cylinderMode = 0
fillColorEnd = Color(0,0,0,1)
fillColorStart = Color(1,.752941,.796078,1)
height = 100
numShades = 20
orientation = 'vertical'
strokeColor = Color(0,0,0,1)
strokeWidth = 2
width = 100
x = 0
y = 0
```

## flags

This file is a collection of flag graphics as widgets.

All flags are represented at the ratio of 1:2, even where the official ratio for the flag is something else (such as 3:5 for the German national flag). The only exceptions are for where this would look *\_very\_* wrong, such as the Danish flag whose (ratio is 28:37), or the Swiss flag (which is square).

Unless otherwise stated, these flags are all the 'national flags' of the countries, rather than their state flags, naval flags, ensigns or any other variants. (National flags are the flag flown by civilians of a country and the ones usually used to represent a country abroad. State flags are the variants used by the government and by diplomatic missions overseas).

To check on how close these are to the 'official' representations of flags, check the World Flag Database at <http://www.flags.ndirect.co.uk/>

The flags this file contains are:

EU Members:

United Kingdom, Austria, Belgium, Denmark, Finland, France, Germany, Greece, Ireland, Italy, Luxembourg, Holland (The Netherlands), Spain, Sweden

Others:

USA, Czech Republic, European Union, Switzerland, Turkey, Brazil

(Brazilian flag contributed by Publio da Costa Melo [publio@planetarium.com.br]).

## Classes

### Flag(\_Symbol)

This is a generic flag class that all the flags in this file use as a basis.

This class basically provides edges and a tidy-up routine to hide any bits of line that overlap the 'outside' of the flag

possible attributes:

'x', 'y', 'size', 'fillColor'

### Public Attributes

**border** Whether a background is drawn

**dx** symbol x coordinate adjustment

**dy** symbol x coordinate adjustment

**fillColor** Background color

**kind** Which flag

**size** None

**strokeColor** None

**strokeWidth** None

**x** symbol x coordinate

**y** symbol y coordinate

## **Star(\_Symbol)**

This draws a 5-pointed star.

possible attributes:

'x', 'y', 'size', 'fillColor', 'strokeColor'

## **Public Attributes**

**angle** angle in degrees

**dx** symbol x coordinate adjustment

**dy** symbol x coordinate adjustment

**fillColor** None

**size** None

**strokeColor** None

**strokeWidth** None

**x** symbol x coordinate

**y** symbol y coordinate

## **eventcal**

This file is a

## **Classes**

### **EventCalendar(Widget)**

## **Public Attributes**

### *Example*

```
def demo(self):
    msg = "demo() must be implemented for each Widget!"
    raise shapes.NotImplementedError, msg
```

### *Properties of Example Widget*

```
data = []
day = 0
endTime = None
height = 150
startTime = None
timeColWidth = None
trackNames = None
trackRowHeight = 20
width = 300
x = 0
y = 0
```

## **signsandsymbols**

This file is a collection of widgets to produce some common signs and symbols.

Widgets include:

- ETriangle (an equilateral triangle),
- RTriangle (a right angled triangle),
- Octagon,
- Crossbox,
- Tickbox,
- SmileyFace,
- StopSign,
- NoEntry,
- NotAllowed (the red roundel from 'no smoking' signs),
- NoSmoking,
- DangerSign (a black exclamation point in a yellow triangle),
- YesNo (returns a tickbox or a crossbox depending on a testvalue),
- FloppyDisk,
- ArrowOne, and
- ArrowTwo

## **Classes**

### **ArrowOne ( \_Symbol )**

This widget draws an arrow (style one).

possible attributes:  
'x', 'y', 'size', 'fillColor'

### **Public Attributes**

**dx** symbol x coordinate adjustment

**dy** symbol x coordinate adjustment

**fillColor** None

**size** None

**strokeColor** None

**strokeWidth** None

**x** symbol x coordinate

**y** symbol y coordinate

### **ArrowTwo ( ArrowOne )**

This widget draws an arrow (style two).

possible attributes:  
'x', 'y', 'size', 'fillColor'

### **Public Attributes**

**dx** symbol x coordinate adjustment

**dy** symbol x coordinate adjustment

**fillColor** None

**size** None

**strokeColor** None

**strokeWidth** None

**x** symbol x coordinate

**y** symbol y coordinate

### **Crossbox( \_Symbol )**

This draws a black box with a red cross in it - a 'checkbox'.

possible attributes:

'x', 'y', 'size', 'crossColor', 'strokeColor', 'crosswidth'

### **Public Attributes**

**crossColor** None

**crosswidth** None

**dx** symbol x coordinate adjustment

**dy** symbol x coordinate adjustment

**fillColor** None

**size** None

**strokeColor** None

**strokeWidth** None

**x** symbol x coordinate

**y** symbol y coordinate

### **DangerSign( \_Symbol )**

This draws a 'danger' sign: a yellow box with a black exclamation point.

possible attributes:

'x', 'y', 'size', 'strokeColor', 'fillColor', 'strokeWidth'

### **Public Attributes**

**dx** symbol x coordinate adjustment

**dy** symbol x coordinate adjustment

**fillColor** None

**size** None

**strokeColor** None

**strokeWidth** None

**x** symbol x coordinate

**y** symbol y coordinate



**ETriangle(\_Symbol)**

This draws an equilateral triangle.

**Public Attributes**

**dx** symbol x coordinate adjustment

**dy** symbol x coordinate adjustment

**fillColor** None

**size** None

**strokeColor** None

**strokeWidth** None

**x** symbol x coordinate

**y** symbol y coordinate

**FloppyDisk(\_Symbol)**

This widget draws an icon of a floppy disk.

possible attributes:

'x', 'y', 'size', 'diskcolor'

**Public Attributes**

**diskColor** None

**dx** symbol x coordinate adjustment

**dy** symbol x coordinate adjustment

**fillColor** None

**size** None

**strokeColor** None

**strokeWidth** None

**x** symbol x coordinate

**y** symbol y coordinate

**NoEntry(\_Symbol)**

This draws a (British) No Entry sign - a red circle with a white line on it.

possible attributes:

'x', 'y', 'size'

**Public Attributes**

**dx** symbol x coordinate adjustment

**dy** symbol x coordinate adjustment

**fillColor** None

**innerBarColor** color of the inner bar

**size** None

**strokeColor** None

**strokeWidth** None

**x** symbol x coordinate

**y** symbol y coordinate

### **NoSmoking(NotAllowed)**

This draws a no-smoking sign.

possible attributes:

'x', 'y', 'size'

### **Public Attributes**

**dx** symbol x coordinate adjustment

**dy** symbol x coordinate adjustment

**fillColor** None

**size** None

**strokeColor** None

**strokeWidth** None

**x** symbol x coordinate

**y** symbol y coordinate

### **NotAllowed(\_Symbol)**

This draws a 'forbidden' roundel (as used in the no-smoking sign).

possible attributes:

'x', 'y', 'size'

### **Public Attributes**

**dx** symbol x coordinate adjustment

**dy** symbol x coordinate adjustment

**fillColor** None

**size** None

**strokeColor** None

**strokeWidth** None

**x** symbol x coordinate

**y** symbol y coordinate

### **Octagon(\_Symbol)**

This widget draws an Octagon.

possible attributes:

'x', 'y', 'size', 'fillColor', 'strokeColor'

### Public Attributes

**dx** symbol x coordinate adjustment

**dy** symbol x coordinate adjustment

**fillColor** None

**size** None

**strokeColor** None

**strokeWidth** None

**x** symbol x coordinate

**y** symbol y coordinate

### **RTriangle(\_Symbol)**

This draws a right-angled triangle.

possible attributes:

'x', 'y', 'size', 'fillColor', 'strokeColor'

### Public Attributes

**dx** symbol x coordinate adjustment

**dy** symbol x coordinate adjustment

**fillColor** None

**size** None

**strokeColor** None

**strokeWidth** None

**x** symbol x coordinate

**y** symbol y coordinate

### **SmileyFace(\_Symbol)**

This draws a classic smiley face.

possible attributes:

'x', 'y', 'size', 'fillColor'

### Public Attributes

**dx** symbol x coordinate adjustment

**dy** symbol x coordinate adjustment

**fillColor** None

**size** None

**strokeColor** None

**strokeWidth** None

**x** symbol x coordinate

**y** symbol y coordinate

## **StopSign( \_Symbol )**

This draws a (British) stop sign.

possible attributes:

'x', 'y', 'size'

## **Public Attributes**

**dx** symbol x coordinate adjustment

**dy** symbol x coordinate adjustment

**fillColor** None

**size** None

**stopColor** color of the word stop

**strokeColor** None

**strokeWidth** None

**x** symbol x coordinate

**y** symbol y coordinate

## **Tickbox( \_Symbol )**

This draws a black box with a red tick in it - another 'checkbox'.

possible attributes:

'x', 'y', 'size', 'tickColor', 'strokeColor', 'tickwidth'

## **Public Attributes**

**dx** symbol x coordinate adjustment

**dy** symbol x coordinate adjustment

**fillColor** None

**size** None

**strokeColor** None

**strokeWidth** None

**tickColor** None

**tickwidth** None

**x** symbol x coordinate

**y** symbol y coordinate

## **YesNo( \_Symbol )**

This widget draw a tickbox or crossbox depending on 'testValue'.

If this widget is supplied with a 'True' or 1 as a value for testValue, it will use the tickbox widget. Otherwise, it will produce a crossbox.

possible attributes:

'x', 'y', 'size', 'tickcolor', 'crosscolor', 'testValue'

## Public Attributes

**crosscolor** None

**dx** symbol x coordinate adjustment

**dy** symbol x coordinate adjustment

**fillColor** None

**size** None

**strokeColor** None

**strokeWidth** None

**testValue** None

**tickcolor** None

**x** symbol x coordinate

**y** symbol y coordinate

## \_\_Symbol(Widget)

Abstract base widget

possible attributes:

'x', 'y', 'size', 'fillColor', 'strokeColor'

## Public Attributes

**dx** symbol x coordinate adjustment

**dy** symbol x coordinate adjustment

**fillColor** None

**size** None

**strokeColor** None

**strokeWidth** None

**x** symbol x coordinate

**y** symbol y coordinate