

---

# **pygal Documentation**

*Release 2.0.0*

**Florian Mounier**

September 05, 2016



<b>1</b>	<b>Sexy python charting</b>	<b>1</b>
<b>2</b>	<b>Simple python charting</b>	<b>3</b>
<b>3</b>	<b>Index</b>	<b>5</b>
	<b>Python Module Index</b>	<b>105</b>



---

**Sexy python charting**

---



---

## Simple python charting

---

```
pygal.Bar() (1, 3, 3, 7) (1, 6, 6, 4).render()
```





## 3.1 Documentation

### 3.1.1 First steps

**Caution:** First you need to install pygal, see installing.

When it's done, you are ready to make your first chart:

```
import pygal # First import pygal
bar_chart = pygal.Bar() # Then create a bar graph object
bar_chart.add('Fibonacci', [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]) # Add some values
bar_chart.render_to_file('bar_chart.svg') # Save the svg to a file
```

Now you should have a svg file called `bar_chart.svg` in your current directory.

You can open it with various programs such as your web browser, inkscape or any svg compatible viewer.

The resulting chart will be the following:

```
bar_chart = pygal.Bar()
bar_chart.add('Fibonacci', [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55])
bar_chart.render()
```

**Caution:** pygal relies on svg css styling. This is sadly not fully supported by gnome `librsvg` and therefore can lead to black svg being displayed. This is not a bug in pygal. [See this bugzilla search](#)

To make a multiple series graph just add another one:

```
bar_chart = pygal.Bar()
bar_chart.add('Fibonacci', [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55])
bar_chart.add('Padovan', [1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12])
bar_chart.render()
```

If you want to stack them, use `StackedBar` instead of `Bar`:

```
bar_chart = pygal.StackedBar()
bar_chart.add('Fibonacci', [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55])
bar_chart.add('Padovan', [1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12])
bar_chart.render()
```

You can also make it horizontal with `HorizontalStackedBar`:

```
bar_chart = pygal.HorizontalStackedBar()
bar_chart.add('Fibonacci', [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55])
bar_chart.add('Padovan', [1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12])
bar_chart.render()
```

And finally add a title and some labels:

```
bar_chart = pygal.HorizontalStackedBar()
bar_chart.title = "Remarkable sequences"
bar_chart.x_labels = map(str, range(11))
bar_chart.add('Fibonacci', [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55])
bar_chart.add('Padovan', [1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12])
bar_chart.render()
```

The public API is chainable and can be simplified as call arguments, the last chart can be also written:

```
bar_chart = pygal.HorizontalStackedBar(
    title="Remarkable sequences", x_labels=map(str, range(11)))(
    0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, title='Fibonacci')(
    1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12, title='Padovan')
```

## 3.1.2 Chart types

pygal provides various kinds of charts:

### Line

#### Basic

Basic simple line graph:

```
line_chart = pygal.Line()
line_chart.title = 'Browser usage evolution (in %)'
line_chart.x_labels = map(str, range(2002, 2013))
line_chart.add('Firefox', [None, None, 0, 16.6, 25, 31, 36.4, 45.5, 46.3, 42.8, 37.1])
line_chart.add('Chrome', [None, None, None, None, None, None, 0, 3.9, 10.8, 23.8, 35.3])
line_chart.add('IE', [85.8, 84.6, 84.7, 74.5, 66, 58.6, 54.7, 44.8, 36.2, 26.6, 20.1])
line_chart.add('Others', [14.2, 15.4, 15.3, 8.9, 9, 10.4, 8.9, 5.8, 6.7, 6.8, 7.5])
line_chart.render()
```

#### Horizontal Line

Same graph but horizontal and with a range of 0-100.

```
line_chart = pygal.HorizontalLine()
line_chart.title = 'Browser usage evolution (in %)'
line_chart.x_labels = map(str, range(2002, 2013))
line_chart.add('Firefox', [None, None, 0, 16.6, 25, 31, 36.4, 45.5, 46.3, 42.8, 37.1])
line_chart.add('Chrome', [None, None, None, None, None, None, 0, 3.9, 10.8, 23.8, 35.3])
line_chart.add('IE', [85.8, 84.6, 84.7, 74.5, 66, 58.6, 54.7, 44.8, 36.2, 26.6, 20.1])
line_chart.add('Others', [14.2, 15.4, 15.3, 8.9, 9, 10.4, 8.9, 5.8, 6.7, 6.8, 7.5])
line_chart.range = [0, 100]
line_chart.render()
```

## Stacked

Same graph but with stacked values and filled rendering:

```

line_chart = pygal.StackedLine(fill=True)
line_chart.title = 'Browser usage evolution (in %)'
line_chart.x_labels = map(str, range(2002, 2013))
line_chart.add('Firefox', [None, None, 0, 16.6, 25, 31, 36.4, 45.5, 46.3, 42.8, 37.1])
line_chart.add('Chrome', [None, None, None, None, None, None, 0, 3.9, 10.8, 23.8, 35.3])
line_chart.add('IE', [85.8, 84.6, 84.7, 74.5, 66, 58.6, 54.7, 44.8, 36.2, 26.6, 20.1])
line_chart.add('Others', [14.2, 15.4, 15.3, 8.9, 9, 10.4, 8.9, 5.8, 6.7, 6.8, 7.5])
line_chart.render()

```

## Time

For time related plots, just format your labels or use one variant of xy charts:

```

from datetime import datetime, timedelta
date_chart = pygal.Line(x_label_rotation=20)
date_chart.x_labels = map(lambda d: d.strftime('%Y-%m-%d'), [
    datetime(2013, 1, 2),
    datetime(2013, 1, 12),
    datetime(2013, 2, 2),
    datetime(2013, 2, 22)])
date_chart.add("Visits", [300, 412, 823, 672])
date_chart.render()

```

## None values

None values will be skipped. It is also possible to break lines.

## Bar

### Basic

Basic simple bar graph:

```

line_chart = pygal.Bar()
line_chart.title = 'Browser usage evolution (in %)'
line_chart.x_labels = map(str, range(2002, 2013))
line_chart.add('Firefox', [None, None, 0, 16.6, 25, 31, 36.4, 45.5, 46.3, 42.8, 37.1])
line_chart.add('Chrome', [None, None, None, None, None, None, 0, 3.9, 10.8, 23.8, 35.3])
line_chart.add('IE', [85.8, 84.6, 84.7, 74.5, 66, 58.6, 54.7, 44.8, 36.2, 26.6, 20.1])
line_chart.add('Others', [14.2, 15.4, 15.3, 8.9, 9, 10.4, 8.9, 5.8, 6.7, 6.8, 7.5])
line_chart.render()

```

### Stacked

Same graph but with stacked values:

```
line_chart = pygal.StackedBar()
line_chart.title = 'Browser usage evolution (in %)'
line_chart.x_labels = map(str, range(2002, 2013))
line_chart.add('Firefox', [None, None, 0, 16.6, 25, 31, 36.4, 45.5, 46.3, 42.8, 37.1])
line_chart.add('Chrome', [None, None, None, None, None, None, 0, 3.9, 10.8, 23.8, 35.3])
line_chart.add('IE', [85.8, 84.6, 84.7, 74.5, 66, 58.6, 54.7, 44.8, 36.2, 26.6, 20.1])
line_chart.add('Others', [14.2, 15.4, 15.3, 8.9, 9, 10.4, 8.9, 5.8, 6.7, 6.8, 7.5])
line_chart.render()
```

## Horizontal

Horizontal bar diagram:

```
line_chart = pygal.HorizontalBar()
line_chart.title = 'Browser usage in February 2012 (in %)'
line_chart.add('IE', 19.5)
line_chart.add('Firefox', 36.6)
line_chart.add('Chrome', 36.3)
line_chart.add('Safari', 4.5)
line_chart.add('Opera', 2.3)
line_chart.render()
```

## Histogram

### Basic

Histogram are special bars that take 3 values for a bar: the ordinate height, the abscissa start and the abscissa end.

```
hist = pygal.Histogram()
hist.add('Wide bars', [(5, 0, 10), (4, 5, 13), (2, 0, 15)])
hist.add('Narrow bars', [(10, 1, 2), (12, 4, 4.5), (8, 11, 13)])
hist.render()
```

## XY

### Basic

Basic XY lines, drawing cosinus:

```
from math import cos
xy_chart = pygal.XY()
xy_chart.title = 'XY Cosinus'
xy_chart.add('x = cos(y)', [(cos(x / 10.), x / 10.) for x in range(-50, 50, 5)])
xy_chart.add('y = cos(x)', [(x / 10., cos(x / 10.)) for x in range(-50, 50, 5)])
xy_chart.add('x = 1', [(1, -5), (1, 5)])
xy_chart.add('x = -1', [(-1, -5), (-1, 5)])
xy_chart.add('y = 1', [(-5, 1), (5, 1)])
xy_chart.add('y = -1', [(-5, -1), (5, -1)])
xy_chart.render()
```

## Scatter Plot

Disabling stroke make a good scatter plot

```
xy_chart = pygal.XY(stroke=False)
xy_chart.title = 'Correlation'
xy_chart.add('A', [(0, 0), (.1, .2), (.3, .1), (.5, 1), (.8, .6), (1, 1.08), (1.3, 1.1), (2, 3.23),
xy_chart.add('B', [(0.1, .15), (.12, .23), (.4, .3), (.6, .4), (.21, .21), (.5, .3), (.6, .8), (.7, .8),
xy_chart.add('C', [(0.05, .01), (.13, .02), (1.5, 1.7), (1.52, 1.6), (1.8, 1.63), (1.5, 1.82), (1.7, 1.82),
xy_chart.render()
```

## Dates

You can use these helpers to plot date related charts:

### DateTime

```
from datetime import datetime
datetimeline = pygal.DateTimeLine(
    x_label_rotation=35, truncate_label=-1,
    x_value_formatter=lambda dt: dt.strftime('%d, %b %Y at %I:%M:%S %p'))
datetimeline.add("Serie", [
    (datetime(2013, 1, 2, 12, 0), 300),
    (datetime(2013, 1, 12, 14, 30, 45), 412),
    (datetime(2013, 2, 2, 6), 823),
    (datetime(2013, 2, 22, 9, 45), 672)
])
datetimeline.render()
```

**Caution:** datetime are taken in utc by default (ie: no tzinfo). If you have dates with timezones ensure that all your dates have timezone otherwise you will have incoherences.

### Date

```
from datetime import date
dateline = pygal.DateLine(x_label_rotation=25)
dateline.x_labels = [
    date(2013, 1, 1),
    date(2013, 7, 1),
    date(2014, 1, 1),
    date(2014, 7, 1),
    date(2015, 1, 1),
    date(2015, 7, 1)
]
dateline.add("Serie", [
    (date(2013, 1, 2), 213),
    (date(2013, 8, 2), 281),
    (date(2014, 12, 7), 198),
    (date(2015, 3, 21), 120)
])
dateline.render()
```

### Time

```
from datetime import time
dateline = pygal.TimeLine(x_label_rotation=25)
dateline.add("Serie", [
    (time(), 0),
    (time(6), 5),
    (time(8, 30), 12),
    (time(11, 59, 59), 4),
    (time(18), 10),
    (time(23, 30), -1),
])
dateline.render()
```

### TimeDelta

```
from datetime import timedelta
dateline = pygal.TimeDeltaLine(x_label_rotation=25)
dateline.add("Serie", [
    (timedelta(), 0),
    (timedelta(seconds=6), 5),
    (timedelta(minutes=11, seconds=59), 4),
    (timedelta(days=3, microseconds=30), 12),
    (timedelta(weeks=1), 10),
])
dateline.render()
```

### None values

None values will be skipped. It is also possible to break lines.

### Pie

#### Basic

Simple pie:

```
pie_chart = pygal.Pie()
pie_chart.title = 'Browser usage in February 2012 (in %)'
pie_chart.add('IE', 19.5)
pie_chart.add('Firefox', 36.6)
pie_chart.add('Chrome', 36.3)
pie_chart.add('Safari', 4.5)
pie_chart.add('Opera', 2.3)
pie_chart.render()
```

#### Multi-series pie

Same pie but divided in sub category:

```
pie_chart = pygal.Pie()
pie_chart.title = 'Browser usage by version in February 2012 (in %)'
pie_chart.add('IE', [5.7, 10.2, 2.6, 1])
pie_chart.add('Firefox', [.6, 16.8, 7.4, 2.2, 1.2, 1, 1, 1.1, 4.3, 1])
pie_chart.add('Chrome', [.3, .9, 17.1, 15.3, .6, .5, 1.6])
```

```
pie_chart.add('Safari', [4.4, .1])
pie_chart.add('Opera', [.1, 1.6, .1, .5])
pie_chart.render()
```

## Donut

It is possible to specify an inner radius to get a donut:

```
pie_chart = pygal.Pie(inner_radius=.4)
pie_chart.title = 'Browser usage in February 2012 (in %)'
pie_chart.add('IE', 19.5)
pie_chart.add('Firefox', 36.6)
pie_chart.add('Chrome', 36.3)
pie_chart.add('Safari', 4.5)
pie_chart.add('Opera', 2.3)
pie_chart.render()
```

or a ring:

```
pie_chart = pygal.Pie(inner_radius=.75)
pie_chart.title = 'Browser usage in February 2012 (in %)'
pie_chart.add('IE', 19.5)
pie_chart.add('Firefox', 36.6)
pie_chart.add('Chrome', 36.3)
pie_chart.add('Safari', 4.5)
pie_chart.add('Opera', 2.3)
pie_chart.render()
```

## Half pie

```
pie_chart = pygal.Pie(half_pie=True)
pie_chart.title = 'Browser usage in February 2012 (in %)'
pie_chart.add('IE', 19.5)
pie_chart.add('Firefox', 36.6)
pie_chart.add('Chrome', 36.3)
pie_chart.add('Safari', 4.5)
pie_chart.add('Opera', 2.3)
pie_chart.render()
```

## Radar

### Basic

Simple Kiviati diagram:

```
radar_chart = pygal.Radar()
radar_chart.title = 'V8 benchmark results'
radar_chart.x_labels = ['Richards', 'DeltaBlue', 'Crypto', 'RayTrace', 'EarleyBoyer', 'RegExp', 'Spl
```

## Box

### Extremes (default)

By default, the extremes mode is used that is the whiskers are the extremes of the data set, the box goes from the first quartile to the third and the middle line is the median.

```
box_plot = pygal.Box()
box_plot.title = 'V8 benchmark results'
box_plot.add('Chrome', [6395, 8212, 7520, 7218, 12464, 1660, 2123, 8607])
box_plot.add('Firefox', [7473, 8099, 11700, 2651, 6361, 1044, 3797, 9450])
box_plot.add('Opera', [3472, 2933, 4203, 5229, 5810, 1828, 9013, 4669])
box_plot.add('IE', [43, 41, 59, 79, 144, 136, 34, 102])
box_plot.render()
```

### 1.5 interquartile range

Same as above except the whiskers are the first quartile minus 1.5 times the interquartile range and the third quartile plus 1.5 times the interquartile range.

```
box_plot = pygal.Box(box_mode="1.5IQR")
box_plot.title = 'V8 benchmark results'
box_plot.add('Chrome', [6395, 8212, 7520, 7218, 12464, 1660, 2123, 8607])
box_plot.add('Firefox', [7473, 8099, 11700, 2651, 6361, 1044, 3797, 9450])
box_plot.add('Opera', [3472, 2933, 4203, 5229, 5810, 1828, 9013, 4669])
box_plot.add('IE', [43, 41, 59, 79, 144, 136, 34, 102])
box_plot.render()
```

### Tukey

The whiskers are the lowest datum within the 1.5 IQR of the lower quartile and the highest datum still within 1.5 IQR of the upper quartile. The outliers are shown too.

```
box_plot = pygal.Box(box_mode="tukey")
box_plot.title = 'V8 benchmark results'
box_plot.add('Chrome', [6395, 8212, 7520, 7218, 12464, 1660, 2123, 8607])
box_plot.add('Firefox', [7473, 8099, 11700, 2651, 6361, 1044, 3797, 9450])
box_plot.add('Opera', [3472, 2933, 4203, 5229, 5810, 1828, 9013, 4669])
box_plot.add('IE', [43, 41, 59, 79, 144, 136, 34, 102])
box_plot.render()
```

### Standard deviation

The whiskers are defined here by the standard deviation of the data.

```
box_plot = pygal.Box(box_mode="stdev")
box_plot.title = 'V8 benchmark results'
box_plot.add('Chrome', [6395, 8212, 7520, 7218, 12464, 1660, 2123, 8607])
box_plot.add('Firefox', [7473, 8099, 11700, 2651, 6361, 1044, 3797, 9450])
box_plot.add('Opera', [3472, 2933, 4203, 5229, 5810, 1828, 9013, 4669])
box_plot.add('IE', [43, 41, 59, 79, 144, 136, 34, 102])
box_plot.render()
```



## Population standard deviation

The whiskers are defined here by the population standard deviation of the data.

```
box_plot = pygal.Box(box_mode="pstdev")
box_plot.title = 'V8 benchmark results'
box_plot.add('Chrome', [6395, 8212, 7520, 7218, 12464, 1660, 2123, 8607])
box_plot.add('Firefox', [7473, 8099, 11700, 2651, 6361, 1044, 3797, 9450])
box_plot.add('Opera', [3472, 2933, 4203, 5229, 5810, 1828, 9013, 4669])
box_plot.add('IE', [43, 41, 59, 79, 144, 136, 34, 102])
box_plot.render()
```

## Dot

### Basic

Punch card like chart:

```
dot_chart = pygal.Dot(x_label_rotation=30)
dot_chart.title = 'V8 benchmark results'
dot_chart.x_labels = ['Richards', 'DeltaBlue', 'Crypto', 'RayTrace', 'EarleyBoyer', 'RegExp', 'Splay']
dot_chart.add('Chrome', [6395, 8212, 7520, 7218, 12464, 1660, 2123, 8607])
dot_chart.add('Firefox', [7473, 8099, 11700, 2651, 6361, 1044, 3797, 9450])
dot_chart.add('Opera', [3472, 2933, 4203, 5229, 5810, 1828, 9013, 4669])
dot_chart.add('IE', [43, 41, 59, 79, 144, 136, 34, 102])
dot_chart.render()
```

### Negative

Negative values are also supported, drawing the dot empty:

```
dot_chart = pygal.Dot(x_label_rotation=30)
dot_chart.add('Normal', [10, 50, 76, 80, 25])
dot_chart.add('With negatives', [0, -34, -29, 39, -75])
dot_chart.render()
```

## Funnel

### Basic

Funnel chart:

```
funnel_chart = pygal.Funnel()
funnel_chart.title = 'V8 benchmark results'
funnel_chart.x_labels = ['Richards', 'DeltaBlue', 'Crypto', 'RayTrace', 'EarleyBoyer', 'RegExp', 'Splay']
funnel_chart.add('Opera', [3472, 2933, 4203, 5229, 5810, 1828, 9013, 4669])
funnel_chart.add('Firefox', [7473, 8099, 11700, 2651, 6361, 1044, 3797, 9450])
funnel_chart.add('Chrome', [6395, 8212, 7520, 7218, 12464, 1660, 2123, 8607])
funnel_chart.render()
```

## SolidGauge

SolidGauge charts

## Normal

```
gauge = pygal.SolidGauge(inner_radius=0.70)
percent_formatter = lambda x: '{:.10g}%'.format(x)
dollar_formatter = lambda x: '{:.10g}$'.format(x)
gauge.value_formatter = percent_formatter

gauge.add('Series 1', [{'value': 225000, 'max_value': 1275000}],
         formatter=dollar_formatter)
gauge.add('Series 2', [{'value': 110, 'max_value': 100}])
gauge.add('Series 3', [{'value': 3}])
gauge.add(
    'Series 4', [
        {'value': 51, 'max_value': 100},
        {'value': 12, 'max_value': 100}])
gauge.add('Series 5', [{'value': 79, 'max_value': 100}])
gauge.add('Series 6', 99)
gauge.add('Series 7', [{'value': 100, 'max_value': 100}])
gauge.render()
```

## Half

```
gauge = pygal.SolidGauge(
    half_pie=True, inner_radius=0.70,
    style=pygal.style.styles['default'](value_font_size=10))

percent_formatter = lambda x: '{:.10g}%'.format(x)
dollar_formatter = lambda x: '{:.10g}$'.format(x)
gauge.value_formatter = percent_formatter

gauge.add('Series 1', [{'value': 225000, 'max_value': 1275000}],
         formatter=dollar_formatter)
gauge.add('Series 2', [{'value': 110, 'max_value': 100}])
gauge.add('Series 3', [{'value': 3}])
gauge.add(
    'Series 4', [
        {'value': 51, 'max_value': 100},
        {'value': 12, 'max_value': 100}])
gauge.add('Series 5', [{'value': 79, 'max_value': 100}])
gauge.add('Series 6', 99)
gauge.add('Series 7', [{'value': 100, 'max_value': 100}])
gauge.render()
```

## Gauge

### Basic

Gauge chart:

```
gauge_chart = pygal.Gauge(human_readable=True)
gauge_chart.title = 'DeltaBlue V8 benchmark results'
gauge_chart.range = [0, 10000]
gauge_chart.add('Chrome', 8212)
gauge_chart.add('Firefox', 8099)
```

```
gauge_chart.add('Opera', 2933)
gauge_chart.add('IE', 41)
gauge_chart.render()
```

## Pyramid

### Basic

Population pyramid:

```
ages = [(364381, 358443, 360172, 345848, 334895, 326914, 323053, 312576, 302015, 301277, 309874, 318
(346205, 340570, 342668, 328475, 319010, 312898, 308153, 296752, 289639, 290466, 296190, 303871, 3
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 23, 91, 412, 1319, 2984, 5816, 10053, 16045, 242
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 74, 392, 1351, 3906, 7847, 12857, 19913, 29108, 4
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 5, 17, 15, 31, 34, 38, 35, 45, 299, 295, 218,
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 8, 0, 8, 21, 34, 49, 84, 97, 368, 401, 414, 5
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 5, 0, 11, 35, 137, 331, 803, 1580, 2361, 3
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 6, 11, 20, 68, 179, 480, 1077, 2094, 3581, 51

types = ['Males single', 'Females single',
        'Males married', 'Females married',
        'Males widowed', 'Females widowed',
        'Males divorced', 'Females divorced']

pyramid_chart = pygal.Pyramid(human_readable=True, legend_at_bottom=True)
pyramid_chart.title = 'England population by age in 2010 (source: ons.gov.uk)'
pyramid_chart.x_labels = map(lambda x: str(x) if not x % 5 else '', range(90))
for type, age in zip(types, ages):
    pyramid_chart.add(type, age)
pyramid_chart.render()
```

## Treemap

### Basic

Treemap:

```
treemap = pygal.Treemap()
treemap.title = 'Binary TreeMap'
treemap.add('A', [2, 1, 12, 4, 2, 1, 1, 3, 12, 3, 4, None, 9])
treemap.add('B', [4, 2, 5, 10, 3, 4, 2, 7, 4, -10, None, 8, 3, 1])
treemap.add('C', [3, 8, 3, 3, 5, 3, 3, 5, 4, 12])
treemap.add('D', [23, 18])
treemap.add('E', [1, 2, 1, 2, 3, 3, 1, 2, 3,
                 4, 3, 1, 2, 1, 1, 1, 1, 1])
treemap.add('F', [31])
treemap.add('G', [5, 9.3, 8.1, 12, 4, 3, 2])
treemap.add('H', [12, 3, 3])
treemap.render()
```

## Maps

Maps are now packaged separately to keep pygal a reasonable sized package.

There are currently 3 available packages:

### World map

**Installing** The world map plugin can be installed by doing a:

```
pip install pygal_maps_world
```

**Countries** Then you will have access to the `pygal.maps.world` module. Now you can plot countries by specifying their code (see below for the big list of supported country codes)

```
worldmap_chart = pygal.maps.world.World()
worldmap_chart.title = 'Some countries'
worldmap_chart.add('F countries', ['fr', 'fi'])
worldmap_chart.add('M countries', ['ma', 'mc', 'md', 'me', 'mg',
                                   'mk', 'ml', 'mm', 'mn', 'mo',
                                   'mr', 'mt', 'mu', 'mv', 'mw',
                                   'mx', 'my', 'mz'])
worldmap_chart.add('U countries', ['ua', 'ug', 'us', 'uy', 'uz'])
worldmap_chart.render()
```

You can also specify a value for a country:

```
worldmap_chart = pygal.maps.world.World()
worldmap_chart.title = 'Minimum deaths by capital punishment (source: Amnesty International)'
worldmap_chart.add('In 2012', {
    'af': 14,
    'bd': 1,
    'by': 3,
    'cn': 1000,
    'gm': 9,
    'in': 1,
    'ir': 314,
    'iq': 129,
    'jp': 7,
    'kp': 6,
    'pk': 1,
    'ps': 6,
    'sa': 79,
    'so': 6,
    'sd': 5,
    'tw': 6,
    'ae': 1,
    'us': 43,
    'ye': 28
})
worldmap_chart.render()
```

**Continents** You have also access to continents:

```
supra = pygal.maps.world.SupranationalWorld()
supra.add('Asia', [('asia', 1)])
supra.add('Europe', [('europe', 1)])
supra.add('Africa', [('africa', 1)])
supra.add('North america', [('north_america', 1)])
supra.add('South america', [('south_america', 1)])
```

```

supra.add('Oceania', [('oceania', 1)])
supra.add('Antartica', [('antartica', 1)])
supra.render()

```

**Country code list** The following countries are supported:

code	Country
ad	Andorra
ae	United Arab Emirates
af	Afghanistan
al	Albania
am	Armenia
ao	Angola
aq	Antarctica
ar	Argentina
at	Austria
au	Australia
az	Azerbaijan
ba	Bosnia and Herzegovina
bd	Bangladesh
be	Belgium
bf	Burkina Faso
bg	Bulgaria
bh	Bahrain
bi	Burundi
bj	Benin
bn	Brunei Darussalam
bo	Bolivia, Plurinational State of
br	Brazil
bt	Bhutan
bw	Botswana
by	Belarus
bz	Belize
ca	Canada
cd	Congo, the Democratic Republic of the
cf	Central African Republic
cg	Congo
ch	Switzerland
ci	Cote d'Ivoire
cl	Chile
cm	Cameroon
cn	China
co	Colombia
cr	Costa Rica
cu	Cuba
cv	Cape Verde
cy	Cyprus
cz	Czech Republic
de	Germany
dj	Djibouti

Continued on next page

Table 3.1 – continued from previous page

code	Country
dk	Denmark
do	Dominican Republic
dz	Algeria
ec	Ecuador
ee	Estonia
eg	Egypt
eh	Western Sahara
er	Eritrea
es	Spain
et	Ethiopia
fi	Finland
fr	France
ga	Gabon
gb	United Kingdom
ge	Georgia
gf	French Guiana
gh	Ghana
gl	Greenland
gm	Gambia
gn	Guinea
gq	Equatorial Guinea
gr	Greece
gt	Guatemala
gu	Guam
gw	Guinea-Bissau
gy	Guyana
hk	Hong Kong
hn	Honduras
hr	Croatia
ht	Haiti
hu	Hungary
id	Indonesia
ie	Ireland
il	Israel
in	India
iq	Iraq
ir	Iran, Islamic Republic of
is	Iceland
it	Italy
jm	Jamaica
jo	Jordan
jp	Japan
ke	Kenya
kg	Kyrgyzstan
kh	Cambodia
kp	Korea, Democratic People's Republic of
kr	Korea, Republic of
kw	Kuwait
kz	Kazakhstan
Continued on next page	

Table 3.1 – continued from previous page

code	Country
la	Lao People's Democratic Republic
lb	Lebanon
li	Liechtenstein
lk	Sri Lanka
lr	Liberia
ls	Lesotho
lt	Lithuania
lu	Luxembourg
lv	Latvia
ly	Libyan Arab Jamahiriya
ma	Morocco
mc	Monaco
md	Moldova, Republic of
me	Montenegro
mg	Madagascar
mk	Macedonia, the former Yugoslav Republic of
ml	Mali
mm	Myanmar
mn	Mongolia
mo	Macao
mr	Mauritania
mt	Malta
mu	Mauritius
mv	Maldives
mw	Malawi
mx	Mexico
my	Malaysia
mz	Mozambique
na	Namibia
ne	Niger
ng	Nigeria
ni	Nicaragua
nl	Netherlands
no	Norway
np	Nepal
nz	New Zealand
om	Oman
pa	Panama
pe	Peru
pg	Papua New Guinea
ph	Philippines
pk	Pakistan
pl	Poland
pr	Puerto Rico
ps	Palestine, State of
pt	Portugal
py	Paraguay
re	Reunion
ro	Romania

Continued on next page

Table 3.1 – continued from previous page

code	Country
rs	Serbia
ru	Russian Federation
rw	Rwanda
sa	Saudi Arabia
sc	Seychelles
sd	Sudan
se	Sweden
sg	Singapore
sh	Saint Helena, Ascension and Tristan da Cunha
si	Slovenia
sk	Slovakia
sl	Sierra Leone
sm	San Marino
sn	Senegal
so	Somalia
sr	Suriname
st	Sao Tome and Principe
sv	El Salvador
sy	Syrian Arab Republic
sz	Swaziland
td	Chad
tg	Togo
th	Thailand
tj	Tajikistan
tl	Timor-Leste
tm	Turkmenistan
tn	Tunisia
tr	Turkey
tw	Taiwan, Province of China
tz	Tanzania, United Republic of
ua	Ukraine
ug	Uganda
us	United States
uy	Uruguay
uz	Uzbekistan
va	Holy See (Vatican City State)
ve	Venezuela, Bolivarian Republic of
vn	Viet Nam
ye	Yemen
yt	Mayotte
za	South Africa
zm	Zambia
zw	Zimbabwe



**Continent list**

code	name
asia	Asia
europa	Europe
africa	Africa
north_america	North America
south_america	South America
oceania	Oceania
antarctica	Antartica

**French map**

**Installing** The french map plugin can be installed by doing a:

```
pip install pygal_maps_fr
```

**Department** Then you will have access to the `pygal.maps.fr` module.

You can now plot departments (see below for the list):

```
fr_chart = pygal.maps.fr.Departments()
fr_chart.title = 'Some departments'
fr_chart.add('Métropole', ['69', '92', '13'])
fr_chart.add('Corse', ['2A', '2B'])
fr_chart.add('DOM COM', ['971', '972', '973', '974'])
fr_chart.render()
```

Or specify an number for a department:

```
fr_chart = pygal.maps.fr.Departments(human_readable=True)
fr_chart.title = 'Population by department'
fr_chart.add('In 2011', {
    '01': 603827, '02': 541302, '03': 342729, '04': 160959, '05': 138605, '06': 1081244, '07': 317277,
})
fr_chart.render()
```

**Regions** You can do the same with regions:

```
fr_chart = pygal.maps.fr.Regions()
fr_chart.title = 'Some regions'
fr_chart.add('Métropole', ['82', '11', '93'])
fr_chart.add('Corse', ['94'])
fr_chart.add('DOM COM', ['01', '02', '03', '04'])
fr_chart.render()
```

You can also specify a number for a region and use a department to region aggregation:

```
from pygal.maps.fr import aggregate_regions
fr_chart = pygal.maps.fr.Regions(human_readable=True)
fr_chart.title = 'Population by region'
fr_chart.add('In 2011', aggregate_regions({
    '01': 603827, '02': 541302, '03': 342729, '04': 160959, '05': 138605, '06': 1081244, '07': 317277,
}))
fr_chart.render()
```

code	Department
01	Ain
02	Aisne
03	Allier
04	Alpes-de-Haute-Provence
05	Hautes-Alpes
06	Alpes-Maritimes
07	Ardèche
08	Ardennes
09	Ariège
10	Aube
11	Aude
12	Aveyron
13	Bouches-du-Rhône
14	Calvados
15	Cantal
16	Charente
17	Charente-Maritime
18	Cher
19	Corrèze
2A	Corse-du-Sud
2B	Haute-Corse
21	Côte-d'Or
22	Côtes-d'Armor
23	Creuse
24	Dordogne
25	Doubs
26	Drôme
27	Eure
28	Eure-et-Loir
29	Finistère
30	Gard
31	Haute-Garonne
32	Gers
33	Gironde
34	Hérault
35	Ille-et-Vilaine
36	Indre
37	Indre-et-Loire
38	Isère
39	Jura
40	Landes
41	Loir-et-Cher
42	Loire
43	Haute-Loire
44	Loire-Atlantique
45	Loiret
46	Lot
47	Lot-et-Garonne
48	Lozère
49	Maine-et-Loire

Continued on next page

Table 3.2 – continued from previous page

code	Department
50	Manche
51	Marne
52	Haute-Marne
53	Mayenne
54	Meurthe-et-Moselle
55	Meuse
56	Morbihan
57	Moselle
58	Nièvre
59	Nord
60	Oise
61	Orne
62	Pas-de-Calais
63	Puy-de-Dôme
64	Pyrénées-Atlantiques
65	Hautes-Pyrénées
66	Pyrénées-Orientales
67	Bas-Rhin
68	Haut-Rhin
69	Rhône
70	Haute-Saône
71	Saône-et-Loire
72	Sarthe
73	Savoie
74	Haute-Savoie
75	Paris
76	Seine-Maritime
77	Seine-et-Marne
78	Yvelines
79	Deux-Sèvres
80	Somme
81	Tarn
82	Tarn-et-Garonne
83	Var
84	Vaucluse
85	Vendée
86	Vienne
87	Haute-Vienne
88	Vosges
89	Yonne
90	Territoire de Belfort
91	Essonne
92	Hauts-de-Seine
93	Seine-Saint-Denis
94	Val-de-Marne
95	Val-d’Oise
971	Guadeloupe
972	Martinique
973	Guyane
Continued on next page	

Table 3.2 – continued from previous page

code	Department
974	Réunion
975	Saint Pierre et Miquelon
976	Mayotte

### Department list

code	Region
11	Île-de-France
21	Champagne-Ardenne
22	Picardie
23	Haute-Normandie
24	Centre
25	Basse-Normandie
26	Bourgogne
31	Nord-Pas-de-Calais
41	Lorraine
42	Alsace
43	Franche-Comté
52	Pays-de-la-Loire
53	Bretagne
54	Poitou-Charentes
72	Aquitaine
73	Midi-Pyrénées
74	Limousin
82	Rhône-Alpes
83	Auvergne
91	Languedoc-Roussillon
93	Provence-Alpes-Côte d'Azur
94	Corse
01	Guadeloupe
02	Martinique
03	Guyane
04	Réunion
05	Saint Pierre et Miquelon
06	Mayotte

### Region list

### Swiss map

**Installing** The swiss map plugin can be installed by doing a:

```
pip install pygal_maps_ch
```

**Canton** Then you will have access to the `pygal.maps.ch` module.

You can now plot cantons (see below for the list):

```
ch_chart = pygal.maps.ch.Cantons()
ch_chart.title = 'Some cantons'
ch_chart.add('Cantons 1', ['kt-zh', 'kt-be', 'kt-nw'])
```

```
ch_chart.add('Cantons 2', ['kt-ow', 'kt-bs', 'kt-ne'])
ch_chart.render()
```

**Canton list**

code	Canton
kt-zh	ZH
kt-be	BE
kt-lu	LU
kt-ju	JH
kt-ur	UR
kt-sz	SZ
kt-ow	OW
kt-nw	NW
kt-gl	GL
kt-zg	ZG
kt-fr	FR
kt-so	SO
kt-bl	BL
kt-bs	BS
kt-sh	SH
kt-ar	AR
kt-ai	AI
kt-sg	SG
kt-gr	GR
kt-ag	AG
kt-tg	TG
kt-ti	TI
kt-vd	VD
kt-vs	VS
kt-ne	NE
kt-ge	GE

### 3.1.3 Styles

There are three ways to style the charts:

#### Built-in Styles

pygal provides 14 built-in styles:

##### Default

```
from pygal.style import DefaultStyle
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=DefaultStyle) # Setting style here is
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

## DarkStyle

```
from pygal.style import DarkStyle
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=DarkStyle)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

## Neon

```
from pygal.style import NeonStyle
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=NeonStyle)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

## Dark Solarized

```
from pygal.style import DarkSolarizedStyle
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=DarkSolarizedStyle)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

## Light Solarized

```
from pygal.style import LightSolarizedStyle
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=LightSolarizedStyle)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

## Light

```
from pygal.style import LightStyle
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=LightStyle)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
```

```
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

### Clean

```
from pygal.style import CleanStyle
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=CleanStyle)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

### Red Blue

```
from pygal.style import RedBlueStyle
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=RedBlueStyle)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

### Dark Colorized

```
from pygal.style import DarkColorizedStyle
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=DarkColorizedStyle)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

### Light Colorized

```
from pygal.style import LightColorizedStyle
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=LightColorizedStyle)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

### Turquoise

```
from pygal.style import TurquoiseStyle
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=TurquoiseStyle)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

### Light green

```
from pygal.style import LightGreenStyle
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=LightGreenStyle)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

### Dark green

```
from pygal.style import DarkGreenStyle
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=DarkGreenStyle)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

### Dark green blue

```
from pygal.style import DarkGreenBlueStyle
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=DarkGreenBlueStyle)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

### Blue

```
from pygal.style import BlueStyle
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=BlueStyle)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
```



```
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

## Parametric Styles

pygal provides 5 parametric styles:

### Usage

A parametric style is initiated with a default color and the other are generated from this one:

```
from pygal.style import LightenStyle
dark_lighten_style = LightenStyle('#336676')
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=dark_lighten_style)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

You can set the *step* parameter to tell between how much colors the color modifier will be applied

```
from pygal.style import LightenStyle
dark_lighten_style = LightenStyle('#336676', step=5)
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=dark_lighten_style)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

and the *max\_* to limit the amplitude at a certain value (in % for all color operation except rotate which is 360):

```
from pygal.style import LightenStyle
dark_lighten_style = LightenStyle('#336676', step=5, max_=10)
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=dark_lighten_style)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

You can tell the style to inheritate all the styles from another theme:

```
from pygal.style import LightenStyle, LightColorizedStyle
dark_lighten_style = LightenStyle('#336676', base_style=LightColorizedStyle)
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=dark_lighten_style)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

And you can manually set the properties just like any other theme:

```
from pygal.style import LightenStyle, LightColorizedStyle
dark_lighten_style = LightenStyle('#336676', base_style=LightColorizedStyle)
dark_lighten_style.background = '#ffcccc'
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=dark_lighten_style)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

## Styles

### Rotate

```
from pygal.style import RotateStyle
dark_rotate_style = RotateStyle('#9e6ffe')
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=dark_rotate_style)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

```
from pygal.style import RotateStyle, LightColorizedStyle
dark_rotate_style = RotateStyle('#75ff98', base_style=LightColorizedStyle)
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=dark_rotate_style)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

### Lighten

```
from pygal.style import LightenStyle
dark_lighten_style = LightenStyle('#004466')
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=dark_lighten_style)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

### Darken

```
from pygal.style import DarkenStyle
darken_style = DarkenStyle('#ff8723')
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=darken_style)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
```

```
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

### Saturate

```
from pygal.style import SaturateStyle
saturate_style = SaturateStyle('#609f86')
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=saturate_style)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

### Desaturate

```
from pygal.style import DesaturateStyle
desaturate_style = DesaturateStyle('#8322dd', step=8)
chart = pygal.StackedLine(fill=True, interpolate='cubic', style=desaturate_style)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

## Custom Styles

pygal provides 2 ways to customize styles:

### Using Style class

You can instantiate the Style class with some customizations for quick styling:

```
from pygal.style import Style
custom_style = Style(
    background='transparent',
    plot_background='transparent',
    foreground='#53E89B',
    foreground_strong='#53A0E8',
    foreground_subtle='#630C0D',
    opacity='.6',
    opacity_hover='.9',
    transition='400ms ease-in',
    colors=('E853A0', 'E8537A', 'E95355', 'E87653', 'E89B53'))

chart = pygal.StackedLine(fill=True, interpolate='cubic', style=custom_style)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()
```

**Properties** Style objects supports the following properties:

Properties	Description
plot_background	The color of the chart area background
background	The color of the image background
foreground	The main foreground color
foreground_strong	The emphasis foreground color
foreground_subtle	The subtle foreground color
font_family	The main font family
label_font_family	The label font family
major_label_font_family	The major label font family
value_font_family	The <code>print_values</code> font family
value_label_font_family	The <code>print_labels</code> font family
tooltip_font_family	The tooltip font family
title_font_family	The title font family
legend_font_family	The legend font family
no_data_font_family	The no data text font family
guide_stroke_dasharray	The dasharray for guide line
major_guide_stroke_dasharray	The dasharray for major guide line
label_font_size	The label font size
major_label_font_size	The major label font size
value_font_size	The <code>print_values</code> font size
value_label_font_size	The <code>print_labels</code> font size
tooltip_font_size	The tooltip font size
title_font_size	The title font size
legend_font_size	The legend font size
no_data_font_size	The no data font size
opacity	The opacity of chart element
opacity_hover	The opacity of chart element on mouse hover
transition	Define the global transition property for animation
colors	The serie color list
value_colors	The <code>print_values</code> color list

**Google font** It is possible to give a google font to any font family property by specifying the `googlefont :` prefix:

```
style = Style(font_family='googlefont:Raleway')
```

### Using a custom css

You can also specify a file containing a custom css for more customization. The `css` option is an array containing included css by default (except from `base.css` which is always included).

It supports local file names and external stylesheet too, just append your URI in the list.

(See the [default css](#))

NB: Now the css rules are prefixed by an unique id, to prevent collisions when including several svg directly into a web page. You can disable it with the `no_prefix` option.

```
from tempfile import NamedTemporaryFile
custom_css = '''
  {{ id }}text {
    fill: green;
    font-family: monospace;
```

```

}
{{ id }}.legends .legend text {
    font-size: {{ font_sizes.legend }};
}
{{ id }}.axis {
    stroke: #666;
}
{{ id }}.axis text {
    font-size: {{ font_sizes.label }};
    font-family: sans;
    stroke: none;
}
{{ id }}.axis.y text {
    text-anchor: end;
}
{{ id }}#tooltip text {
    font-size: {{ font_sizes.tooltip }};
}
{{ id }}.dot {
    fill: yellow;
}
{{ id }}.color-0 {
    stroke: #ff1100;
    fill: #ff1100;
}
{{ id }}.color-1 {
    stroke: #ffee00;
    fill: #ffee00;
}
{{ id }}.color-2 {
    stroke: #66bb44;
    fill: #66bb44;
}
{{ id }}.color-3 {
    stroke: #88bbdd;
    fill: #88bbdd;
}
{{ id }}.color-4 {
    stroke: #0000ff;
    fill: #0000ff;
}
'''
custom_css_file = '/tmp/pygal_custom_style.css'
with open(custom_css_file, 'w') as f:
    f.write(custom_css)
config = pygal.Config(fill=True, interpolate='cubic')
config.css.append('file://' + custom_css_file)
chart = pygal.StackedLine(config)
chart.add('A', [1, 3, 5, 16, 13, 3, 7])
chart.add('B', [5, 2, 3, 2, 5, 7, 17])
chart.add('C', [6, 10, 9, 7, 3, 1, 0])
chart.add('D', [2, 3, 5, 9, 12, 9, 5])
chart.add('E', [7, 4, 2, 1, 2, 10, 0])
chart.render()

```

### 3.1.4 Chart configuration

#### How

pygal is customized at chart level with the help of the Config class).

#### Instance

The config class works this way:

```
from pygal import Config

config = Config()
config.show_legend = False
config.human_readable = True
config.fill = True
chart = pygal.XY(config)
...
```

and you can share the config object between several charts. For one shot chart rendering several shorthand are available:

#### Attribute

Config values are settable on the chart object.

```
chart = pygal.XY(config)
chart.show_legend = False
chart.human_readable = True
chart.fill = True
...
```

#### Keyword args

Config values can be given as keyword args at init:

```
chart = pygal.XY(show_legend=False, human_readable=True, fill=True)
```

And at render:

```
chart = pygal.XY()
chart.render(show_legend=False, human_readable=True, fill=True)
```

#### Options

##### Sizing

Svg size is configurable with `width` and `height` parameter.

##### `width`

```
chart = pygal.Bar(width=200)
chart.add('1', 1)
chart.add('2', 2)
chart.render()
```

**height**

```
chart = pygal.Bar(height=100)
chart.add('1', 1)
chart.add('2', 2)
chart.render()
```

**explicit\_size** Size can be written directly to the svg tag to force display of the requested size using `explicit_size`.

**spacing** Spacing determines the space between all elements:

```
chart = pygal.Bar(spacing=50)
chart.x_labels = u'αβγδ'
chart.add('line 1', [5, 15, 10, 8])
chart.add('line 2', [15, 20, 8, 11])
chart.render()
```

**margin** Margin is the external chart margin:

```
chart = pygal.Bar(margin=50)
chart.x_labels = u'αβγδ'
chart.add('line 1', [5, 15, 10, 8])
chart.add('line 2', [15, 20, 8, 11])
chart.render()
```

Individual margins can also be specified

**margin\_top**

```
chart = pygal.Bar(margin_top=50)
chart.x_labels = u'αβγδ'
chart.add('line 1', [5, 15, 10, 8])
chart.add('line 2', [15, 20, 8, 11])
chart.render()
```

**margin\_right**

```
chart = pygal.Bar(margin_right=50)
chart.x_labels = u'αβγδ'
chart.add('line 1', [5, 15, 10, 8])
chart.add('line 2', [15, 20, 8, 11])
chart.render()
```

**margin\_bottom**

```
chart = pygal.Bar(margin_bottom=50)
chart.x_labels = u'αβγδ'
chart.add('line 1', [5, 15, 10, 8])
```

```
chart.add('line 2', [15, 20, 8, 11])
chart.render()
```

### margin\_left

```
chart = pygal.Bar(margin_left=50)
chart.x_labels = u'αβγδ'
chart.add('line 1', [5, 15, 10, 8])
chart.add('line 2', [15, 20, 8, 11])
chart.render()
```

## Titles

**title** You can add a title to the chart by setting the `title` option:

```
chart = pygal.Line(title=u'Some points')
chart.add('line', [.0002, .0005, .00035])
chart.render()
```

**x\_title** You can add a title to the x axis by setting the `x_title` option:

```
chart = pygal.Line(title=u'Some points', x_title='X Axis')
chart.add('line', [.0002, .0005, .00035])
chart.render()
```

**y\_title** You can add a title to the y axis by setting the `y_title` option:

```
chart = pygal.Line(title=u'Some points', y_title='Y Axis')
chart.add('line', [.0002, .0005, .00035])
chart.render()
```

## Labels

You can specify x labels and y labels, depending on the graph type:

### x\_labels

```
chart = pygal.Line()
chart.x_labels = 'Red', 'Blue', 'Green'
chart.add('line', [.0002, .0005, .00035])
chart.render()
```

It is possible for dual charts to define a custom scale:

```
chart = pygal.XY()
chart.x_labels = (.00012, .00024, .00048, .00096)
chart.add('line', [(.0002, 10), (.0005, 20), (.00035, 15)])
chart.render()
```

And in this case it is possible to set text labels in place of values:



```

chart = pygal.XY()
chart.x_labels = ({
    'label': 'Twelve',
    'value': .00012
}, {
    'label': 'Twenty four',
    'value': .00024
}, {
    'label': 'Forty eight',
    'value': .00048
}, {
    'label': 'Ninety six',
    'value': .00096})
chart.add('line', [(0.0002, 10), (0.0005, 20), (0.00035, 15)])
chart.render()

```

### y\_labels

```

chart = pygal.Line()
chart.y_labels = .0001, .0003, .0004, .00045, .0005
chart.add('line', [.0002, .0005, .00035])
chart.render()

```

It is now possible to add text to labels values:

```

chart = pygal.Line()
chart.y_labels = [
    {'label': 'One', 'value': .0001},
    {'label': 'Three', 'value': .0003},
    {'label': 'Four', 'value': .0004},
    {'label': 'Four and a half', 'value': .00045},
    {'label': 'Five', 'value': .0005}]
chart.add('line', [.0002, .0005, .00035])
chart.render()

```

**show\_x\_labels** Set this to False to deactivate x labels:

```

chart = pygal.Line(show_x_labels=False)
chart.x_labels = 'Red', 'Blue', 'Green'
chart.add('line', [.0002, .0005, .00035])
chart.render()

```

**show\_y\_labels** Set this to False to deactivate y labels:

```

chart = pygal.Line(show_y_labels=False)
chart.x_labels = 'Red', 'Blue', 'Green'
chart.add('line', [.0002, .0005, .00035])
chart.render()

```

Allow label rotation (in degrees) to avoid axis cluttering:

```

chart = pygal.Line()
chart.x_labels = [
    'This is the first point !',
    'This is the second point !',
    'This is the third point !',
    'This is the fourth point !']

```

```
chart.add('line', [0, .0002, .0005, .00035])
chart.render()
```

### **x\_label\_rotation**

```
chart = pygal.Line(x_label_rotation=20)
chart.x_labels = [
    'This is the first point !',
    'This is the second point !',
    'This is the third point !',
    'This is the fourth point !']
chart.add('line', [0, .0002, .0005, .00035])
chart.render()
```

### **y\_label\_rotation**

```
chart = pygal.Line(y_label_rotation=20)
chart.add('line', [0, .0002, .0005, .00035])
chart.render()
```

You can alter major minor behaviour of axes thanks to [Arjen Stolk](#)

### **x\_labels\_major**

```
chart = pygal.Line(x_label_rotation=20)
chart.x_labels = [
    'This is the first point !',
    'This is the second point !',
    'This is the third point !',
    'This is the fourth point !']
chart.x_labels_major = ['This is the first point !', 'This is the fourth point !']
chart.add('line', [0, .0002, .0005, .00035])
chart.render()
```

### **x\_labels\_major\_every**

```
chart = pygal.Line(x_label_rotation=20, x_labels_major_every=3)
chart.x_labels = [
    'This is the first point !',
    'This is the second point !',
    'This is the third point !',
    'This is the fourth point !']
chart.add('line', [0, .0002, .0005, .00035])
chart.render()
```

### **x\_labels\_major\_count**

```
chart = pygal.Line(x_label_rotation=20, x_labels_major_count=3)
chart.x_labels = [
    'This is the first point !',
    'This is the second point !',
    'This is the third point !',
    'This is the fourth point !']
chart.add('line', [0, .0002, .0005, .00035])
chart.render()
```

**show\_minor\_x\_labels**

```

chart = pygal.Line(x_label_rotation=20, show_minor_x_labels=False)
chart.x_labels = [
    'This is the first point !',
    'This is the second point !',
    'This is the third point !',
    'This is the fourth point !']
chart.x_labels_major = ['This is the first point !', 'This is the fourth point !']
chart.add('line', [0, .0002, .0005, .00035])
chart.render()

```

**y\_labels\_major**

```

chart = pygal.Line(y_label_rotation=-20)
chart.y_labels_major = []
chart.add('line', [0, .0002, .0005, .00035])
chart.render()

```

```

chart = pygal.Line()
chart.y_labels_major = [.0001, .0004]
chart.add('line', [0, .0002, .0005, .00035])
chart.render()

```

**y\_labels\_major\_every**

```

chart = pygal.Line(y_label_rotation=20, y_labels_major_every=3)
chart.add('line', [0, .0002, .0005, .00035])
chart.render()

```

**y\_labels\_major\_count**

```

chart = pygal.Line(y_labels_major_count=3)
chart.add('line', [0, .0002, .0005, .00035])
chart.render()

```

**show\_minor\_y\_labels**

```

chart = pygal.Line(y_labels_major_every=2, show_minor_y_labels=False)
chart.add('line', [0, .0002, .0005, .00035])
chart.render()

```

**truncate\_label** By default long labels are automatically truncated at reasonable length to fit in the graph.

You can override that by setting truncation length with `truncate_label`.

```

chart = pygal.Line(truncate_label=17)
chart.x_labels = [
    'This is the first point !',
    'This is the second point !',
    'This is the third point !',
    'This is the fourth point !']
chart.add('line', [0, .0002, .0005, .00035])
chart.render()

```

or disable it by setting this to -1

```
chart = pygal.Line(truncate_label=-1)
chart.x_labels = [
    'This is the first point !',
    'This is the second point !',
    'This is the third point !',
    'This is the fourth point !']
chart.add('line', [0, .0002, .0005, .00035])
chart.render()
```

## Legend

**show\_legend** You can remove legend by setting this to `False`

```
chart = pygal.Line(show_legend=False)
chart.add('Serie 1', [1, 2, 3])
chart.add('Serie 2', [4, 2, 0])
chart.add('Serie 3', [1, -1, 1])
chart.add('Serie 4', [3, 1, 5])
chart.render()
```

**legend\_at\_bottom** You can put legend at bottom by setting `legend_at_bottom` to `True`:

```
chart = pygal.Line(legend_at_bottom=True)
chart.add('Serie 1', [1, 2, 3])
chart.add('Serie 2', [4, 2, 0])
chart.add('Serie 3', [1, -1, 1])
chart.add('Serie 4', [3, 1, 5])
chart.render()
```

**legend\_at\_bottom\_columns** Force the number of legend columns when set at bottom

```
chart = pygal.Line(legend_at_bottom=True, legend_at_bottom_columns=4)
chart.add('Serie 1', [1, 2, 3])
chart.add('Serie 2', [4, 2, 0])
chart.add('Serie 3', [1, -1, 1])
chart.add('Serie 4', [3, 1, 5])
chart.render()
```

## legend\_box\_size

```
chart = pygal.Line(legend_box_size=18)
chart.add('Serie 1', [1, 2, 3])
chart.add('Serie 2', [4, 2, 0])
chart.add('Serie 3', [1, -1, 1])
chart.add('Serie 4', [3, 1, 5])
chart.render()
```

**truncate\_legend** By default long legends are automatically truncated at reasonable length to fit in the graph.

You can override that by setting truncation length with `truncate_legend`.

```
chart = pygal.Line(truncate_legend=17)
chart.x_labels = [
    'This is the first point !',
```

```
'This is the second point !',
'This is the third point !',
'This is the fourth point !']
chart.add('line', [0, .0002, .0005, .00035])
chart.render()
```

or disable it by setting this to -1

```
chart = pygal.Line(truncate_legend=-1)
chart.x_labels = [
'This is the first point !',
'This is the second point !',
'This is the third point !',
'This is the fourth point !']
chart.add('line', [0, .0002, .0005, .00035])
chart.render()
```

## Axis

**include\_x\_axis** Scales are computed automatically between the min and the max values.

You may want to always have the abscissa in your graph:

```
chart = pygal.Line(include_x_axis=True)
chart.add('line', [.0002, .0005, .00035])
chart.render()
```

## inverse\_y\_axis

```
chart = pygal.Line(inverse_y_axis=True)
chart.add('line', [.0002, .0005, .00035])
chart.render()
```

**range** In pygal you can override automatic scaling by setting `y_labels` to the values you want, but if you want to change the scaling range and keep auto scaling in it, you can set a `range` which is a tuple containing the desired min and max:

```
chart = pygal.Line(range=(.0001, .001))
chart.add('line', [.0002, .0005, .00035])
chart.render()
```

**xrange** For xy graph xrange can be used for the x axis.

```
chart = pygal.XY(xrange=(10, 30))
chart.add('line', [(10, .0002), (15, .0005), (12, .00035)])
chart.render()
```

**secondary\_range** For chart with two axis, the `secondary_range` defines the range for the secondary axis.

```
chart = pygal.Line(secondary_range=(10, 25))
chart.add('primary', [.0002, .0005, .00035])
chart.add('secondary', [10, 15, 12], secondary=True)
chart.render()
```

**logarithmic** You can set the scale to be logarithmic:

```
chart = pygal.Line(logarithmic=True)
values = [1, 3, 43, 123, 1231, 23192]
chart.x_labels = map(str, values)
chart.add('log example', values)
chart.render()
```

**Caution:** Negative values are ignored

**min\_scale** You can specify the minimum number of scale graduation to generate with auto scaling if possible.

```
chart = pygal.Line(min_scale=12)
chart.add('line', [1, 10, 100, 50, 25])
chart.render()
```

**max\_scale** You can specify the maximum number of scale graduation to generate with auto scaling if possible.

```
chart = pygal.Line(max_scale=6)
chart.add('line', [1, 10, 100, 50, 25])
chart.render()
```

**order\_min** You can specify at which precision pygal should stop scaling (in log10) usefull in conjunction of the two previous properties:

```
chart = pygal.Line(order_min=1)
chart.add('line', [1, 10, 100, 50, 25])
chart.render()
```

## Interpolations

pygal allow you to interpolate most of line charts. Take this chart for instance:

```
chart = pygal.Line()
chart.add('line', [1, 5, 17, 12, 5, 10])
chart.render()
```

### interpolate

**cubic** You can set the cubic interpolation:

```
chart = pygal.Line(interpolate='cubic')
chart.add('line', [1, 5, 17, 12, 5, 10])
chart.render()
```

### quadratic

```
chart = pygal.Line(interpolate='quadratic')
chart.add('line', [1, 5, 17, 12, 5, 10])
chart.render()
```

**lagrange**

```
chart = pygal.Line(interpolate='lagrange')
chart.add('line', [1, 5, 17, 12, 5, 10])
chart.render()
```

**trigonometric**

```
chart = pygal.Line(interpolate='trigonometric')
chart.add('line', [1, 5, 17, 12, 5, 10])
chart.render()
```

**hermite**

```
chart = pygal.Line(interpolate='hermite')
chart.add('line', [1, 5, 17, 12, 5, 10])
chart.render()
```

**interpolation\_parameters** For hermite you can also pass additional parameters to configure tangent behaviour:

```
chart = pygal.Line(interpolate='hermite', interpolation_parameters={'type': 'finite_difference'})
chart.add('line', [1, 5, 17, 12, 5, 10])
chart.render()
```

```
chart = pygal.Line(interpolate='hermite', interpolation_parameters={'type': 'cardinal', 'c': .75})
chart.add('line', [1, 5, 17, 12, 5, 10])
chart.render()
```

```
chart = pygal.Line(interpolate='hermite', interpolation_parameters={'type': 'kochanek_bartels', 'b': .75})
chart.add('line', [1, 5, 17, 12, 5, 10])
chart.render()
```

For more information see the [wikipedia article](#)

**interpolation\_precision** You can change the resolution of the interpolation with the help of `interpolation_precision`:

```
chart = pygal.Line(interpolate='quadratic')
chart.add('line', [1, 5, 17, 12, 5, 10])
chart.render()
```

```
chart = pygal.Line(interpolate='quadratic', interpolation_precision=3)
chart.add('line', [1, 5, 17, 12, 5, 10])
chart.render()
```

**Data**

**value\_formatter** You can specify how the values are displayed on the tooltip using a lambda function. The code below shows the values to 2 decimal places.

```
chart = pygal.Line()
chart.add('line', [.070106781, 1.414213562, 3.141592654])
chart.value_formatter = lambda x: "%.2f" % x
chart.render()
```

**x\_value\_formatter** Same on x axis for xy like charts:

```
chart = pygal.XY()
chart.add('line', [(12, 31), (8, 28), (89, 12)])
chart.x_value_formatter = lambda x: '%s%%' % x
chart.render()
```

**print\_values** When using pygal to display static charts for printing for example you can chose to activate this option to print all values as text.

```
from pygal.style import DefaultStyle
chart = pygal.Bar(print_values=True, style=DefaultStyle(
    value_font_family='googlefont:Raleway',
    value_font_size=30,
    value_colors=('white',)))
chart.add('line', [0, 12, 31, 8, -28, 0])
chart.render()
```

**dynamic\_print\_values** Show print\_values only on legend hover.

```
from pygal.style import DefaultStyle
chart = pygal.Bar(dynamic_print_values=True, style=DefaultStyle(
    value_font_family='googlefont:Raleway',
    value_font_size=30,
    value_colors=('white',)))
chart.add('line', [0, 12, 31, 8, -28, 0])
chart.render()
```

**print\_values\_position** Change print value position (in bar charts only).

```
chart = pygal.Bar(print_values=True, print_values_position='top')
chart.add('line', [0, 12, 31, 8, -28, 0])
chart.render()
```

```
chart = pygal.Bar(print_values=True, print_values_position='bottom')
chart.add('line', [0, 12, 31, 8, -28, 0])
chart.render()
```

**print\_zeroes** zero values are shown by default but you can use this option to hide them.

```
chart = pygal.Bar(print_values=True, print_zeroes=False)
chart.add('line', [0, 12, 31, 8, -28, 0])
chart.render()
```

**print\_labels** You can activate value label display:

```
chart = pygal.Bar(print_labels=True)
chart.add('line', [
    0,
    {'value': 12, 'label': 'Twelve'},
    31,
    {'value': 8, 'label': 'eight'},
    28,
    0
```



```
])
chart.render()
```

Displaying both is also possible:

```
chart = pygal.Bar(print_labels=True, print_values=True)
chart.add('line', [
    0,
    {'value': 12, 'label': 'Twelve'},
    31,
    {'value': 8, 'label': 'eight'},
    28,
    0
])
chart.render()
```

**human\_readable** Display values in human readable form:

```
1 230 000 -> 1.23M
.00 098 7 -> 987µ
```

```
chart = pygal.Line(human_readable=True)
chart.add('line', [0, .0002, .0005, .00035])
chart.render()
```

**no\_data\_text** Text to display instead of the graph when no data is supplied:

```
chart = pygal.Line()
chart.add('line', [])
chart.render()
```

```
from pygal.style import DefaultStyle
chart = pygal.Line(no_data_text='No result found',
                  style=DefaultStyle(no_data_font_size=40))
chart.add('line', [])
chart.render()
```

## Tooltip

Tooltips are displayed when the pygal javascript is used.

### tooltip\_border\_radius

```
chart = pygal.Line(tooltip_border_radius=10)
chart.add('line', [.0002, .0005, .00035])
chart.render()
```

## Rendering

**stroke** On line graphs you can disable line stroking:

```
chart = pygal.Line(stroke=False)
chart.add('line', [.0002, .0005, .00035])
chart.render()
```

**fill** And enable line filling:

```
chart = pygal.Line(fill=True)
chart.add('line', [.0002, .0005, .00035])
chart.render()
```

**zero** To fill to an other reference than zero:

```
chart = pygal.Line(fill=True, zero=.0004)
chart.add('line', [.0002, .0005, .00035])
chart.render()
```

**show\_dots** You can remove dots by setting `show_dots` at `False`

```
chart = pygal.Line(show_dots=False)
chart.add('line', [.0002, .0005, .00035])
chart.render()
```

**show\_only\_major\_dots** You can remove minor x-labelled dots by setting `show_only_major_dots` at `True`

```
chart = pygal.Line(show_only_major_dots=True)
chart.add('line', range(12))
chart.x_labels = map(str, range(12))
chart.x_labels_major = ['2', '4', '8', '11']
chart.render()
```

**dots\_size** You can change the dot size

```
chart = pygal.Line(dots_size=5)
chart.add('line', [.0002, .0005, .00035])
chart.render()
```

**stroke\_style** It is possible to set a default style for lines with the `stroke_style` dictionary.

```
chart = pygal.Line(stroke_style={'width': 5, 'dasharray': '3, 6', 'linecap': 'round', 'linejoin': 'round'})
chart.add('line', [.0002, .0005, .00035])
chart.render()
```

**show\_x\_guides** You can force the display of x guides

```
chart = pygal.Line(show_x_guides=True)
chart.x_labels = ['alpha', 'beta', 'gamma']
chart.add('line', [.0002, .0005, .00035])
chart.render()
```

**show\_y\_guides** Or disable y guides:

```
chart = pygal.Line(show_y_guides=False)
chart.x_labels = ['alpha', 'beta', 'gamma']
chart.add('line', [.0002, .0005, .00035])
chart.render()
```

**style** see styles

You can add or replace css/js files in pygal using the `css` and `js` array options. These lists contain absolute filenames and/or external URI. (Relative filenames are relative to pygal internal files)

All config lists now support the use of ellipsis as an extender. For instance:

```
config = Config()
config.css.append('style.css')
chart = pygal.Line(config)
```

can now be replaced with:

```
chart = pygal.Line(css=(..., 'style.css'))
```

or if you are still using python from the last decade:

```
from pygal._compat import _ellipsis
chart = pygal.Line(css=( _ellipsis, 'style.css'))
```

**css** Default:

```
css = ['file://style.css', 'file://graph.css']
```

Css can also specified inline by prepending *inline:* to the css:

```
css = ['inline:.rect { fill: blue; }']
```

**classes** You can alter pygal svg node classes with the `classes` option:

```
chart = pygal.Line(classes=(..., 'flex'))
```

**defs** You can add defs like `linearGradient`, `radialGradient`, `pattern` to the `defs` config:

```
config = pygal.Config()
config.style = pygal.style.DarkStyle
config.defs.append('''
<linearGradient id="gradient-0" x1="0" x2="0" y1="0" y2="1">
  <stop offset="0%" stop-color="#ff5995" />
  <stop offset="100%" stop-color="#feed6c" />
</linearGradient>
''')
config.defs.append('''
<linearGradient id="gradient-1" x1="0" x2="0" y1="0" y2="1">
  <stop offset="0%" stop-color="#b6e354" />
  <stop offset="100%" stop-color="#8cedff" />
</linearGradient>
''')
config.css.append('''inline:
.color-0 {
  fill: url(#gradient-0) !important;
  stroke: url(#gradient-0) !important;
}''')
config.css.append('''inline:
.color-1 {
  fill: url(#gradient-1) !important;
  stroke: url(#gradient-1) !important;
}''')
```

```
}''')
chart = pygal.Line(config)
chart.add('1', [1, 3, 12, 3, 4, None, 9])
chart.add('2', [7, -4, 10, None, 8, 3, 1])
chart.x_labels = ('a', 'b', 'c', 'd', 'e', 'f', 'g')
chart.legend_at_bottom = True
chart.interpolate = 'cubic'
chart.render()
```

### js

```
js = [
    '//kozea.github.io/pygal.js/2.0.x/pygal-tooltips.min.js'
]
```

See [pygal.js](#)

**force\_uri\_protocol** In case of rendering the svg as a data uri, it is mandatory to specify a protocol.

It can be set to http or https and will be used for '//domain/' like uri.

It is used along with `render_data_uri`.

### Misc

**pretty\_print** You can enable pretty print if you want to edit the source by hand (look at this frame source):

```
chart = pygal.Bar(pretty_print=True)
chart.add('values', [3, 10, 7, 2, 9, 7])
chart.render()
```

**disable\_xml\_declaration** When you want to embed directly your SVG in your html, this option disables the xml prolog in the output.

Since no encoding is declared, the result will be in unicode instead of bytes.

**no\_prefix** Normally pygal set an unique id to the chart and use it to style each chart to avoid collisions when svg are directly embedded in html. This can be a problem if you use external styling overriding the prefixed css. You can set this to True in order to prevent that behaviour.

**strict** This activates strict value mode which disable some data adapting and filters. This will make a logarithmic chart crash on negative values for example.

### Specific options

These options are specific for certain chart types.

**rounded\_bars** You can add a round effect to bar diagrams with `rounded_bars`:

```
chart = pygal.Bar(rounded_bars=20)
chart.add('values', [3, 10, 7, 2, 9, 7])
chart.render()
```

**half\_pie**

```
pie_chart = pygal.Pie(half_pie=True)
pie_chart.title = 'Browser usage in February 2012 (in %)'
pie_chart.add('IE', 19.5)
pie_chart.add('Firefox', 36.6)
pie_chart.add('Chrome', 36.3)
pie_chart.add('Safari', 4.5)
pie_chart.add('Opera', 2.3)
pie_chart.render()
```

**inner\_radius** Donut like pies

```
pie_chart = pygal.Pie(inner_radius=.6)
pie_chart.title = 'Browser usage in February 2012 (in %)'
pie_chart.add('IE', 19.5)
pie_chart.add('Firefox', 36.6)
pie_chart.add('Chrome', 36.3)
pie_chart.add('Safari', 4.5)
pie_chart.add('Opera', 2.3)
pie_chart.render()
```

**box\_mode** box plot has several modes:**extremes**

```
box_plot = pygal.Box(box_mode="extremes")
box_plot.title = 'V8 benchmark results'
box_plot.add('Chrome', [6395, 8212, 7520, 7218, 12464, 1660, 2123, 8607])
box_plot.add('Firefox', [7473, 8099, 11700, 2651, 6361, 1044, 3797, 9450])
box_plot.add('Opera', [3472, 2933, 4203, 5229, 5810, 1828, 9013, 4669])
box_plot.add('IE', [43, 41, 59, 79, 144, 136, 34, 102])
box_plot.render()
```

**1.5IQR**

```
box_plot = pygal.Box(box_mode="1.5IQR")
box_plot.title = 'V8 benchmark results'
box_plot.add('Chrome', [6395, 8212, 7520, 7218, 12464, 1660, 2123, 8607])
box_plot.add('Firefox', [7473, 8099, 11700, 2651, 6361, 1044, 3797, 9450])
box_plot.add('Opera', [3472, 2933, 4203, 5229, 5810, 1828, 9013, 4669])
box_plot.add('IE', [43, 41, 59, 79, 144, 136, 34, 102])
box_plot.render()
```

**tukey**

```
box_plot = pygal.Box(box_mode="tukey")
box_plot.title = 'V8 benchmark results'
box_plot.add('Chrome', [6395, 8212, 7520, 7218, 12464, 1660, 2123, 8607])
box_plot.add('Firefox', [7473, 8099, 11700, 2651, 6361, 1044, 3797, 9450])
box_plot.add('Opera', [3472, 2933, 4203, 5229, 5810, 1828, 9013, 4669])
box_plot.add('IE', [43, 41, 59, 79, 144, 136, 34, 102])
box_plot.render()
```

### stdev

```
box_plot = pygal.Box(box_mode="stdev")
box_plot.title = 'V8 benchmark results'
box_plot.add('Chrome', [6395, 8212, 7520, 7218, 12464, 1660, 2123, 8607])
box_plot.add('Firefox', [7473, 8099, 11700, 2651, 6361, 1044, 3797, 9450])
box_plot.add('Opera', [3472, 2933, 4203, 5229, 5810, 1828, 9013, 4669])
box_plot.add('IE', [43, 41, 59, 79, 144, 136, 34, 102])
box_plot.render()
```

### pstdev

```
box_plot = pygal.Box(box_mode="pstdev")
box_plot.title = 'V8 benchmark results'
box_plot.add('Chrome', [6395, 8212, 7520, 7218, 12464, 1660, 2123, 8607])
box_plot.add('Firefox', [7473, 8099, 11700, 2651, 6361, 1044, 3797, 9450])
box_plot.add('Opera', [3472, 2933, 4203, 5229, 5810, 1828, 9013, 4669])
box_plot.add('IE', [43, 41, 59, 79, 144, 136, 34, 102])
box_plot.render()
```

**stack\_from\_top** You can reverse the stacking order for StackedBar and StackedLine

```
line_chart = pygal.StackedLine(fill=True)
line_chart.title = 'Browser usage evolution (in %)'
line_chart.x_labels = map(str, range(2002, 2013))
line_chart.add('Firefox', [None, None, 0, 16.6, 25, 31, 36.4, 45.5, 46.3, 42.8, 37.1])
line_chart.add('Chrome', [None, None, None, None, None, None, 0, 3.9, 10.8, 23.8, 35.3])
line_chart.add('IE', [85.8, 84.6, 84.7, 74.5, 66, 58.6, 54.7, 44.8, 36.2, 26.6, 20.1])
line_chart.add('Others', [14.2, 15.4, 15.3, 8.9, 9, 10.4, 8.9, 5.8, 6.7, 6.8, 7.5])
line_chart.render()
```

```
line_chart = pygal.StackedLine(stack_from_top=True, fill=True)
line_chart.title = 'Browser usage evolution (in %)'
line_chart.x_labels = map(str, range(2002, 2013))
line_chart.add('Firefox', [None, None, 0, 16.6, 25, 31, 36.4, 45.5, 46.3, 42.8, 37.1])
line_chart.add('Chrome', [None, None, None, None, None, None, 0, 3.9, 10.8, 23.8, 35.3])
line_chart.add('IE', [85.8, 84.6, 84.7, 74.5, 66, 58.6, 54.7, 44.8, 36.2, 26.6, 20.1])
line_chart.add('Others', [14.2, 15.4, 15.3, 8.9, 9, 10.4, 8.9, 5.8, 6.7, 6.8, 7.5])
line_chart.render()
```

```
line_chart = pygal.StackedBar()
line_chart.title = 'Browser usage evolution (in %)'
line_chart.x_labels = map(str, range(2002, 2013))
line_chart.add('Firefox', [None, None, 0, 16.6, 25, 31, 36.4, 45.5, 46.3, 42.8, 37.1])
line_chart.add('Chrome', [None, None, None, None, None, None, 0, 3.9, 10.8, 23.8, 35.3])
line_chart.add('IE', [85.8, 84.6, 84.7, 74.5, 66, 58.6, 54.7, 44.8, 36.2, 26.6, 20.1])
line_chart.add('Others', [14.2, 15.4, 15.3, 8.9, 9, 10.4, 8.9, 5.8, 6.7, 6.8, 7.5])
line_chart.render()
```

```
line_chart = pygal.StackedBar(stack_from_top=True)
line_chart.title = 'Browser usage evolution (in %)'
line_chart.x_labels = map(str, range(2002, 2013))
line_chart.add('Firefox', [None, None, 0, 16.6, 25, 31, 36.4, 45.5, 46.3, 42.8, 37.1])
line_chart.add('Chrome', [None, None, None, None, None, None, 0, 3.9, 10.8, 23.8, 35.3])
line_chart.add('IE', [85.8, 84.6, 84.7, 74.5, 66, 58.6, 54.7, 44.8, 36.2, 26.6, 20.1])
line_chart.add('Others', [14.2, 15.4, 15.3, 8.9, 9, 10.4, 8.9, 5.8, 6.7, 6.8, 7.5])
line_chart.render()
```

**missing\_value\_fill\_truncation** Filled series with missing x and/or y values at the end of a series are closed at the first value with a missing. 'x' is default.

### 3.1.5 Serie configuration

#### How

Series are customized using keyword args set in the add or call function:

```
chart = pygal.Line()
chart(1, 2, 3, fill=True)
chart.add(' ', [3, 2, 1], dot=False)
```

#### Options

- *secondary*
- *stroke*
- *fill*
- *show\_dots*
- *show\_only\_major\_dots*
- *dots\_size*
- *stroke\_style*
- *rounded\_bars*
- *inner\_radius*
- *allow\_interruptions*
- *formatter*

#### secondary

You can plot your values to 2 separate axes, thanks to [wiktorin](#) This is the only serie only option.

```
chart = pygal.Line(title=u'Some different points')
chart.x_labels = ('one', 'two', 'three')
chart.add('line', [.0002, .0005, .00035])
chart.add('other line', [1000, 2000, 7000], secondary=True)
chart.render()
```

#### stroke

```
xy_chart = pygal.XY(stroke=False)
xy_chart.title = 'Correlation'
xy_chart.add('A', [(0, 0), (.1, .2), (.3, .1), (.5, 1), (.8, .6), (1, 1.08), (1.3, 1.1), (2, 3.23),
xy_chart.add('B', [(0.1, .15), (.12, .23), (.4, .3), (.6, .4), (.21, .21), (.5, .3), (.6, .8), (.7, .8),
xy_chart.add('C', [(0.05, .01), (.13, .02), (1.5, 1.7), (1.52, 1.6), (1.8, 1.63), (1.5, 1.82), (1.7, 1.8),
xy_chart.add('Correl', [(0, 0), (2.8, 2.4)], stroke=True)
xy_chart.render()
```

### fill

```
chart = pygal.Line()
chart.add('line', [.0002, .0005, .00035], fill=True)
chart.add('line', [.0004, .0009, .001])
chart.render()
```

### show\_dots

```
chart = pygal.Line()
chart.add('line', [.0002, .0005, .00035], show_dots=False)
chart.add('line', [.0004, .0009, .001])
chart.render()
```

### show\_only\_major\_dots

```
chart = pygal.Line()
chart.add('line', range(12))
chart.add('line', range(12)[::-1], show_only_major_dots=True)
chart.x_labels = map(str, range(12))
chart.x_labels_major = ['2', '4', '8', '11']
chart.render()
```

### dots\_size

```
chart = pygal.Line()
chart.add('line', [.0002, .0005, .00035], dots_size=4)
chart.add('line', [.0004, .0009, .001], dots_size=12)
chart.render()
```

### stroke\_style

```
chart = pygal.Line()
chart.add('line', [.0002, .0005, .00035], stroke_style={'width': 5, 'dasharray': '3, 6', 'linecap':
chart.add('line', [.0004, .0009, .001], stroke_style={'width': 2, 'dasharray': '3, 6, 12, 24'})
chart.render()
```

### rounded\_bars

```
chart = pygal.Bar()
for i in range(10):
    chart.add(str(i), i, rounded_bars=2 * i)
chart.render()
```

### inner\_radius



```
chart = pygal.Pie()
for i in range(10):
    chart.add(str(i), i, inner_radius=(10 - i) / 10)
chart.render()
```

### allow\_interruptions

You can set `allow_interruptions` to True in order to break lines on None values.

```
interrupted_chart = pygal.Line()
interrupted_chart.add(
    'Temperature', [22, 34, 43, 12, None, 12, 55, None, 56],
    allow_interruptions=True)
interrupted_chart.add(
    'Temperature', [11, 17, 21.5, 6, None, 6, 27.5, None, 28])
interrupted_chart.render()
```

### formatter

You can add a *formatter* function for this serie values. It will be used for value printing and tooltip. (Not for axis.)

```
chart = pygal.Bar(print_values=True, value_formatter=lambda x: '{}$'.format(x))
chart.add('bar', [.0002, .0005, .00035], formatter=lambda x: '<{%s}> % x)
chart.add('bar', [.0004, .0009, .001])
chart.render()
```

## 3.1.6 Value configuration

### How

Values are customized by replacing the value with a dictionary containing the value as 'value':

```
chart = pygal.Line()
chart.add('', [1, {'value': 2, 'label': 'two'}, 3])
chart.add('', [3, 2, 1])
```

### Labels

You can add per value metadata like labels, by specifying a dictionary instead of a value:

```
chart = pygal.Bar()
chart.add('First', [{'value': 2, 'label': 'This is the first'}])
chart.add('Second', [{'value': 4, 'label': 'This is the second'}])
chart.add('Third', 7)
chart.add('Fourth', [{'value': 5}])
chart.add('Fifth', [{'value': 3, 'label': 'This is the fifth'}])
chart.render()
```

### Style

You can force the color of a value by specifying a color key:

```
chart = pygal.Bar()
chart.add('Serie', [
    {'value': 2}, 3, 4,
    {'value': 10, 'color': 'red'},
    {'value': 11, 'color': 'rgba(255, 45, 20, .6)'}, 4, 2
])
chart.render()
```

The color key set the fill and the stroke style. You can also set the css style manually:

```
chart = pygal.Bar()
chart.add('Serie', [
    {'value': 2}, 3, 4,
    {'value': 10, 'style': 'fill: red; stroke: black; stroke-width: 4'},
    {'value': 11, 'style': 'fill: rgba(255, 45, 20, .6); stroke: black; stroke-dasharray: 15, 10, 5, 10, 4, 2
4, 2
])
chart.render()
```

### Value formatting

You can add a *formatter* metadata for a specific value.

```
chart = pygal.Bar(print_values=True, value_formatter=lambda x: '{}$'.format(x))
chart.add('bar', [.0002, .0005, .00035], formatter=lambda x: '<%s>' % x)
chart.add('bar', [.0004, {'value': .0009, 'formatter': lambda x: '<<%s>' % x}, .001])
chart.render()
```

### Node attributes

It is possible to pass svg attribute to the node representing value.

```
chart = pygal.Line()
chart.add('Serie', [
    {'value': 1, 'node': {'r': 2}},
    {'value': 2, 'node': {'r': 4}},
    {'value': 3, 'node': {'r': 6}},
    {'value': 4, 'node': {'r': 8}}
])
chart.render()
```

### Links

#### Basic

You can also add hyper links:

```
chart = pygal.Bar()
chart.add('First', [{
    'value': 2,
    'label': 'This is the first',
    'xlink': 'http://en.wikipedia.org/wiki/First'}])

chart.add('Second', [{
```

```

    'value': 4,
    'label': 'This is the second',
    'xlink': 'http://en.wikipedia.org/wiki/Second'])

chart.add('Third', 7)

chart.add('Fourth', [{
    'value': 5,
    'xlink': 'http://en.wikipedia.org/wiki/Fourth'}])

chart.add('Fifth', [{
    'value': 3,
    'label': 'This is the fifth',
    'xlink': 'http://en.wikipedia.org/wiki/Fifth'}])
chart.render()

```

## Advanced

You can specify a dictionary to xlink with all links attributes:

```

chart = pygal.Bar()
chart.add('First', [{
    'value': 2,
    'label': 'This is the first',
    'xlink': {'href': 'http://en.wikipedia.org/wiki/First'}}])

chart.add('Second', [{
    'value': 4,
    'label': 'This is the second',
    'xlink': {
        'href': 'http://en.wikipedia.org/wiki/Second',
        'target': '_top'}
    })

chart.add('Third', 7)

chart.add('Fourth', [{
    'value': 5,
    'xlink': {
        'href': 'http://en.wikipedia.org/wiki/Fourth',
        'target': '_blank'}
    })

chart.add('Fifth', [{
    'value': 3,
    'label': 'This is the fifth',
    'xlink': {
        'href': 'http://en.wikipedia.org/wiki/Fifth',
        'target': '_self'}
    })
chart.render()

```

## Legend

Finally legends can be link with the same mechanism:

```
chart = pygal.Bar()
chart.add({
    'title': 'First',
    'tooltip': 'It is the first actually',
    'xlink': {'href': 'http://en.wikipedia.org/wiki/First'}
}, [{
    'value': 2,
    'label': 'This is the first',
    'xlink': {'href': 'http://en.wikipedia.org/wiki/First'}
}])

chart.add({
    'title': 'Second',
    'xlink': {
        'href': 'http://en.wikipedia.org/wiki/Second',
        'target': '_top'
    }
}, [{
    'value': 4,
    'label': 'This is the second',
    'xlink': {
        'href': 'http://en.wikipedia.org/wiki/Second',
        'target': '_top'
    }
}])

chart.add('Third', 7)

chart.add({
    'title': 'Fourth',
    'xlink': {
        'href': 'http://en.wikipedia.org/wiki/Fourth',
        'target': '_blank'
    }
}, [{
    'value': 5,
    'xlink': {
        'href': 'http://en.wikipedia.org/wiki/Fourth',
        'target': '_blank'
    }
}])

chart.add({
    'title': 'Fifth',
    'xlink': {
        'href': 'http://en.wikipedia.org/wiki/Fifth',
        'target': '_self'
    }
}, [{
    'value': 3,
    'label': 'This is the fifth',
    'xlink': {
        'href': 'http://en.wikipedia.org/wiki/Fifth',
        'target': '_self'
    }
}])
chart.render()
```

## Confidence Intervals

```
chart = pygal.Bar(style=pygal.style.styles['default'](ci_colors=(
    'black', 'blue')))
chart.add('First', [{'value': 2, 'ci': {
    'type': 'continuous', 'sample_size': 50, 'stddev': .5, 'confidence': .95}}])
chart.add('Second', [{'value': 4, 'ci': {'low': 2, 'high': 5}}])
chart.add('Third', 7)
chart.add('Fourth', [{'value': 5}])
chart.add('Fifth', [{'value': 3, 'ci': {
    'type': 'dichotomous', 'sample_size': 1000}}])
chart.render()
```

### 3.1.7 Sparklines

pygal provides a simple way to get beautiful sparklines.

#### Basic

```
chart = pygal.Line()
chart.add('', [1, 3, 5, 16, 13, 3, 7])
chart.render_sparkline()
```

#### Options

Sparklines support the same options as normal charts but for those that are overridden by sparkline settings, pass them to the `render_sparkline` method:

```
chart = pygal.Line(interpolate='cubic')
chart.add('', [1, 3, 5, 16, 13, 3, 7])
chart.render_sparkline()
```

```
from pygal.style import LightSolarizedStyle
chart = pygal.Line(style=LightSolarizedStyle)
chart.add('', [1, 3, 5, 16, 13, 3, 7, 9, 2, 1, 4, 9, 12, 10, 12, 16, 14, 12, 7, 2])
chart.render_sparkline(width=500, height=25, show_dots=True)
```

With labels:

```
chart = pygal.Line()
chart.add('', [1, 3, 5, 16, 13, 3, 7])
chart.x_labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
chart.render_sparkline(show_x_labels=True, show_y_labels=True)
```

#### Sparktext

If you want to get a simple sparktext, use the `render_sparktext` function:

```
chart = pygal.Line()
chart.add('', [1, 3, 5, 16, 13, 3, 7])
chart.render_sparktext()
```

→

You can also specify an explicit minimum for the values:

```
chart = pygal.Line()
chart.add(' ', [1, 3, 5, 16, 13, 3, 7])
chart.render_sparktext(relative_to=0)
```

→

### 3.1.8 Table

pygal also supports a html table export of given data using the `render_table` option:

```
line_chart = pygal.Bar()
line_chart.title = 'Browser usage evolution (in %)'
line_chart.x_labels = map(str, range(2002, 2013))
line_chart.add('Firefox', [None, None, 0, 16.6, 25, 31, 36.4, 45.5, 46.3, 42.8, 37.1])
line_chart.add('Chrome', [None, None, None, None, None, None, 0, 3.9, 10.8, 23.8, 35.3])
line_chart.add('IE', [85.8, 84.6, 84.7, 74.5, 66, 58.6, 54.7, 44.8, 36.2, 26.6, 20.1])
line_chart.add('Others', [14.2, 15.4, 15.3, 8.9, 9, 10.4, 8.9, 5.8, 6.7, 6.8, 7.5])
line_chart.value_formatter = lambda x: '%.2f%%' % x if x is not None else ''
line_chart.render()
```

#### Default

```
line_chart = pygal.Bar()
line_chart.title = 'Browser usage evolution (in %)'
line_chart.x_labels = map(str, range(2002, 2013))
line_chart.add('Firefox', [None, None, 0, 16.6, 25, 31, 36.4, 45.5, 46.3, 42.8, 37.1])
line_chart.add('Chrome', [None, None, None, None, None, None, 0, 3.9, 10.8, 23.8, 35.3])
line_chart.add('IE', [85.8, 84.6, 84.7, 74.5, 66, 58.6, 54.7, 44.8, 36.2, 26.6, 20.1])
line_chart.add('Others', [14.2, 15.4, 15.3, 8.9, 9, 10.4, 8.9, 5.8, 6.7, 6.8, 7.5])
line_chart.value_formatter = lambda x: '%.2f%%' % x if x is not None else ''
line_chart.render_table()
```

#### Style

```
line_chart = pygal.Bar()
line_chart.title = 'Browser usage evolution (in %)'
line_chart.x_labels = map(str, range(2002, 2013))
line_chart.add('Firefox', [None, None, 0, 16.6, 25, 31, 36.4, 45.5, 46.3, 42.8, 37.1])
line_chart.add('Chrome', [None, None, None, None, None, None, 0, 3.9, 10.8, 23.8, 35.3])
line_chart.add('IE', [85.8, 84.6, 84.7, 74.5, 66, 58.6, 54.7, 44.8, 36.2, 26.6, 20.1])
line_chart.add('Others', [14.2, 15.4, 15.3, 8.9, 9, 10.4, 8.9, 5.8, 6.7, 6.8, 7.5])
line_chart.value_formatter = lambda x: '%.2f%%' % x if x is not None else ''
line_chart.render_table(style=True)
```

#### Total

```
line_chart = pygal.Bar()
line_chart.title = 'Browser usage evolution (in %)'
line_chart.x_labels = map(str, range(2002, 2013))
line_chart.add('Firefox', [None, None, 0, 16.6, 25, 31, 36.4, 45.5, 46.3, 42.8, 37.1])
```

```

line_chart.add('Chrome', [None, None, None, None, None, None, 0, 3.9, 10.8, 23.8, 35.3])
line_chart.add('IE', [85.8, 84.6, 84.7, 74.5, 66, 58.6, 54.7, 44.8, 36.2, 26.6, 20.1])
line_chart.add('Others', [14.2, 15.4, 15.3, 8.9, 9, 10.4, 8.9, 5.8, 6.7, 6.8, 7.5])
line_chart.value_formatter = lambda x: '%.2f%%' % x if x is not None else ''
line_chart.render_table(style=True, total=True)

```

## Transposed

```

line_chart = pygal.Bar()
line_chart.title = 'Browser usage evolution (in %)'
line_chart.x_labels = map(str, range(2002, 2013))
line_chart.add('Firefox', [None, None, 0, 16.6, 25, 31, 36.4, 45.5, 46.3, 42.8, 37.1])
line_chart.add('Chrome', [None, None, None, None, None, None, 0, 3.9, 10.8, 23.8, 35.3])
line_chart.add('IE', [85.8, 84.6, 84.7, 74.5, 66, 58.6, 54.7, 44.8, 36.2, 26.6, 20.1])
line_chart.add('Others', [14.2, 15.4, 15.3, 8.9, 9, 10.4, 8.9, 5.8, 6.7, 6.8, 7.5])
line_chart.value_formatter = lambda x: '%.2f%%' % x if x is not None else ''
line_chart.render_table(style=True, total=True, transpose=True)

```

## 3.1.9 Output

pygal can generate multiple output formats.

### SVG

#### String

The obvious output is the vectorial output in svg format:

```

chart = pygal.Line()
...
chart.render() # Return the svg as bytes

```

It can be rendered as unicode when specifying `is_unicode=True` or when `disable_xml_declaration` is used

```

chart = pygal.Line()
...
chart.render(is_unicode=True) # Return the svg as a unicode string

```

### File

You can also write the chart to a file using `render_to_file`:

```

chart = pygal.Line()
...
chart.render_to_file('/tmp/chart.svg') # Write the chart in the specified file

```

### PNG

With `cairosvg` installed you can directly get the png file using `render_to_png`:

```
chart = pygal.Line()
...
chart.render_to_png('/tmp/chart.png') # Write the chart in the specified file
```

In case of rendered image turning up black, installing lxml, tinycss and cssselect should fix the issue.

## Etree

It is possible to get the xml etree root element of the chart (or lxml etree node if lxml is installed) by calling the `render_tree` method:

```
chart = pygal.Line()
...
chart.render_tree() # Return the svg root etree node
```

## Base 64 data URI

You can directly output a base 64 encoded data uri for `<embed>` or `<image>` inclusion:

```
chart = pygal.Line()
...
chart.render_data_uri() # Return `data:image/svg+xml;charset=utf-8;base64,...`
```

## Browser

With lxml installed you can use the `render_in_browser` method to magically make your chart appear in you default browser.

```
chart = pygal.Line()
...
chart.render_in_browser()
```

## PyQuery

If pyquery is installed you can get the pyquery object wrapping the chart by calling `render_pyquery`:

(This is mainly used for testing)

```
chart = pygal.Line()
...
chart.render_pyquery() # Return pyquery object
```

## Flask App

If you are using pygal in a flask app the `render_response` may come in handy:

```
@app.route('/charts/line.svg')
def line_route():
    chart = pygal.Line()
    ...
    return chart.render_response()
```



An other way is to use a Base 64 data URI for your flask app.

In python file:

```
@app.route('/charts/')
def line_route():
    chart = pygal.Line()
    ...
    chart = chart.render_data_uri()

    return render_template('charts.html', chart = chart)
```

In HTML file:

```
<!-- Don't forget the "|safe"! -->
<div id="chart">
    <embed type="image/svg+xml" src= {{ chart|safe }} />
</div>
```

## Django response

Same thing for django with `render_django_response`.

### 3.1.10 Embedding in a web page

#### Within an embed tag

First set up an url entry point for your svg: `/mysvg.svg` don't forget to set the mime-type to `image/svg+xml`. (If you are using flask you can use the `render_response` method.)

Then in your html put an embed tag like this:

```
<!DOCTYPE html>
<html>
  <head>
    <!-- ... -->
  </head>
  <body>
    <figure>
      <embed type="image/svg+xml" src="/mysvg.svg" />
    </figure>
  </body>
</html>
```

You can also use an `iframe` tag, but automatic sizing with `width: 100%` will not work.

#### Directly in the html

You can insert it directly in a html page with the use of `disable_xml_declaration`. You have to put the javascript manually in you webpage, for instance:

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript" src="http://kozea.github.com/pygal.js/latest/pygal-tooltips.min.js">
    <!-- ... -->
```

```
</head>
<body>
  <figure>
    <!-- Pygal render() result: -->
    <svg
      xmlns:xlink="http://www.w3.org/1999/xlink"
      xmlns="http://www.w3.org/2000/svg"
      id="chart-e6700c90-7a2b-4602-961c-83ccf5e59204"
      class="pygal-chart "
      viewBox="0 0 800 600">
      <!--Generated with pygal 1.0.0 @Kozea 2011-2013 on 2013-06-25-->
      <!--http://pygal.org-->
      <!--http://github.com/Kozea/pygal-->
      <defs>
        <!-- ... -->
      </defs>
      <title>Pygal</title>
      <g class="graph bar-graph vertical">
        <!-- ... -->
      </g>
    </svg>
    <!-- End of Pygal render() result: -->
  </figure>
</body>
</html>
```

You can use `explicit_size` to set the svg size from the width, height properties.

## 3.2 Installing

pygal is available for python 2.7 and 3.2, 3.3, 3.4, 3.5 and pypy.

### 3.2.1 PyPI

pygal is available on PyPI. To install, just type as superuser:

```
pip install pygal
```

### 3.2.2 Dependencies

There are no required dependency.

Optional dependencies are as follow:

- `lxml` which can improve rendering speed (except on pypy).
- `cairosvg`, `tinycss`, `cssselect` to render png.

### 3.2.3 Git Repository

If you want the development version of pygal, take a look at the [git repository on GitHub](#), or clone it with:

```
git clone git://github.com/Kozea/pygal.git
```

You can also download the development snapshot from [github](#).

## 3.2.4 Linux Distribution Packages

Pygal has been packaged for:

- [Fedora](#)
- [Gentoo](#)
- [Ubuntu](#)
- [Debian](#)
- [Arch Linux](#)

If you are interested in creating packages for Linux distributions, contact us.

## 3.3 Contributing

### 3.3.1 Github

Submit your bug reports and feature requests to the [github bug tracker](#).

### 3.3.2 Code style

The pygal code tries to respect the [pep8](#) please keep that in mind when writing code for pygal. (The code style is checked along with the unit tests, see next paragraph).

### 3.3.3 Testing

Before submitting a pull request, please check that all tests still pass.

To do this install `py.test` and then run `py.test` in the root of your pygal clone:

```
[dev@dev pygal/]$ py.test --flake8
```

Even better if you have several python versions installed you can run `tox`.

### 3.3.4 Continuous Integration

The current build status can be seen at our [ymci](#)

## 3.4 Changelog

### 3.4.1 2.3.0

- New call API: `chart = Line(fill=True); chart.add('title', [1, 3, 12]); chart.render()` can now be replaced with `Line(fill=True)(1, 3, 12, title='title').render()`

- Drop python 2.6 support

### 3.4.2 2.2.3

- Fix bar static value positioning (#315)
- Add `stroke_opacity` style (#321)
- Remove useless js in sparklines. (#312)

### 3.4.3 2.2.2

- Add `classes` option.
- Handle ellipsis in list type configs to auto-extend parent. (Viva python3)

### 3.4.4 2.2.0

- Support interruptions in line charts (thanks @piotrmaslanka #300)
- Fix confidence interval reactiveness (thanks @chartique #296)
- Add horizontal line charts (thanks @chartique #301)
- There is now a `formatter` config option to format values as specified. The formatter callable may or may not take `chart`, `serie` and `index` as argument. The default value formatting is now chart dependent and is `value_formatter` for most graph but could be a combination of `value_formatter` and `x_value_formatter` for dual charts.
- The `human_readable` option has been removed. Now you have to use the `pygal.formatters.human_readable` formatter (`value_formatter=human_readable` instead of `human_readable=True`)
- New chart type: `SolidGauge` (thanks @chartique #295)
- Fix range option for some Charts (#297 #298)
- Fix timezones for `DateTimeLine` for python 2 (#306, #302)
- Set default uri protocol to https (should fix a lot of “no tooltips” bugs).

### 3.4.5 2.1.1

- Import `scipy` as a last resort in `stats.py` (should workaround bugs like #294 if `scipy` is installed but not used)

### 3.4.6 2.1.0

- Bar print value positioning with `print_values_position`. Can be `top`, `center` or `bottom` (thanks @chartique #291) ci doc
- Confidence intervals (thanks @chartique #292) data doc

### 3.4.7 2.0.12

- Use custom `xml_declaration` avoiding conflict with processing instructions

### 3.4.8 2.0.11

- lxml 3.5 compatibility (#282)

### 3.4.9 2.0.10

- Fix `transposable_node` in case all attributes are not there. (thanks @yobuntu).

### 3.4.10 2.0.9

- Add `dynamic_print_values` to show `print_values` on legend hover. (#279)
- Fix `unparse_color` for python 3.5+ compatibility (thanks @felixonmars, @sjourdois)
- Process major labels as labels. (#263)
- Fix labels rotation > 180 (#257)
- Fix secondary axis
- Don't forget secondary series in table rendering (#260)
- Add `defs` config option to allow adding gradients and patterns.

### 3.4.11 2.0.8

- Fix value overwrite in map. (#275)

### 3.4.12 2.0.7

- Fixing to checks breaking rendering of `DateTimeline` and `TimeDeltaLine` (#264) (thanks @mmrose)
- Fix `render_in_browser`. (#266) (#268) (thanks @waixwong)

### 3.4.13 2.0.6

- Avoid x label formatting when label is a string

### 3.4.14 2.0.5

- Fix x label formatting

### 3.4.15 2.0.4

- Fix map coloration

### 3.4.16 2.0.3

- Fix label adaptation. (#256)
- Fix wrong radar truncation. (#255)

### 3.4.17 2.0.2

- Fix view box differently to avoid getting a null height on huge numbers. (#254)
- Fix broken font\_family default
- Fix non namespaced svg (without embed) javascript by adding uuid in config object. (config is in window.pygal now).

### 3.4.18 2.0.1

- Fix the missing title on x\_labels with labels.
- Auto cast to str x labels in non dual charts (#178)
- Add print\_labels option to print label too. (#197)
- Add value\_label\_font\_family and value\_label\_font\_size style options for print\_labels.
- Default print\_zeroes to True
- (Re)Add xlink in desc to show on tooltip
- Activate element on tooltip hovering. (#106)
- Fix radar axis behaviour (#247)
- Add tooltip support in metadata to add a title (#249).
- Take config class options in account too.

### 3.4.19 2.0.0

- Rework the ghost mechanism to come back to a more object oriented behavior, storing all state in a state object which is created on every render. (#161)
- Refactor maps
- Add world continents
- Add swiss cantons map (thanks @sergedroz)
- Add inverse\_y\_axis options to reverse graph (#24)
- Fix DateTimeLine time data loss (#193)
- Fix no data for graphs with only zeroes (#148)
- Support value formatter for pie graphs (#218) (thanks @never-eat-yellow-snow)
- Add new Box plot modes and outliers and set extremes as default (#226 #121 #149) (thanks @djezar)
- Add secondary\_range option to set range for secondary values. (#203)
- Maps are now plugins, they are removed from pygal core and moved to packages (pygal\_maps\_world, pygal\_maps\_fr, pygal\_maps\_ch, ...) (#225)
- Dot now supports negative values
- Fix dot with log scale (#201)
- Fix y\_labels behaviour for lines
- Fix x\_labels and y\_labels behaviour for xy like

- Improve gauge a bit
- Finally allow call chains on add
- Transform `min_scale` and `max_scale` as options
- `mode` option has been renamed to a less generic name: `box_mode`
- fix `stack_from_top` for stacked lines
- Add flake8 test to `py.test` in `tox`
- Remove `stroke_style` in `style` and set it as a global / serie configuration.
- Fix `None` values in tables
- Fix timezones in `DateTimeLine`
- Rename in `Style` `foreground_light` as `foreground_strong`
- Rename in `Style` `foreground_dark` as `foreground_subtle`
- Add a `render_data_uri` method (#237)
- Move `font_size` config to `style`
- Add `font_family` for various elements in `style`
- Add `googlefont : font` support for `style` fonts
- Add `tooltip_fancy_mode` to revert to old tooltips
- Add `auto_print_value_color` + a configurable `value_colors` list in `style`
- Add `guide_stroke_dasharray` and `guide_stroke_dasharray` in `style` to customize guides (#242) (thanks @cbergmiller)
- Refactor label processing in a `_compute_x_labels` and `_compute_y_labels` method. Handle both string and numbers for all charts. Create a `Dual` base chart for dual axis charts. (#236)
- Better js integration in maps. Use the normal tooltip.

### 3.4.20 1.7.0

- Remove `DateY` and replace it by real XY datetime, date, time and timedelta support. (#188)
- Introduce new XY configuration options: `xrange`, `x_value_formatter`.
- Add `show_x_labels` option to remove them and the x axis.
- Set `print_values` to `False` by default.
- Fix secondary serie text values when `None` in data. (#192)

### 3.4.21 1.6.2

- Add `margin_top`, `margin_right`, `margin_bottom`, `margin_left` options which defaults to `margin`. (thanks @djt)
- Update `django_mime` parameter from `mimetype` to `content_type`. (thanks @kswiat)
- Allow a color and a style parameter to value metadata.

### 3.4.22 1.6.1

- Fix Decimal incompatibility

### 3.4.23 1.6.0

- Adds config option `missing_value_fill_truncation`. (thanks @sirlark)
- Avoid HTTP 301 Moved Permanently (thanks @jean)
- Add a Django response method (thanks @inlanger)
- Fix `setup.py` (#170)
- Fix format error on list like in table
- Add `legend_at_bottom_columns` option to specify number of columns in legend when at bottom. (#157)
- Fix secondary interpolation (#165)
- Adds an extra class (`axis`) to horizontal guides if the label is “0” (#147) (thanks @sirlark)
- Add line stroke customization parameters to `style.py` (#154) (thanks @blakev)

### 3.4.24 1.5.1

- Add `stack_from_top` option to reverse stack graph data order
- Minor fix for empty logarithmic chart
- Reorders axes in SVG output. Fix #145 (thanks @sirlark)

### 3.4.25 1.5.0

- Add per serie configuration
- Add half pie (thanks @philt2001)
- Make `lxml` an optionnal dependency (huge speed boost in pypy)
- Add `render_table` (WIP)
- Support colors in `rgb / rgba` for parametric styles

### 3.4.26 1.4.6

- Add support for n separated multiline titles (thanks @sirlark)
- New `show_only_major_dots` option (thanks @Le-Stagiaire)
- Remove 16 colors limitation
- Fix 0 in range (thanks @elpaso)

### 3.4.27 1.4.5

- Fix `y_labels` map iterator exhaustion in python 3



### 3.4.28 1.4.4

- Fix division by zero in spark text (thanks @laserpony)
- Fix config metaclass problem in python 3
- Fix `-version` in `pygal_gen`

### 3.4.29 1.4.3

- Allow arbitrary number of x-labels on line plot (thanks @nsmgr8)

### 3.4.30 1.4.2

- Fix broken tests

### 3.4.31 1.4.1

- Fix value formatting in maps

### 3.4.32 1.4.0

- Finally a changelog !
- Hopefully fix weird major scale algorithm
- Add options to customize major labels (`y_labels_major`, `y_labels_major_every`, `y_labels_major_count`)
- Css can now be inline with the “inline:” prefix
- Visited links bug fixed
- Add french maps by department and region (This will be externalized in an extension later)

### 3.4.33 1.3.x

- Whisker Box Plot
- Python 3 fix
- DateY X axis formatting (`x_label_format`)

## 3.5 API

### 3.5.1 pygal package

Main pygal package.

This package holds all available charts in pygal, the Config class and the maps extensions namespace module.

**class** `pygal.PluginImportFixer`

Bases: `object`

Allow external map plugins to be imported from `pygal.maps` package.

It is a `sys.meta_path` loader.

**find\_module** (*fullname*, *path=None*)

Tell if the module to load can be loaded by the `load_module` function, ie: if it is a `pygal.maps.*` module.

**load\_module** (*name*)

Load the `pygal.maps.name` module from the previously loaded plugin

## Subpackages

### `pygal.graph` package

Graph package containing all builtin charts

## Submodules

**pygal.graph.bar module** Bar chart that presents grouped data with rectangular bars with lengths proportional to the values that they represent.

**class** `pygal.graph.bar.Bar` (*config=None*, *\*\*kwargs*)

Bases: `pygal.graph.graph.Graph`

Bar graph class

**bar** (*serie*, *rescale=False*)

Draw a bar graph for a serie

**pygal.graph.base module** Base for pygal charts

**class** `pygal.graph.base.BaseGraph` (*config=None*, *\*\*kwargs*)

Bases: `object`

Chart internal behaviour related functions

**prepare\_values** (*raw*, *offset=0*)

Prepare the values to start with sane values

**setup** (*\*\*kwargs*)

Set up the transient state prior rendering

**teardown** ()

Remove the transient state after rendering

**pygal.graph.box module** Box plot: a convenient way to display series as box with whiskers and outliers Different types are available throught the `box_mode` option

**class** `pygal.graph.box.Box` (*config=None*, *\*\*kwargs*)

Bases: `pygal.graph.graph.Graph`

Box plot For each series, shows the median value, the 25th and 75th percentiles, and the values within 1.5 times the interquartile range of the 25th and 75th percentiles.

See [http://en.wikipedia.org/wiki/Box\\_plot](http://en.wikipedia.org/wiki/Box_plot)

**pygal.graph.dot module** Dot chart displaying values as a grid of dots, the bigger the value the bigger the dot

**class** `pygal.graph.dot.Dot` (*config=None, \*\*kwargs*)

Bases: `pygal.graph.graph.Graph`

Dot graph class

**dot** (*serie, r\_max*)

Draw a dot line

**pygal.graph.dual module** Dual chart base. Dual means a chart with 2 scaled axis like xy

**class** `pygal.graph.dual.Dual` (*config=None, \*\*kwargs*)

Bases: `pygal.graph.graph.Graph`

**pygal.graph.funnel module** Funnel chart: Represent values as a funnel

**class** `pygal.graph.funnel.Funnel` (*config=None, \*\*kwargs*)

Bases: `pygal.graph.graph.Graph`

Funnel graph class

**funnel** (*serie*)

Draw a funnel slice

**pygal.graph.gauge module** Gauge chart representing values as needles on a polar scale

**class** `pygal.graph.gauge.Gauge` (*config=None, \*\*kwargs*)

Bases: `pygal.graph.graph.Graph`

Gauge graph class

**needle** (*serie*)

Draw a needle for each value

**needle\_width = 0.05**

**pygal.graph.graph module** Chart properties and drawing

**class** `pygal.graph.graph.Graph` (*config=None, \*\*kwargs*)

Bases: `pygal.graph.public.PublicApi`

Graph super class containing generic common functions

**add\_squares** (*squares*)

**all\_series**

Getter for all series (nomal and secondary)

**pygal.graph.histogram module** Histogram chart: like a bar chart but with data plotted along a x axis as bars of varying width.

**class** `pygal.graph.histogram.Histogram` (*config=None, \*\*kwargs*)

Bases: `pygal.graph.dual.Dual, pygal.graph.bar.Bar`

Histogram chart class

**bar** (*serie, rescale=False*)  
Draw a bar graph for a serie

**xvals**  
All x values

**yvals**  
All y values

**pygal.graph.horizontal module** Horizontal graph mixin

**class** `pygal.graph.horizontal.HorizontalGraph` (*\*args, \*\*kwargs*)  
Bases: `pygal.graph.graph.Graph`  
Horizontal graph mixin

**pygal.graph.horizontalbar module** Horizontal bar graph

**class** `pygal.graph.horizontalbar.HorizontalBar` (*\*args, \*\*kwargs*)  
Bases: `pygal.graph.horizontal.HorizontalGraph`, `pygal.graph.bar.Bar`  
Horizontal Bar graph

**pygal.graph.horizontalline module** Horizontal line graph

**class** `pygal.graph.horizontalline.HorizontalLine` (*\*args, \*\*kwargs*)  
Bases: `pygal.graph.horizontal.HorizontalGraph`, `pygal.graph.line.Line`  
Horizontal Line graph

**pygal.graph.horizontalstackedbar module** Horizontal stacked graph

**class** `pygal.graph.horizontalstackedbar.HorizontalStackedBar` (*\*args, \*\*kwargs*)  
Bases: `pygal.graph.horizontal.HorizontalGraph`, `pygal.graph.stackedbar.StackedBar`  
Horizontal Stacked Bar graph

**pygal.graph.horizontalstackedline module** Horizontal Stacked Line graph

**class** `pygal.graph.horizontalstackedline.HorizontalStackedLine` (*\*args, \*\*kwargs*)  
Bases: `pygal.graph.horizontal.HorizontalGraph`, `pygal.graph.stackedline.StackedLine`  
Horizontal Stacked Line graph

**pygal.graph.line module** Line chart: Display series of data as markers (dots) connected by straight segments

**class** `pygal.graph.line.Line` (*\*args, \*\*kwargs*)  
Bases: `pygal.graph.graph.Graph`  
Line graph class

**line** (*serie, rescale=False*)  
Draw the line serie

**pygal.graph.map module** pygal contains no map but a base class to create extension see the pygal\_maps\_world package to get an exemple. [https://github.com/Kozea/pygal\\_maps\\_world](https://github.com/Kozea/pygal_maps_world)

**class** `pygal.graph.map.BaseMap` (*config=None, \*\*kwargs*)

Bases: `pygal.graph.graph.Graph`

Base class for maps

**adapt\_code** (*area\_code*)

Hook to change the area code

**enumerate\_values** (*serie*)

Hook to replace default enumeration on values

**pygal.graph.pie module** Pie chart: A circular chart divided into slice to illustrate proportions It can be made as a donut or a half pie.

**class** `pygal.graph.pie.Pie` (*config=None, \*\*kwargs*)

Bases: `pygal.graph.graph.Graph`

Pie graph class

**slice** (*serie, start\_angle, total*)

Make a serie slice

**pygal.graph.public module** pygal public api functions

**class** `pygal.graph.public.PublicApi` (*config=None, \*\*kwargs*)

Bases: `pygal.graph.base.BaseGraph`

Chart public functions

**add** (*title, values, \*\*kwargs*)

Add a serie to this graph, compat api

**add\_xml\_filter** (*callback*)

Add an xml filter for in tree post processing

**render** (*is\_unicode=False, \*\*kwargs*)

Render the graph, and return the svg string

**render\_data\_uri** (*\*\*kwargs*)

Output a base 64 encoded data uri

**render\_django\_response** (*\*\*kwargs*)

Render the graph, and return a Django response

**render\_in\_browser** (*\*\*kwargs*)

Render the graph, open it in your browser with black magic

**render\_pyquery** (*\*\*kwargs*)

Render the graph, and return a pyquery wrapped tree

**render\_response** (*\*\*kwargs*)

Render the graph, and return a Flask response

**render\_sparkline** (*\*\*kwargs*)

Render a sparkline

**render\_sparktext** (*relative\_to=None*)

Make a mini text sparkline from chart

**render\_table** (\*\*kwargs)

Render the data as a html table

**render\_to\_file** (filename, \*\*kwargs)

Render the graph, and write it to filename

**render\_to\_png** (filename=None, dpi=72, \*\*kwargs)

Render the graph, convert it to png and write it to filename

**render\_tree** (\*\*kwargs)

Render the graph, and return (l)xml etree

**pygal.graph.pyramid module** Pyramid chart: Stacked bar chart containing only positive values divided by two axes, generally gender for age pyramid.

**class** pygal.graph.pyramid.**Pyramid** (\*args, \*\*kwargs)

Bases: *pygal.graph.horizontal.HorizontalGraph*, *pygal.graph.pyramid.VerticalPyramid*

Horizontal Pyramid graph class like the one used by age pyramid

**class** pygal.graph.pyramid.**VerticalPyramid** (config=None, \*\*kwargs)

Bases: *pygal.graph.stackedbar.StackedBar*

Vertical Pyramid graph class

**pygal.graph.radar module** Radar chart: As known as kiviart chart or spider chart is a polar line chart useful for multivariate observation.

**class** pygal.graph.radar.**Radar** (\*args, \*\*kwargs)

Bases: *pygal.graph.line.Line*

Rada graph class

**pygal.graph.solidgauge module** Solid Guage For each series a solid guage is shown on the plot area.

**class** pygal.graph.solidgauge.**SolidGauge** (config=None, \*\*kwargs)

Bases: *pygal.graph.graph.Graph*

**gaugify** (serie, squares, sq\_dimensions, current\_square)

**pygal.graph.stackedbar module** Stacked Bar chart: Like a bar chart but with all series stacking on top of the others instead of being displayed side by side.

**class** pygal.graph.stackedbar.**StackedBar** (config=None, \*\*kwargs)

Bases: *pygal.graph.bar.Bar*

Stacked Bar graph class

**pygal.graph.stackedline module** Stacked Line chart: Like a line chart but with all lines stacking on top of the others. Used along fill=True option.

**class** pygal.graph.stackedline.**StackedLine** (\*args, \*\*kwargs)

Bases: *pygal.graph.line.Line*

Stacked Line graph class

**pygal.graph.time module** XY time extensions: handle conversion of date, time, datetime, timedelta into float for xy plot and back to their type for display

**class** `pygal.graph.time.DateLine(*args, **kwargs)`

Bases: `pygal.graph.time.DateTimeLine`

Date abscissa xy graph class

**class** `pygal.graph.time.DateTimeLine(*args, **kwargs)`

Bases: `pygal.graph.xy.XY`

DateTime abscissa xy graph class

**class** `pygal.graph.time.TimeDeltaLine(*args, **kwargs)`

Bases: `pygal.graph.xy.XY`

TimeDelta abscissa xy graph class

**class** `pygal.graph.time.TimeLine(*args, **kwargs)`

Bases: `pygal.graph.time.DateTimeLine`

Time abscissa xy graph class

`pygal.graph.time.date_to_datetime(x)`

Convert a date into a datetime

`pygal.graph.time.datetime_to_time(x)`

Convert a datetime into a time

`pygal.graph.time.datetime_to_timestamp(x)`

Convert a datetime into a utc float timestamp

`pygal.graph.time.seconds_to_time(x)`

Convert a number of second into a time

`pygal.graph.time.time_to_datetime(x)`

Convert a time into a datetime

`pygal.graph.time.time_to_seconds(x)`

Convert a time in a seconds sum

`pygal.graph.time.timedelta_to_seconds(x)`

Convert a timedelta into an amount of seconds

**pygal.graph.treemap module** Treemap chart: Visualize data using nested rectangles

**class** `pygal.graph.treemap.Treemap(config=None, **kwargs)`

Bases: `pygal.graph.graph.Graph`

Treemap graph class

**pygal.graph.xy module** XY Line graph: Plot a set of couple data points (x, y) connected by straight segments.

**class** `pygal.graph.xy.XY(*args, **kwargs)`

Bases: `pygal.graph.line.Line`, `pygal.graph.dual.Dual`

XY Line graph class

**xvals**

All x values

**yvals**

All y values

## pygal.maps package

Maps extensions namespace module

## pygal.test package

Pygal test package

`pygal.test.adapt` (*chart, data*)  
Adapt data to chart type

`pygal.test.get_data` (*i*)  
Return sample test data for an index

`pygal.test.make_data` (*chart, datas*)  
Add sample data to the test chart

## Submodules

**pygal.test.conftest module** pytest fixtures

`pygal.test.conftest.etreefx` (*request*)  
Fixture allowing to test with builtin etree and lxml

`pygal.test.conftest.pytest_generate_tests` (*metafunc*)  
Generate the tests for etree and lxml

**pygal.test.test\_bar module** Bar chart related tests

`pygal.test.test_bar.test_simple_bar` ()  
Simple bar test

**pygal.test.test\_box module** Box chart related tests

`pygal.test.test_box.test_quartiles` ()  
Test box points for the 1.5IQR computation method

`pygal.test.test_box.test_quartiles_min_extremes` ()  
Test box points for the extremes computation method

`pygal.test.test_box.test_quartiles_stdev` ()  
Test box points for the stdev computation method

`pygal.test.test_box.test_quartiles_tukey` ()  
Test box points for the tukey computation method

`pygal.test.test_box.test_simple_box` ()  
Simple box test

**pygal.test.test\_colors module** Color utility functions tests

`pygal.test.test_colors.test_darken` ()  
Test darken color function

`pygal.test.test_colors.test_desaturate` ()  
Test color desaturation function



```
pygal.test.test_colors.test_hsl_to_rgb_part_0()  
    Test hsl to rgb color function  
pygal.test.test_colors.test_hsl_to_rgb_part_1()  
    Test hsl to rgb color function  
pygal.test.test_colors.test_hsl_to_rgb_part_10()  
    Test hsl to rgb color function  
pygal.test.test_colors.test_hsl_to_rgb_part_11()  
    Test hsl to rgb color function  
pygal.test.test_colors.test_hsl_to_rgb_part_12()  
    Test hsl to rgb color function  
pygal.test.test_colors.test_hsl_to_rgb_part_13()  
    Test hsl to rgb color function  
pygal.test.test_colors.test_hsl_to_rgb_part_14()  
    Test hsl to rgb color function  
pygal.test.test_colors.test_hsl_to_rgb_part_15()  
    Test hsl to rgb color function  
pygal.test.test_colors.test_hsl_to_rgb_part_16()  
    Test hsl to rgb color function  
pygal.test.test_colors.test_hsl_to_rgb_part_17()  
    Test hsl to rgb color function  
pygal.test.test_colors.test_hsl_to_rgb_part_18()  
    Test hsl to rgb color function  
pygal.test.test_colors.test_hsl_to_rgb_part_2()  
    Test hsl to rgb color function  
pygal.test.test_colors.test_hsl_to_rgb_part_3()  
    Test hsl to rgb color function  
pygal.test.test_colors.test_hsl_to_rgb_part_4()  
    Test hsl to rgb color function  
pygal.test.test_colors.test_hsl_to_rgb_part_5()  
    Test hsl to rgb color function  
pygal.test.test_colors.test_hsl_to_rgb_part_6()  
    Test hsl to rgb color function  
pygal.test.test_colors.test_hsl_to_rgb_part_7()  
    Test hsl to rgb color function  
pygal.test.test_colors.test_hsl_to_rgb_part_8()  
    Test hsl to rgb color function  
pygal.test.test_colors.test_hsl_to_rgb_part_9()  
    Test hsl to rgb color function  
pygal.test.test_colors.test_lighten()  
    Test lighten color function  
pygal.test.test_colors.test_parse_color()  
    Test color parse function
```

`pygal.test.test_colors.test_rgb_to_hsl_part_0()`  
Test rgb to hsl color function

`pygal.test.test_colors.test_rgb_to_hsl_part_1()`  
Test rgb to hsl color function

`pygal.test.test_colors.test_rgb_to_hsl_part_10()`  
Test rgb to hsl color function

`pygal.test.test_colors.test_rgb_to_hsl_part_11()`  
Test rgb to hsl color function

`pygal.test.test_colors.test_rgb_to_hsl_part_12()`  
Test rgb to hsl color function

`pygal.test.test_colors.test_rgb_to_hsl_part_13()`  
Test rgb to hsl color function

`pygal.test.test_colors.test_rgb_to_hsl_part_14()`  
Test rgb to hsl color function

`pygal.test.test_colors.test_rgb_to_hsl_part_15()`  
Test rgb to hsl color function

`pygal.test.test_colors.test_rgb_to_hsl_part_16()`  
Test rgb to hsl color function

`pygal.test.test_colors.test_rgb_to_hsl_part_17()`  
Test rgb to hsl color function

`pygal.test.test_colors.test_rgb_to_hsl_part_18()`  
Test rgb to hsl color function

`pygal.test.test_colors.test_rgb_to_hsl_part_2()`  
Test rgb to hsl color function

`pygal.test.test_colors.test_rgb_to_hsl_part_3()`  
Test rgb to hsl color function

`pygal.test.test_colors.test_rgb_to_hsl_part_4()`  
Test rgb to hsl color function

`pygal.test.test_colors.test_rgb_to_hsl_part_5()`  
Test rgb to hsl color function

`pygal.test.test_colors.test_rgb_to_hsl_part_6()`  
Test rgb to hsl color function

`pygal.test.test_colors.test_rgb_to_hsl_part_7()`  
Test rgb to hsl color function

`pygal.test.test_colors.test_rgb_to_hsl_part_8()`  
Test rgb to hsl color function

`pygal.test.test_colors.test_rgb_to_hsl_part_9()`  
Test rgb to hsl color function

`pygal.test.test_colors.test_rotate()`  
Test color rotation function

`pygal.test.test_colors.test_saturate()`  
Test color saturation function

---

`pygal.test.test_colors.test_unparse_color()`  
Test color unparse function

**pygal.test.test\_config module** Various config options tested on one chart type or more

`pygal.test.test_config.test_classes(Chart)`  
Test classes option

`pygal.test.test_config.test_config_alterations_class()`  
Assert a config can be changed on config class

`pygal.test.test_config.test_config_alterations_instance()`  
Assert a config can be changed on instance

`pygal.test.test_config.test_config_alterations_kwargs()`  
Assert a config can be changed with keyword args

`pygal.test.test_config.test_config_behaviours()`  
Test that all different way to set config produce same results

`pygal.test.test_config.test_css(Chart)`  
Test css file option

`pygal.test.test_config.test_fill(Chart)`  
Test fill option

`pygal.test.test_config.test_formatters(Chart)`  
Test custom formatters

`pygal.test.test_config.test_human_readable()`  
Test human readable option

`pygal.test.test_config.test_include_x_axis(Chart)`  
Test x axis inclusion option

`pygal.test.test_config.test_inline_css(Chart)`  
Test inline css option

`pygal.test.test_config.test_interpolation(Chart)`  
Test interpolation option

`pygal.test.test_config.test_label_rotation(Chart)`  
Test label rotation option

`pygal.test.test_config.test_legend_at_bottom(Chart)`  
Test legend at bottom option

`pygal.test.test_config.test_logarithmic()`  
Test logarithmic option

`pygal.test.test_config.test_logarithmic_bad_interpolation()`  
Test interpolation option with a logarithmic chart

`pygal.test.test_config.test_logarithmic_big_scale()`  
Test logarithmic option with a large range of value

`pygal.test.test_config.test_logarithmic_small_scale()`  
Test logarithmic with a small range of values

`pygal.test.test_config.test_meta_config()`  
Test config metaclass

`pygal.test.test_config.test_no_data()`  
Test no data and no data text option

`pygal.test.test_config.test_no_data_interpolation(Chart)`  
Test interpolation option with no data

`pygal.test.test_config.test_no_data_with_empty_serie_interpolation(Chart)`  
Test interpolation option with an empty serie

`pygal.test.test_config.test_no_y_labels(Chart)`  
Test no y labels chart

`pygal.test.test_config.test_range(Chart)`  
Test y label major option

`pygal.test.test_config.test_render_data_uri(Chart)`  
Test the render data uri

`pygal.test.test_config.test_show_dots()`  
Test show dots option

`pygal.test.test_config.test_show_legend()`  
Test show legend option

`pygal.test.test_config.test_value_formatter()`  
Test value formatter option

`pygal.test.test_config.test_x_label_major(Chart)`  
Test x label major option

`pygal.test.test_config.test_x_y_title(Chart)`  
Test x title and y title options

`pygal.test.test_config.test_y_label_major(Chart)`  
Test y label major option

**pygal.test.test\_date module** Date related charts tests

`pygal.test.test_date.test_date()`  
Test a simple dateline

`pygal.test.test_date.test_date_labels()`  
Test dateline with xrange

`pygal.test.test_date.test_date_xrange()`  
Test dateline with xrange

`pygal.test.test_date.test_datetime()`  
Test a simple datetimeline

`pygal.test.test_date.test_time()`  
Test a simple timeline

`pygal.test.test_date.test_timedelta()`  
Test a simple timedeltaline

`pygal.test.test_date.test_utc_timestamping()`

**pygal.test.test\_formatters module** Test formatters

`pygal.test.test_formatters.test_human_readable()`  
Test human\_readable formatter

`pygal.test.test_formatters.test_human_readable_custom()`  
Test human\_readable formatter option

`pygal.test.test_formatters.test_significant()`  
Test significant formatter

**pygal.test.test\_graph module** Generate tests for different chart types with different data

`pygal.test.test_graph.test_empty_lists (Chart)`  
Test chart rendering with an empty serie

`pygal.test.test_graph.test_empty_lists_with_nones (Chart)`  
Test chart rendering with a None filled serie

`pygal.test.test_graph.test_ipython_notebook (Chart, datas)`  
Test ipython notebook

`pygal.test.test_graph.test_iterable_types (Chart)`  
Test serie as various iterable

`pygal.test.test_graph.test_labels_with_links (Chart)`  
Test values with links

`pygal.test.test_graph.test_long_title (Chart, datas)`  
Test chart rendering with a long title

`pygal.test.test_graph.test_metadata (Chart)`  
Test metadata values

`pygal.test.test_graph.test_multi_render (Chart, datas)`  
Check that a chart always render the same

`pygal.test.test_graph.test_no_data_with_empty_serie (Chart)`  
Test no data for empty serie

`pygal.test.test_graph.test_no_data_with_empty_series (Chart)`  
Test no data for 2 empty series

`pygal.test.test_graph.test_no_data_with_list_of_none (Chart)`  
Test no data for a None containing serie

`pygal.test.test_graph.test_no_data_with_lists_of_nones (Chart)`  
Test no data for several None containing series

`pygal.test.test_graph.test_no_data_with_no_values (Chart)`  
Test no data

`pygal.test.test_graph.test_no_data_with_no_values_with_include_x_axis (Chart)`  
Test no data and include\_x\_axis

`pygal.test.test_graph.test_no_data_with_none (Chart)`  
Test no data for a None containing serie

`pygal.test.test_graph.test_non_iterable_value (Chart)`  
Test serie as non iterable

`pygal.test.test_graph.test_only_one_value (Chart)`  
Test chart rendering with only one value

`pygal.test.test_graph.test_only_one_value_intrp (Chart)`  
Test interpolated chart rendering with only one value

`pygal.test.test_graph.test_only_one_value_log` (*Chart*)  
Test logarithmic chart rendering with only one value

`pygal.test.test_graph.test_render_to_file` (*Chart, datas*)  
Test in file rendering

`pygal.test.test_graph.test_render_to_png` (*Chart, datas*)  
Test in file png rendering

`pygal.test.test_graph.test_secondary` (*Chart*)  
Test secondary chart

`pygal.test.test_graph.test_sparkline` (*Chart, datas*)  
Test sparkline

`pygal.test.test_graph.test_unicode_labels_decode` (*Chart*)  
Test unicode labels

`pygal.test.test_graph.test_unicode_labels_python2` (*Chart*)  
Test unicode labels in python 2

`pygal.test.test_graph.test_unicode_labels_python3` (*Chart*)  
Test unicode labels in python 3

`pygal.test.test_graph.test_values_by_dict` (*Chart*)  
Test serie as dict

#### **pygal.test.test\_histogram module** Histogram chart related tests

`pygal.test.test_histogram.test_histogram` ()  
Simple histogram test

#### **pygal.test.test\_interpolate module** Interpolations tests

`pygal.test.test_interpolate.test_cubic` (*Chart, datas*)  
Test cubic interpolation

`pygal.test.test_interpolate.test_cubic_prec` (*Chart, datas*)  
Test cubic interpolation precision

`pygal.test.test_interpolate.test_hermite` (*Chart, datas*)  
Test hermite interpolation

`pygal.test.test_interpolate.test_hermite_cardinal` (*Chart, datas*)  
Test hermite cardinal interpolation

`pygal.test.test_interpolate.test_hermite_catmull_rom` (*Chart, datas*)  
Test hermite catmull rom interpolation

`pygal.test.test_interpolate.test_hermite_finite` (*Chart, datas*)  
Test hermite finite difference interpolation

`pygal.test.test_interpolate.test_hermite_kochanek_bartels` (*Chart, datas*)  
Test hermite kochanek bartels interpolation

`pygal.test.test_interpolate.test_lagrange` (*Chart, datas*)  
Test lagrange interpolation

`pygal.test.test_interpolate.test_quadratic` (*Chart, datas*)  
Test quadratic interpolation

---

`pygal.test.test_interpolate.test_trigonometric(Chart, datas)`  
Test trigonometric interpolation

**pygal.test.test\_line module** Line chart related tests

`pygal.test.test_line.test_int_x_labels()`  
Test x\_labels

`pygal.test.test_line.test_line()`  
Another simple line test

`pygal.test.test_line.test_line_secondary()`  
Test line with a secondary serie

`pygal.test.test_line.test_no_dot()`  
Line test with an empty serie

`pygal.test.test_line.test_no_dot_at_all()`  
Line test with no value

`pygal.test.test_line.test_not_equal_x_labels()`  
Test x\_labels

`pygal.test.test_line.test_one_dot()`  
Line test with an unique value

`pygal.test.test_line.test_only_major_dots()`  
Test major dots with specified major labels

`pygal.test.test_line.test_only_major_dots_count()`  
Test major dots with a major label count

`pygal.test.test_line.test_only_major_dots_every()`  
Test major dots

`pygal.test.test_line.test_only_major_dots_no_labels()`  
Test major dots with no labels

`pygal.test.test_line.test_simple_line()`  
Simple line test

**pygal.test.test\_maps module** Map plugins tests are imported here

**pygal.test.test\_pie module** Donut chart related tests

`pygal.test.test_pie.test_donut()`  
Test a donut pie chart

`pygal.test.test_pie.test_half_pie()`  
Test a half pie chart

`pygal.test.test_pie.test_multiseries_donut()`  
Test a donut pie chart with multiserie

**pygal.test.test\_serie\_config module** Test per serie configuration

`pygal.test.test_serie_config.test_global_config()`  
Test global configuration

`pygal.test.test_serie_config.test_no_serie_config()`  
Test per serie no configuration

`pygal.test.test_serie_config.test_serie_config()`  
Test per serie configuration

`pygal.test.test_serie_config.test_serie_precedence_over_global_config()`  
Test that per serie configuration override global configuration

**pygal.test.test\_sparktext module** Test sparktext rendering

`pygal.test.test_sparktext.test_all_sparktext()`  
Test all character sparktext

`pygal.test.test_sparktext.test_another_sparktext()`  
Test that same data produces same sparktext

`pygal.test.test_sparktext.test_basic_sparktext()`  
Test basic sparktext

`pygal.test.test_sparktext.test_negative_and_float_sparktext()`  
Test negative values

`pygal.test.test_sparktext.test_no_data_sparktext()`  
Test no data sparktext

`pygal.test.test_sparktext.test_same_max_and_relative_values_sparktext()`  
Test flat sparktexts

`pygal.test.test_sparktext.test_shifted_sparktext()`  
Test relative\_to option in sparktext

**pygal.test.test\_stacked module** Stacked chart related tests

`pygal.test.test_stacked.test_stacked_line()`  
Test stacked line

`pygal.test.test_stacked.test_stacked_line_interpolate()`  
Test interpolated stacked line

`pygal.test.test_stacked.test_stacked_line_log()`  
Test logarithmic stacked line

`pygal.test.test_stacked.test_stacked_line_reverse()`  
Test stack from top stacked line

**pygal.test.test\_style module** Style related tests

`pygal.test.test_style.test_parametric_styles()`  
Test that no parametric produce the same result

`pygal.test.test_style.test_parametric_styles_with_parameters()`  
Test a parametric style with parameters

**pygal.test.test\_table module** Box chart related tests

`pygal.test.test_table.test_pie_table()`  
Test rendering a table for a pie



**pygal.test.test\_util module** Utility functions tests

`pygal.test.test_util.test_format()`  
Test format function

`pygal.test.test_util.test_majorize()`  
Test majorize function

`pygal.test.test_util.test_mergextend()`  
Test mergextend function

`pygal.test.test_util.test_minify_css()`  
Test css minifier function

`pygal.test.test_util.test_round_to_float()`  
Test round to float function

`pygal.test.test_util.test_round_to_int()`  
Test round to int function

`pygal.test.test_util.test_swap_curly()`  
Test swap curly function

`pygal.test.test_util.test_truncate()`  
Test truncate function

**pygal.test.test\_view module** View related tests

`pygal.test.test_view.test_all_logarithmic(Chart)`  
Test logarithmic view rendering

**pygal.test.test\_xml\_filters module** Xml filter tests

**class** `pygal.test.test_xml_filters.ChangeBarsXMLFilter(a, b)`  
Bases: `object`

xml filter that insert a subplot

`pygal.test.test_xml_filters.test_xml_filters_changeBars()`  
Test the use a xml filter

`pygal.test.test_xml_filters.test_xml_filters_round_trip()`  
Ensure doing nothing does nothing

**pygal.test.utils module** Tests helpers

`pygal.test.utils.texts(i, e)`  
Helper for getting the text of an element

## Submodules

**pygal.adapters module**

Value adapters to use when a chart doesn't accept all value types

`pygal.adapters.decimal_to_float(x)`  
Cast Decimal values to float

`pygal.adapters.none_to_zero(x)`  
Return 0 if value is None

`pygal.adapters.not_zero(x)`  
Return None if value is zero

`pygal.adapters.positive(x)`  
Return zero if value is negative

### pygal.colors module

This package is an utility package oriented on color alteration. This is used by the `pygal.style` package to generate parametric styles.

`pygal.colors.adjust(color, attribute, percent)`  
Adjust an attribute of color by a percent

`pygal.colors.darken(color, percent)`  
Darken a color by decreasing its lightness by percent

`pygal.colors.desaturate(color, percent)`  
Desaturate a color by decreasing its saturation by percent

`pygal.colors.hsl_to_rgb(h, s, l)`  
Convert a color in h, s, l to a color in r, g, b

`pygal.colors.is_foreground_light(color)`  
Determine if the background color need a light or dark foreground color

`pygal.colors.lighten(color, percent)`  
Lighten a color by increasing its lightness by percent

`pygal.colors.normalize_float(f)`  
Round float errors

`pygal.colors.parse_color(color)`  
Take any css color definition and give back a tuple containing the r, g, b, a values along with a type which can be: #rgb, #rgba, #rrggb, #rrggbaa, rgb, rgba

`pygal.colors.rgb_to_hsl(r, g, b)`  
Convert a color in r, g, b to a color in h, s, l

`pygal.colors.rotate(color, percent)`  
Rotate a color by changing its hue value by percent

`pygal.colors.saturate(color, percent)`  
Saturate a color by increasing its saturation by percent

`pygal.colors.unparse_color(r, g, b, a, type)`  
Take the r, g, b, a color values and give back a type css color string. This is the inverse function of `parse_color`

### pygal.config module

Config module holding all options and their default values.

`class pygal.config.BaseConfig(**kwargs)`  
Bases: `pygal.config.ConfigBase`

This class holds the common method for configs.

A config object can be instantiated with keyword arguments and updated on call with keyword arguments.

**copy()**  
Copy this config object into another

**to\_dict()**  
Export a JSON serializable dictionary of the config

**class** `pygal.config.CommonConfig` (*\*\*kwargs*)

Bases: `pygal.config.BaseConfig`

Class holding options used in both chart and serie configuration

**allow\_interruptions** = Type: bool Default: False Break lines on None values

**dots\_size** = Type: float Default: 2.5 Radius of the dots

**fill** = Type: bool Default: False Fill areas under lines

**formatter** = Type: function Default: None A function to convert raw value to strings for this chart or serie Default

**inner\_radius** = Type: float Default: 0 Piechart inner radius (donut), must be <.9

**rounded\_bars** = Type: int Default: None Set this to the desired radius in px (for Bar-like charts)

**show\_dots** = Type: bool Default: True Set to false to remove dots

**show\_only\_major\_dots** = Type: bool Default: False Set to true to show only major dots according to their major

**stroke** = Type: bool Default: True Line dots (set it to false to get a scatter plot)

**stroke\_style** = Type: dict Default: None Stroke style of serie element. This is a dict which can contain a 'width'

**class** `pygal.config.Config` (*\*\*kwargs*)

Bases: `pygal.config.CommonConfig`

Class holding config values

**box\_mode** = Type: str Default: 'extremes' Sets the mode to be used. (Currently only supported on box plot) May be

**classes** = Type: list Default: ('pygal-chart',) Classes of the root svg node <class 'str'>

**css** = Type: list of str Default: ('file://style.css', 'file://graph.css') List of css file It can be any uri from file:///tmp/sty

**defs** = Type: list of str Default: [] Extraneous defs to be inserted in svg Useful for adding gradients / patterns...

**disable\_xml\_declaration** = Type: bool Default: False Don't write xml declaration and return str instead of s

**dynamic\_print\_values** = Type: bool Default: False Show values only on hover

**explicit\_size** = Type: bool Default: False Write width and height attributes

**force\_uri\_protocol** = Type: str Default: 'https' Default uri protocol Default protocol for external files. Can be

**half\_pie** = Type: bool Default: False Create a half-pie chart

**height** = Type: int Default: 600 Graph height

**include\_x\_axis** = Type: bool Default: False Always include x axis

**inner\_radius** = Type: float Default: 0 Piechart inner radius (donut), must be <.9

**interpolate** = Type: str Default: None Interpolation May be cubic or lagrange or quadratic or trigonometric or

**interpolation\_parameters** = Type: dict of int Default: {} Various parameters for parametric interpolations i

**interpolation\_precision** = Type: int Default: 250 Number of interpolated points between two values

**inverse\_y\_axis** = Type: bool Default: False Inverse Y axis direction

**js** = Type: list of str Default: ('//kozea.github.io/pygal.js/2.0.x/pygal-tooltips.min.js',) List of js file It can be any uri

`legend_at_bottom` = Type: bool    Default: False    Set to true to position legend at bottom

`legend_at_bottom_columns` = Type: int    Default: None    Set to true to position legend at bottom

`legend_box_size` = Type: int    Default: 12    Size of legend boxes

`logarithmic` = Type: bool    Default: False    Display values in logarithmic scale

`margin` = Type: int    Default: 20    Margin around chart

`margin_bottom` = Type: int    Default: None    Margin around bottom of chart

`margin_left` = Type: int    Default: None    Margin around left of chart

`margin_right` = Type: int    Default: None    Margin around right of chart

`margin_top` = Type: int    Default: None    Margin around top of chart

`max_scale` = Type: int    Default: 16    Maximum number of scale graduation for auto scaling

`min_scale` = Type: int    Default: 4    Minimum number of scale graduation for auto scaling

`missing_value_fill_truncation` = Type: str    Default: 'x'    Filled series with missing x and/or y values at the end

`no_data_text` = Type: str    Default: 'No data'    Text to display when no data is given

`no_prefix` = Type: bool    Default: False    Don't prefix css

`order_min` = Type: int    Default: None    Minimum order of scale, defaults to None

`pretty_print` = Type: bool    Default: False    Pretty print the svg

`print_labels` = Type: bool    Default: False    Display value labels

`print_values` = Type: bool    Default: False    Display values as text over plot

`print_values_position` = Type: str    Default: 'center'    Customize position of 'print\_values'. (For bars: 'top', 'center', 'bottom')

`print_zeroes` = Type: bool    Default: True    Display zero values as well

`range` = Type: list of int    Default: None    Explicitly specify min and max of values (ie: (0, 100))

`rounded_bars` = Type: int    Default: None    Set this to the desired radius in px

`secondary_range` = Type: list of int    Default: None    Explicitly specify min and max of secondary values (ie: (0, 100))

`show_legend` = Type: bool    Default: True    Set to false to remove legend

`show_minor_x_labels` = Type: bool    Default: True    Set to false to hide x-labels not marked major

`show_minor_y_labels` = Type: bool    Default: True    Set to false to hide y-labels not marked major

`show_x_guides` = Type: bool    Default: False    Set to true to always show x guide lines

`show_x_labels` = Type: bool    Default: True    Set to false to hide x-labels

`show_y_guides` = Type: bool    Default: True    Set to false to hide y guide lines

`show_y_labels` = Type: bool    Default: True    Set to false to hide y-labels

`spacing` = Type: int    Default: 10    Space between titles/legend/axes

`stack_from_top` = Type: bool    Default: False    Stack from top to zero, this makes the stacked data match the legend

`strict` = Type: bool    Default: False    If True don't try to adapt / filter wrong values

`style` = Type: Style    Default: <pygal.style.RotateStyle object at 0x7f812c7e93c8>    Style holding values injected in css

`title` = Type: str    Default: None    Graph title. Leave it to None to disable title.

`tooltip_border_radius` = Type: int    Default: 0    Tooltip border radius

**tooltip\_fancy\_mode** = Type: bool    Default: True    Fancy tooltips Print legend, x label in tooltip and use serie color  
**truncate\_label** = Type: int    Default: None    Label string length truncation threshold None = auto, Negative for no truncation  
**truncate\_legend** = Type: int    Default: None    Legend string length truncation threshold None = auto, Negative for no truncation  
**value\_formatter** = Type: function    Default: <pygal.formatters.Default object at 0x7f812c798cc0>    A function to format values  
**width** = Type: int    Default: 800    Graph width  
**x\_label\_rotation** = Type: int    Default: 0    Specify x labels rotation angles in degrees  
**x\_labels** = Type: list of str    Default: None    X labels, must have same len than data. Leave it to None to disable x labels  
**x\_labels\_major** = Type: list of str    Default: None    X labels that will be marked major.  
**x\_labels\_major\_count** = Type: int    Default: None    Mark n evenly distributed labels as major.  
**x\_labels\_major\_every** = Type: int    Default: None    Mark every n-th x label as major.  
**x\_title** = Type: str    Default: None    Graph X-Axis title. Leave it to None to disable X-Axis title.  
**x\_value\_formatter** = Type: function    Default: <pygal.formatters.Default object at 0x7f812c798cc0>    A function to format x values  
**xrange** = Type: list of int    Default: None    Explicitly specify min and max of x values (used in XY and Date charts) (if None, use data range)  
**y\_label\_rotation** = Type: int    Default: 0    Specify y labels rotation angles in degrees  
**y\_labels** = Type: list of float    Default: None    You can specify explicit y labels Must be a list of numbers  
**y\_labels\_major** = Type: list of str    Default: None    Y labels that will be marked major. Default: auto  
**y\_labels\_major\_count** = Type: int    Default: None    Mark n evenly distributed y labels as major.  
**y\_labels\_major\_every** = Type: int    Default: None    Mark every n-th y label as major.  
**y\_title** = Type: str    Default: None    Graph Y-Axis title. Leave it to None to disable Y-Axis title.  
**zero** = Type: int    Default: 0    Set the ordinate zero value Useful for filling to another base than abscissa

**class** `pygal.config.Key` (*default\_value*, *type\_*, *category*, *doc*, *subdoc*=' ', *subtype*=None)  
 Bases: `object`

Represents a config parameter.

A config parameter has a name, a default value, a type, a category, a documentation, an optional longer documentation and an optional subtype for list style option.

Most of these informations are used in cabaret to auto generate forms representing these options.

**coerce** (*value*)

Cast a string into this key type

**is\_boolean**

Return *True* if this parameter is a boolean

**is\_dict**

Return *True* if this parameter is a mapping

**is\_list**

Return *True* if this parameter is a list

**is\_numeric**

Return *True* if this parameter is numeric (int or float)

**is\_string**

Return *True* if this parameter is a string

**class** `pygal.config.MetaConfig`

Bases: `type`

Config metaclass. Used to get the key name and set it on the value.

**class** `pygal.config.SerieConfig` (*\*\*kwargs*)

Bases: `pygal.config.CommonConfig`

Class holding serie config values

**secondary** = Type: `bool`    Default: `False`    Set it to put the serie in a second axis

**title** = Type: `str`    Default: `None`    Serie title. Leave it to `None` to disable title.

### pygal.etree module

Wrapper for seamless `lxml.etree` / `xml.etree` usage depending on whether `lxml` is installed or not.

**class** `pygal.etree.Etree`

Bases: `object`

Etree wrapper using `lxml.etree` or standard `xml.etree`

**to\_etree** ()

Force `xml.etree` to be used

**to\_lxml** ()

Force `lxml.etree` to be used

### pygal.formatters module

Formatters to use with `value_formatter` and `x_value_formatter` configs

**class** `pygal.formatters.Default` (*precision=10*)

Bases: `pygal.formatters.Significant`, `pygal.formatters.IsoDateTime`,  
`pygal.formatters.Raw`

Try to guess best format from type

**class** `pygal.formatters.Formatter`

Bases: `object`

**class** `pygal.formatters.HumanReadable` (*none\_char=''*)

Bases: `pygal.formatters.Formatter`

Format a number to engineer scale

**ORDERS** = 'yzafpnµm kMGTPPEZY'

**class** `pygal.formatters.Integer`

Bases: `pygal.formatters.Formatter`

Cast number to integer

**class** `pygal.formatters.IsoDateTime`

Bases: `pygal.formatters.Formatter`

Iso format datetimes

**class** `pygal.formatters.Raw`

Bases: `pygal.formatters.Formatter`

Cast everything to string

**class** `pygal.formatters.Significant` (*precision=10*)

Bases: `pygal.formatters.Formatter`

Show precision significant digit of float

## pygal.interpolate module

Interpolation functions

These functions takes two lists of points x and y and returns an iterator over the interpolation between all these points with *precision* interpolated points between each of them

`pygal.interpolate.cubic_interpolate` (*x, y, precision=250, \*\*kwargs*)

Interpolate x, y using a cubic algorithm [https://en.wikipedia.org/wiki/Spline\\_interpolation](https://en.wikipedia.org/wiki/Spline_interpolation)

`pygal.interpolate.hermite_interpolate` (*x, y, precision=250, type='cardinal', c=None, b=None, t=None*)

Interpolate x, y using the hermite method. See [https://en.wikipedia.org/wiki/Cubic\\_Hermite\\_spline](https://en.wikipedia.org/wiki/Cubic_Hermite_spline)

**This interpolation is configurable and contain 4 subtypes:**

- Catmull Rom
- Finite Difference
- Cardinal
- Kochanek Bartels

**The cardinal subtype is customizable with a parameter:**

- c: tension (0, 1)

**This last type is also customizable using 3 parameters:**

- c: continuity (-1, 1)
- b: bias (-1, 1)
- t: tension (-1, 1)

`pygal.interpolate.lagrange_interpolate` (*x, y, precision=250, \*\*kwargs*)

Interpolate x, y using Lagrange polynomials [https://en.wikipedia.org/wiki/Lagrange\\_polynomial](https://en.wikipedia.org/wiki/Lagrange_polynomial)

`pygal.interpolate.quadratic_interpolate` (*x, y, precision=250, \*\*kwargs*)

Interpolate x, y using a quadratic algorithm [https://en.wikipedia.org/wiki/Spline\\_\(mathematics\)](https://en.wikipedia.org/wiki/Spline_(mathematics))

`pygal.interpolate.trigonometric_interpolate` (*x, y, precision=250, \*\*kwargs*)

Interpolate x, y using trigonometric As per [http://en.wikipedia.org/wiki/Trigonometric\\_interpolation](http://en.wikipedia.org/wiki/Trigonometric_interpolation)

## pygal.serie module

Serie property holder

**class** `pygal.serie.Serie` (*index, values, config, metadata=None*)

Bases: `object`

Serie class containing title, values and the graph serie index

**safe\_values**

Property containing all values that are not None

### pygal.state module

Class holding state during render

```
class pygal.state.State (graph, **kwargs)
    Bases: object
```

Class containing config values overridden by chart values overridden by keyword args

### pygal.stats module

```
pygal.stats.confidence_interval_continuous (point_estimate, stdev, sample_size, confidence=0.95, **kwargs)
```

Continuous confidence interval from sample size and standard error

```
pygal.stats.confidence_interval_dichotomous (point_estimate, sample_size, confidence=0.95, bias=False, percentage=True, **kwargs)
```

Dichotomous confidence interval from sample size and maybe a bias

```
pygal.stats.confidence_interval_manual (point_estimate, low, high)
```

```
pygal.stats.erfinv (x, a=0.147)
```

Approximation of the inverse error function [https://en.wikipedia.org/wiki/Error\\_function](https://en.wikipedia.org/wiki/Error_function) #Approximation\_with\_elementary\_functions

```
pygal.stats.norm_ppf (x)
```

```
pygal.stats.ppf (x, n)
```

### pygal.style module

Charts styling classes

```
class pygal.style.BlueStyle (**kwargs)
    Bases: pygal.style.Style
```

A blue style

```
background = '#f0f0f0'
```

```
colors = ('#00b2f0', '#43d9be', '#0662ab', '#00668a', '#98eadb', '#97d959', '#033861', '#ffd541', '#7dcf30', '#3ecdff', '#
```

```
foreground = 'rgba(0, 0, 0, 0.9)'
```

```
foreground_strong = 'rgba(0, 0, 0, 0.9)'
```

```
foreground_subtle = 'rgba(0, 0, 0, 0.6)'
```

```
opacity = '.5'
```

```
opacity_hover = '.9'
```

```
plot_background = '#f8f8f8'
```

```
transition = '250ms ease-in'
```

```
class pygal.style.CleanStyle (**kwargs)
    Bases: pygal.style.Style
```

A rather clean style

```
background = 'transparent'
```



```

colors = ('rgb(12,55,149)', 'rgb(117,38,65)', 'rgb(228,127,0)', 'rgb(159,170,0)', 'rgb(149,12,12)')
foreground = 'rgba(0, 0, 0, 0.9)'
foreground_strong = 'rgba(0, 0, 0, 0.9)'
foreground_subtle = 'rgba(0, 0, 0, 0.5)'
plot_background = 'rgba(240, 240, 240, 0.7)'

```

```
class pygal.style.DarkColorizedStyle (**kwargs)
```

Bases: `pygal.style.Style`

A dark colorized style

```

background = '#2c2230'
colors = ('#c900fe', '#01b8fe', '#59f500', '#ff00e4', '#f9fa00', '#780098', '#0181b2', '#348f00', '#b200a0', '#feff61')
foreground = 'rgba(255, 255, 255, 0.9)'
foreground_strong = 'rgba(255, 255, 255, 0.9)'
foreground_subtle = 'rgba(255, 255, 255, 0.5)'
opacity = '.2'
opacity_hover = '.7'
plot_background = '#3f3145'
transition = '250ms ease-in'

```

```
class pygal.style.DarkGreenBlueStyle (**kwargs)
```

Bases: `pygal.style.Style`

A dark green and blue style

```

background = '#000'
colors = ('#7ed2fa', '#7dcf30', '#247fab', '#64a626', '#2f9ed3', '#97d959', '#1b6081', '#fff')
foreground = 'rgba(255, 255, 255, 0.9)'
foreground_strong = 'rgba(255, 255, 255, 0.9)'
foreground_subtle = 'rgba(255, 255, 255, 0.6)'
opacity = '.55'
opacity_hover = '.9'
plot_background = '#141414'
transition = '250ms ease-in'

```

```
class pygal.style.DarkGreenStyle (**kwargs)
```

Bases: `pygal.style.Style`

A dark green style

```

background = '#161201'
colors = ('#adde09', '#6e8c06', '#4a5e04', '#fcd202', '#C1E34D', '#fee980')
foreground = 'rgba(255, 255, 255, 0.9)'
foreground_strong = 'rgba(255, 255, 255, 0.9)'
foreground_subtle = 'rgba(255, 255, 255, 0.6)'

```

```
opacity = '.6'  
opacity_hover = '.9'  
plot_background = '#201a01'  
transition = '250ms ease-in'
```

```
class pygal.style.DarkSolarizedStyle (**kwargs)  
    Bases: pygal.style.Style
```

Dark solarized popular theme

```
background = '#073642'  
colors = ('#b58900', '#cb4b16', '#dc322f', '#d33682', '#6c71c4', '#268bd2', '#2aa198', '#859900')  
foreground = '#839496'  
foreground_strong = '#fdf6e3'  
foreground_subtle = '#657b83'  
opacity = '.66'  
opacity_hover = '.9'  
plot_background = '#002b36'  
transition = '500ms ease-in'
```

```
class pygal.style.DarkStyle (**kwargs)  
    Bases: pygal.style.Style
```

A dark style (old default)

```
background = 'black'  
colors = ('#ff5995', '#b6e354', '#feed6c', '#8cedff', '#9e6ffe', '#899ca1', '#f8f8f2', '#bf4646', '#516083', '#f92672', '#82b362')  
foreground = '#999'  
foreground_strong = '#eee'  
foreground_subtle = '#555'  
opacity = '.8'  
opacity_hover = '.4'  
plot_background = '#111'  
transition = '250ms'
```

```
class pygal.style.DarkenStyle (color, step=10, max_=None, base_style=None, **kwargs)  
    Bases: pygal.style.ParametricStyleBase
```

Create a style by darkening the given color

```
pygal.style.DefaultStyle  
    alias of Style
```

```
class pygal.style.DesaturateStyle (color, step=10, max_=None, base_style=None, **kwargs)  
    Bases: pygal.style.ParametricStyleBase
```

Create a style by desaturating the given color

```

class pygal.style.LightColorizedStyle (**kwargs)
    Bases: pygal.style.Style
    A light colorized style
    background = '#f8f8f8'
    colors = ('#fe9592', '#534f4c', '#3ac2c0', '#a2a7a1', '#fd4b46', '#7b7571', '#73d5d4', '#c9ccc9', '#ffdfde', '#2e9b99')
    foreground = '#333'
    foreground_strong = '#666'
    foreground_subtle = 'rgba(0, 0, 0, 0.5)'
    opacity = '.5'
    opacity_hover = '.9'
    plot_background = '#ffffff'
    transition = '250ms ease-in'

class pygal.style.LightGreenStyle (**kwargs)
    Bases: pygal.style.Style
    A light green style
    background = '#fbfbfb'
    colors = ('#7dcf30', '#247fab', '#97d959', '#ccc', '#579122', '#ddd', '#2f9ed3', '#17506c')
    foreground = '#333333'
    foreground_strong = '#666'
    foreground_subtle = '#222222'
    opacity = '.5'
    opacity_hover = '.9'
    plot_background = '#fff'
    transition = '250ms ease-in'

class pygal.style.LightSolarizedStyle (**kwargs)
    Bases: pygal.style.DarkSolarizedStyle
    Light solarized popular theme
    background = '#fdf6e3'
    foreground = '#657b83'
    foreground_strong = '#073642'
    foreground_subtle = '#073642'
    plot_background = '#eee8d5'

class pygal.style.LightStyle (**kwargs)
    Bases: pygal.style.Style
    A light style
    background = 'white'
    colors = ('#242424', '#9f6767', '#92ac68', '#d0d293', '#9aacc3', '#bb77a4', '#77bbb5', '#777777')

```

```

foreground = 'rgba(0, 0, 0, 0.7)'
foreground_strong = 'rgba(0, 0, 0, 0.9)'
foreground_subtle = 'rgba(0, 0, 0, 0.5)'
plot_background = 'rgba(0, 0, 255, 0.1)'

```

```

class pygal.style.LightenStyle(color, step=10, max_=None, base_style=None, **kwargs)
    Bases: pygal.style.ParametricStyleBase

```

Create a style by lightening the given color

```

class pygal.style.NeonStyle(**kwargs)
    Bases: pygal.style.DarkStyle

```

Similar to DarkStyle but with more opacity and effects

```

opacity = '.1'
opacity_hover = '.75'
transition = '1s ease-out'

```

```

class pygal.style.ParametricStyleBase(color, step=10, max_=None, base_style=None,
    **kwargs)
    Bases: pygal.style.Style

```

Parametric Style base class for all the parametric operations

```

class pygal.style.RedBlueStyle(**kwargs)
    Bases: pygal.style.Style

```

A red and blue theme

```

background = '#f9f9fa'
colors = ('#d94e4c', '#e5884f', '#39929a', '#e27876', '#245d62', '#f0bb9b', '#c82d2a', '#234547')
foreground = 'rgba(0, 0, 0, 0.9)'
foreground_strong = 'rgba(0, 0, 0, 0.9)'
foreground_subtle = 'rgba(0, 0, 0, 0.5)'
opacity = '.6'
opacity_hover = '.9'
plot_background = '#ffffff'

```

```

class pygal.style.RotateStyle(color, step=10, max_=None, base_style=None, **kwargs)
    Bases: pygal.style.ParametricStyleBase

```

Create a style by rotating the given color

```

class pygal.style.SaturateStyle(color, step=10, max_=None, base_style=None, **kwargs)
    Bases: pygal.style.ParametricStyleBase

```

Create a style by saturating the given color

```

class pygal.style.SolidColorStyle(**kwargs)
    Bases: pygal.style.Style

```

A light style with strong colors

```

background = '#FFFFFF'
colors = ('#FF9900', '#DC3912', '#4674D1', '#109618', '#990099', '#0099C6', '#DD4477', '#74B217', '#B82E2E', '#31687F')

```

```

foreground = '#000000'
foreground_strong = '#000000'
foreground_subtle = '#828282'
opacity = '.8'
opacity_hover = '.9'
plot_background = '#FFFFFF'
transition = '400ms ease-in'

```

```
class pygal.style.Style(**kwargs)
```

```
Bases: object
```

Styling class containing colors for the css generation

```
background = 'rgba(249, 249, 249, 1)'
```

```
ci_colors = ()
```

```
colors = ('#F44336', '#3F51B5', '#009688', '#FFC107', '#FF5722', '#9C27B0', '#03A9F4', '#8BC34A', '#FF9800', '#E91E63')
```

```
font_family = 'Consolas, "Liberation Mono", Menlo, Courier, monospace'
```

```
foreground = 'rgba(0, 0, 0, .87)'
```

```
foreground_strong = 'rgba(0, 0, 0, 1)'
```

```
foreground_subtle = 'rgba(0, 0, 0, .54)'
```

```
get_colors(prefix, len_)
```

```
Get the css color list
```

```
guide_stroke_dasharray = '4,4'
```

```
label_font_family = None
```

```
label_font_size = 10
```

```
legend_font_family = None
```

```
legend_font_size = 14
```

```
major_guide_stroke_dasharray = '6,6'
```

```
major_label_font_family = None
```

```
major_label_font_size = 10
```

```
no_data_font_family = None
```

```
no_data_font_size = 64
```

```
opacity = '.7'
```

```
opacity_hover = '.8'
```

```
plot_background = 'rgba(255, 255, 255, 1)'
```

```
stroke_opacity = '.8'
```

```
stroke_opacity_hover = '.9'
```

```
title_font_family = None
```

```
title_font_size = 16
```

```
to_dict ()
    Convert instance to a serializable mapping.

tooltip_font_family = None
tooltip_font_size = 14
transition = '150ms'
value_background = 'rgba(229, 229, 229, 1)'
value_colors = ()
value_font_family = None
value_font_size = 16
value_label_font_family = None
value_label_font_size = 10
```

```
class pygal.style.TurquoiseStyle (**kwargs)
    Bases: pygal.style.Style
    A turquoise style
    background = '#0e4448'
    colors = ('#93d2d9', '#ef940f', '#8C6243', '#fff', '#48b3be', '#f4b456', '#b68866', '#1b8088')
    foreground = 'rgba(255, 255, 255, 0.9)'
    foreground_strong = 'rgba(255, 255, 255, 0.9)'
    foreground_subtle = 'rgba(255, 255, 255, 0.5)'
    opacity = '.5'
    opacity_hover = '.9'
    plot_background = '#0d3c40'
    transition = '250ms ease-in'
```

## pygal.svg module

Svg helper

```
class pygal.svg.Svg (graph)
    Bases: object
    Svg related methods
    add_scripts ()
        Add the js to the svg
    add_styles ()
        Add the css to the svg
    confidence_interval (node, x, low, high, width=7)
    draw_no_data ()
        Write the no data text to the svg
    gauge_background (serie_node, start_angle, center, radius, small_radius, end_angle, half_pie,
                    max_value)
```

**get\_strokes** ()  
Return a css snippet containing all stroke style options

**line** (*node*, *coords*, *close=False*, *\*\*kwargs*)  
Draw a svg line

**node** (*parent=None*, *tag='g'*, *attrib=None*, *\*\*extras*)  
Make a new svg node

**ns** = 'http://www.w3.org/2000/svg'

**pre\_render** ()  
Last things to do before rendering

**render** (*is\_unicode=False*, *pretty\_print=False*)  
Last thing to do before rendering

**serie** (*serie*)  
Make serie node

**slice** (*serie\_node*, *node*, *radius*, *small\_radius*, *angle*, *start\_angle*, *center*, *val*, *i*, *metadata*)  
Draw a pie slice

**solid\_gauge** (*serie\_node*, *node*, *radius*, *small\_radius*, *angle*, *start\_angle*, *center*, *val*, *i*, *metadata*, *half\_pie*, *end\_angle*, *max\_value*)  
Draw a solid gauge slice and background slice

**transposable\_node** (*parent=None*, *tag='g'*, *attrib=None*, *\*\*extras*)  
Make a new svg node which can be transposed if horizontal

**xlink\_ns** = 'http://www.w3.org/1999/xlink'

### pygal.table module

HTML Table maker.

This class is used to render an html table from a chart data.

**class** `pygal.table.HTML`

Bases: `object`

Lower case adapter of lxml builder

**class** `pygal.table.Table` (*chart*)

Bases: `object`

Table generator class

**render** (*total=False*, *transpose=False*, *style=False*)

Render the HTML table of the chart.

*total* can be specified to include data sums *transpose* make labels becomes columns *style* include scoped style for the table

### pygal.util module

Various utility functions

`pygal.util.alter` (*node*, *metadata*)

Override nodes attributes from metadata node mapping

**class** `pygal.util.cached_property` (*getter, doc=None*)  
Bases: `object`  
Memoize a property

`pygal.util.compose` (*f, g*)  
Chain functions

`pygal.util.compute_logarithmic_scale` (*min\_, max\_, min\_scale, max\_scale*)  
Compute an optimal scale for logarithmic

`pygal.util.compute_scale` (*min\_, max\_, logarithmic, order\_min, min\_scale, max\_scale*)  
Compute an optimal scale between min and max

`pygal.util.coord_abs_project` (*center, rho, theta*)

`pygal.util.coord_diff` (*x, y*)

`pygal.util.coord_dual` (*r*)

`pygal.util.coord_format` (*x*)

`pygal.util.coord_project` (*rho, alpha*)

`pygal.util.cut` (*list\_, index=0*)  
Cut a list by index or arg

`pygal.util.decorate` (*svg, node, metadata*)  
Add metedata next to a node

`pygal.util.deg` (*radiants*)  
Convert radiants in degrees

`pygal.util.filter_kwargs` (*fun, kwargs*)

`pygal.util.float_format` (*number*)  
Format a float to a precision of 3, without zeroes or dots

`pygal.util.get_text_box` (*text, fs*)  
Approximation of text bounds

`pygal.util.get_texts_box` (*texts, fs*)  
Approximation of multiple texts bounds

`pygal.util.ident` (*x*)

`pygal.util.majorize` (*values*)  
Filter sequence to return only major considered numbers

`pygal.util.merge` (*dict1, dict2*)

`pygal.util.mergextend` (*list1, list2*)

`pygal.util.minify_css` (*css*)  
Little css minifier

`pygal.util.rad` (*degrees*)  
Convert degrees in radiants

`pygal.util.reverse_text_len` (*width, fs*)  
Approximation of text length

`pygal.util.round_to_float` (*number, precision*)  
Round a float to a precision



`pygal.util.round_to_int` (*number, precision*)  
Round a number to a precision

`pygal.util.round_to_scale` (*number, precision*)  
Round a number or a float to a precision

`pygal.util.safe_enumerate` (*iterable*)  
Enumerate which does not yield None values

`pygal.util.split_title` (*title, width, title\_fs*)  
Split a string for a specified width and font size

`pygal.util.swap` (*tuple\_*)

`pygal.util.template` (*string, \*\*kwargs*)  
Format a string using double braces

`pygal.util.text_len` (*length, fs*)  
Approximation of text width

`pygal.util.truncate` (*string, index*)  
Truncate a string at index and add ...

### pygal.view module

Projection and bounding helpers

**class** `pygal.view.Box` (*xmin=0, ymin=0, xmax=1, ymax=1*)  
Bases: `object`

Chart boundingings

**fix** (*with\_margin=True*)  
Correct box when no values and take margin in account

**height**  
Helper for box height

**margin = 0.02**

**set\_polar\_box** (*rmin=0, rmax=1, tmin=0, tmax=6.283185307179586*)  
Helper for polar charts

**swap** ()  
Return the box (for horizontal graphs)

**width**  
Helper for box width

**xmax**  
X maximum getter

**xmin**  
X minimum getter

**ymax**  
Y maximum getter

**ymin**  
Y minimum getter

**class** `pygal.view.HorizontalLogView` (*width, height, box*)  
Bases: `pygal.view.XLogView`

Transposed Logarithmic projection

**x** (*x*)  
Project x as y

**y** (*y*)  
Project y as x

**class** `pygal.view.HorizontalView` (*width, height, box*)  
Bases: `pygal.view.View`

Same as view but transposed

**x** (*x*)  
Project x as y

**y** (*y*)  
Project y as x

**class** `pygal.view.LogView` (*width, height, box*)  
Bases: `pygal.view.View`

Y Logarithmic projection

**y** (*y*)  
Project y

**class** `pygal.view.Margin` (*top, right, bottom, left*)  
Bases: `object`

Class representing a margin (top, right, left, bottom)

**x**  
Helper for total x margin

**y**  
Helper for total y margin

**class** `pygal.view.PolarLogView` (*width, height, box*)  
Bases: `pygal.view.View`

Logarithmic polar projection

**class** `pygal.view.PolarThetaLogView` (*width, height, box, aperture=1.0471975511965976*)  
Bases: `pygal.view.View`

Logarithmic polar projection

**class** `pygal.view.PolarThetaView` (*width, height, box, aperture=1.0471975511965976*)  
Bases: `pygal.view.View`

Logarithmic polar projection

**class** `pygal.view.PolarView` (*width, height, box*)  
Bases: `pygal.view.View`

Polar projection for pie like graphs

**class** `pygal.view.ReverseView` (*width, height, box*)  
Bases: `pygal.view.View`

Same as view but reversed vertically

**Y** (*y*)  
Project reversed y

**class** `pygal.view.View` (*width, height, box*)

Bases: `object`

Projection base class

**x** (*x*)  
Project x

**Y** (*y*)  
Project y

**class** `pygal.view.XLogView` (*width, height, box*)

Bases: `pygal.view.View`

X logarithmic projection

**x** (*x*)  
Project x

**class** `pygal.view.XYLogView` (*width, height, box*)

Bases: `pygal.view.XLogView`, `pygal.view.LogView`

X and Y logarithmic projection

- `genindex`
- `modindex`
- `search`



**p**

- pygal, 69
- pygal.adapters, 85
- pygal.colors, 86
- pygal.config, 86
- pygal.etree, 90
- pygal.formatters, 90
- pygal.graph, 70
  - pygal.graph.bar, 70
  - pygal.graph.base, 70
  - pygal.graph.box, 70
  - pygal.graph.dot, 71
  - pygal.graph.dual, 71
  - pygal.graph.funnel, 71
  - pygal.graph.gauge, 71
  - pygal.graph.graph, 71
  - pygal.graph.histogram, 71
  - pygal.graph.horizontal, 72
  - pygal.graph.horizontalbar, 72
  - pygal.graph.horizontalline, 72
  - pygal.graph.horizontalstackedbar, 72
  - pygal.graph.horizontalstackedline, 72
  - pygal.graph.line, 72
  - pygal.graph.map, 73
  - pygal.graph.pie, 73
  - pygal.graph.public, 73
  - pygal.graph.pyramid, 74
  - pygal.graph.radar, 74
  - pygal.graph.solidgauge, 74
  - pygal.graph.stackedbar, 74
  - pygal.graph.stackedline, 74
  - pygal.graph.time, 75
  - pygal.graph.treemap, 75
  - pygal.graph.xy, 75
- pygal.interpolate, 91
- pygal.maps, 76
- pygal.serie, 91
- pygal.state, 92
- pygal.stats, 92
- pygal.style, 92
- pygal.svg, 98
- pygal.table, 99
- pygal.test, 76
  - pygal.test.confctest, 76
  - pygal.test.test\_bar, 76
  - pygal.test.test\_box, 76
  - pygal.test.test\_colors, 76
  - pygal.test.test\_config, 79
  - pygal.test.test\_date, 80
  - pygal.test.test\_formatters, 80
  - pygal.test.test\_graph, 81
  - pygal.test.test\_histogram, 82
  - pygal.test.test\_interpolate, 82
  - pygal.test.test\_line, 83
  - pygal.test.test\_maps, 83
  - pygal.test.test\_pie, 83
  - pygal.test.test\_serie\_config, 83
  - pygal.test.test\_sparktext, 84
  - pygal.test.test\_stacked, 84
  - pygal.test.test\_style, 84
  - pygal.test.test\_table, 84
  - pygal.test.test\_util, 85
  - pygal.test.test\_view, 85
  - pygal.test.test\_xml\_filters, 85
- pygal.test.utils, 85
- pygal.util, 99
- pygal.view, 101



**A**

adapt() (in module pygal.test), 76  
 adapt\_code() (pygal.graph.map.BaseMap method), 73  
 add() (pygal.graph.public.PublicApi method), 73  
 add\_scripts() (pygal.svg.Svg method), 98  
 add\_squares() (pygal.graph.graph.Graph method), 71  
 add\_styles() (pygal.svg.Svg method), 98  
 add\_xml\_filter() (pygal.graph.public.PublicApi method), 73  
 adjust() (in module pygal.colors), 86  
 all\_series (pygal.graph.graph.Graph attribute), 71  
 allow\_interruptions (pygal.config.CommonConfig attribute), 87  
 alter() (in module pygal.util), 99

**B**

background (pygal.style.BlueStyle attribute), 92  
 background (pygal.style.CleanStyle attribute), 92  
 background (pygal.style.DarkColorizedStyle attribute), 93  
 background (pygal.style.DarkGreenBlueStyle attribute), 93  
 background (pygal.style.DarkGreenStyle attribute), 93  
 background (pygal.style.DarkSolarizedStyle attribute), 94  
 background (pygal.style.DarkStyle attribute), 94  
 background (pygal.style.LightColorizedStyle attribute), 95  
 background (pygal.style.LightGreenStyle attribute), 95  
 background (pygal.style.LightSolarizedStyle attribute), 95  
 background (pygal.style.LightStyle attribute), 95  
 background (pygal.style.RedBlueStyle attribute), 96  
 background (pygal.style.SolidColorStyle attribute), 96  
 background (pygal.style.Style attribute), 97  
 background (pygal.style.TurquoiseStyle attribute), 98  
 Bar (class in pygal.graph.bar), 70  
 bar() (pygal.graph.bar.Bar method), 70  
 bar() (pygal.graph.histogram.Histogram method), 71  
 BaseConfig (class in pygal.config), 86  
 BaseGraph (class in pygal.graph.base), 70

BaseMap (class in pygal.graph.map), 73  
 BlueStyle (class in pygal.style), 92  
 Box (class in pygal.graph.box), 70  
 Box (class in pygal.view), 101  
 box\_mode (pygal.config.Config attribute), 87

**C**

cached\_property (class in pygal.util), 99  
 ChangeBarsXMLFilter (class in pygal.test.test\_xml\_filters), 85  
 ci\_colors (pygal.style.Style attribute), 97  
 classes (pygal.config.Config attribute), 87  
 CleanStyle (class in pygal.style), 92  
 coerce() (pygal.config.Key method), 89  
 colors (pygal.style.BlueStyle attribute), 92  
 colors (pygal.style.CleanStyle attribute), 93  
 colors (pygal.style.DarkColorizedStyle attribute), 93  
 colors (pygal.style.DarkGreenBlueStyle attribute), 93  
 colors (pygal.style.DarkGreenStyle attribute), 93  
 colors (pygal.style.DarkSolarizedStyle attribute), 94  
 colors (pygal.style.DarkStyle attribute), 94  
 colors (pygal.style.LightColorizedStyle attribute), 95  
 colors (pygal.style.LightGreenStyle attribute), 95  
 colors (pygal.style.LightStyle attribute), 95  
 colors (pygal.style.RedBlueStyle attribute), 96  
 colors (pygal.style.SolidColorStyle attribute), 96  
 colors (pygal.style.Style attribute), 97  
 colors (pygal.style.TurquoiseStyle attribute), 98  
 CommonConfig (class in pygal.config), 87  
 compose() (in module pygal.util), 100  
 compute\_logarithmic\_scale() (in module pygal.util), 100  
 compute\_scale() (in module pygal.util), 100  
 confidence\_interval() (pygal.svg.Svg method), 98  
 confidence\_interval\_continuous() (in module pygal.stats), 92  
 confidence\_interval\_dichotomous() (in module pygal.stats), 92  
 confidence\_interval\_manual() (in module pygal.stats), 92  
 Config (class in pygal.config), 87  
 coord\_abs\_project() (in module pygal.util), 100  
 coord\_diff() (in module pygal.util), 100

coord\_dual() (in module pygal.util), 100  
 coord\_format() (in module pygal.util), 100  
 coord\_project() (in module pygal.util), 100  
 copy() (pygal.config.BaseConfig method), 86  
 css (pygal.config.Config attribute), 87  
 cubic\_interpolate() (in module pygal.interpolate), 91  
 cut() (in module pygal.util), 100

## D

DarkColorizedStyle (class in pygal.style), 93  
 darken() (in module pygal.colors), 86  
 DarkenStyle (class in pygal.style), 94  
 DarkGreenBlueStyle (class in pygal.style), 93  
 DarkGreenStyle (class in pygal.style), 93  
 DarkSolarizedStyle (class in pygal.style), 94  
 DarkStyle (class in pygal.style), 94  
 date\_to\_datetime() (in module pygal.graph.time), 75  
 DateLine (class in pygal.graph.time), 75  
 datetime\_to\_time() (in module pygal.graph.time), 75  
 datetime\_to\_timestamp() (in module pygal.graph.time), 75  
 DateTimeLine (class in pygal.graph.time), 75  
 decimal\_to\_float() (in module pygal.adapters), 85  
 decorate() (in module pygal.util), 100  
 Default (class in pygal.formatters), 90  
 DefaultStyle (in module pygal.style), 94  
 defs (pygal.config.Config attribute), 87  
 deg() (in module pygal.util), 100  
 desaturate() (in module pygal.colors), 86  
 DesaturateStyle (class in pygal.style), 94  
 disable\_xml\_declaration (pygal.config.Config attribute), 87  
 Dot (class in pygal.graph.dot), 71  
 dot() (pygal.graph.dot.Dot method), 71  
 dots\_size (pygal.config.CommonConfig attribute), 87  
 draw\_no\_data() (pygal.svg.Svg method), 98  
 Dual (class in pygal.graph.dual), 71  
 dynamic\_print\_values (pygal.config.Config attribute), 87

## E

enumerate\_values() (pygal.graph.map.BaseMap method), 73  
 erfinv() (in module pygal.stats), 92  
 Etree (class in pygal.etree), 90  
 etreefx() (in module pygal.test.confest), 76  
 explicit\_size (pygal.config.Config attribute), 87

## F

fill (pygal.config.CommonConfig attribute), 87  
 filter\_kwargs() (in module pygal.util), 100  
 find\_module() (pygal.PluginImportFixer method), 70  
 fix() (pygal.view.Box method), 101  
 float\_format() (in module pygal.util), 100  
 font\_family (pygal.style.Style attribute), 97

force\_uri\_protocol (pygal.config.Config attribute), 87  
 foreground (pygal.style.BlueStyle attribute), 92  
 foreground (pygal.style.CleanStyle attribute), 93  
 foreground (pygal.style.DarkColorizedStyle attribute), 93  
 foreground (pygal.style.DarkGreenBlueStyle attribute), 93  
 foreground (pygal.style.DarkGreenStyle attribute), 93  
 foreground (pygal.style.DarkSolarizedStyle attribute), 94  
 foreground (pygal.style.DarkStyle attribute), 94  
 foreground (pygal.style.LightColorizedStyle attribute), 95  
 foreground (pygal.style.LightGreenStyle attribute), 95  
 foreground (pygal.style.LightSolarizedStyle attribute), 95  
 foreground (pygal.style.LightStyle attribute), 95  
 foreground (pygal.style.RedBlueStyle attribute), 96  
 foreground (pygal.style.SolidColorStyle attribute), 96  
 foreground (pygal.style.Style attribute), 97  
 foreground (pygal.style.TurquoiseStyle attribute), 98  
 foreground\_strong (pygal.style.BlueStyle attribute), 92  
 foreground\_strong (pygal.style.CleanStyle attribute), 93  
 foreground\_strong (pygal.style.DarkColorizedStyle attribute), 93  
 foreground\_strong (pygal.style.DarkGreenBlueStyle attribute), 93  
 foreground\_strong (pygal.style.DarkGreenStyle attribute), 93  
 foreground\_strong (pygal.style.DarkSolarizedStyle attribute), 94  
 foreground\_strong (pygal.style.DarkStyle attribute), 94  
 foreground\_strong (pygal.style.LightColorizedStyle attribute), 95  
 foreground\_strong (pygal.style.LightGreenStyle attribute), 95  
 foreground\_strong (pygal.style.LightSolarizedStyle attribute), 95  
 foreground\_strong (pygal.style.LightStyle attribute), 96  
 foreground\_strong (pygal.style.RedBlueStyle attribute), 96  
 foreground\_strong (pygal.style.SolidColorStyle attribute), 97  
 foreground\_strong (pygal.style.Style attribute), 97  
 foreground\_strong (pygal.style.TurquoiseStyle attribute), 98  
 foreground\_subtle (pygal.style.BlueStyle attribute), 92  
 foreground\_subtle (pygal.style.CleanStyle attribute), 93  
 foreground\_subtle (pygal.style.DarkColorizedStyle attribute), 93  
 foreground\_subtle (pygal.style.DarkGreenBlueStyle attribute), 93  
 foreground\_subtle (pygal.style.DarkGreenStyle attribute), 93  
 foreground\_subtle (pygal.style.DarkSolarizedStyle attribute), 94  
 foreground\_subtle (pygal.style.DarkStyle attribute), 94



- foreground\_subtle (pygal.style.LightColorizedStyle attribute), 95
- foreground\_subtle (pygal.style.LightGreenStyle attribute), 95
- foreground\_subtle (pygal.style.LightSolarizedStyle attribute), 95
- foreground\_subtle (pygal.style.LightStyle attribute), 96
- foreground\_subtle (pygal.style.RedBlueStyle attribute), 96
- foreground\_subtle (pygal.style.SolidColorStyle attribute), 97
- foreground\_subtle (pygal.style.Style attribute), 97
- foreground\_subtle (pygal.style.TurquoiseStyle attribute), 98
- Formatter (class in pygal.formatters), 90
- formatter (pygal.config.CommonConfig attribute), 87
- Funnel (class in pygal.graph.funnel), 71
- funnel() (pygal.graph.funnel.Funnel method), 71
- ## G
- Gauge (class in pygal.graph.gauge), 71
- gauge\_background() (pygal.svg.Svg method), 98
- gaugify() (pygal.graph.solidgauge.SolidGauge method), 74
- get\_colors() (pygal.style.Style method), 97
- get\_data() (in module pygal.test), 76
- get\_strokes() (pygal.svg.Svg method), 98
- get\_text\_box() (in module pygal.util), 100
- get\_texts\_box() (in module pygal.util), 100
- Graph (class in pygal.graph.graph), 71
- guide\_stroke\_dasharray (pygal.style.Style attribute), 97
- ## H
- half\_pie (pygal.config.Config attribute), 87
- height (pygal.config.Config attribute), 87
- height (pygal.view.Box attribute), 101
- hermite\_interpolate() (in module pygal.interpolate), 91
- Histogram (class in pygal.graph.histogram), 71
- HorizontalBar (class in pygal.graph.horizontalbar), 72
- HorizontalGraph (class in pygal.graph.horizontal), 72
- HorizontalLine (class in pygal.graph.horizontalline), 72
- HorizontalLogView (class in pygal.view), 101
- HorizontalStackedBar (class in pygal.graph.horizontalstackedbar), 72
- HorizontalStackedLine (class in pygal.graph.horizontalstackedline), 72
- HorizontalView (class in pygal.view), 102
- hsl\_to\_rgb() (in module pygal.colors), 86
- HTML (class in pygal.table), 99
- HumanReadable (class in pygal.formatters), 90
- ## I
- ident() (in module pygal.util), 100
- include\_x\_axis (pygal.config.Config attribute), 87
- inner\_radius (pygal.config.CommonConfig attribute), 87
- inner\_radius (pygal.config.Config attribute), 87
- Integer (class in pygal.formatters), 90
- interpolate (pygal.config.Config attribute), 87
- interpolation\_parameters (pygal.config.Config attribute), 87
- interpolation\_precision (pygal.config.Config attribute), 87
- inverse\_y\_axis (pygal.config.Config attribute), 87
- is\_boolean (pygal.config.Key attribute), 89
- is\_dict (pygal.config.Key attribute), 89
- is\_foreground\_light() (in module pygal.colors), 86
- is\_list (pygal.config.Key attribute), 89
- is\_numeric (pygal.config.Key attribute), 89
- is\_string (pygal.config.Key attribute), 89
- IsoDateTime (class in pygal.formatters), 90
- ## J
- js (pygal.config.Config attribute), 87
- ## K
- Key (class in pygal.config), 89
- ## L
- label\_font\_family (pygal.style.Style attribute), 97
- label\_font\_size (pygal.style.Style attribute), 97
- lagrange\_interpolate() (in module pygal.interpolate), 91
- legend\_at\_bottom (pygal.config.Config attribute), 87
- legend\_at\_bottom\_columns (pygal.config.Config attribute), 88
- legend\_box\_size (pygal.config.Config attribute), 88
- legend\_font\_family (pygal.style.Style attribute), 97
- legend\_font\_size (pygal.style.Style attribute), 97
- LightColorizedStyle (class in pygal.style), 94
- lighten() (in module pygal.colors), 86
- LightenStyle (class in pygal.style), 96
- LightGreenStyle (class in pygal.style), 95
- LightSolarizedStyle (class in pygal.style), 95
- LightStyle (class in pygal.style), 95
- Line (class in pygal.graph.line), 72
- line() (pygal.graph.line.Line method), 72
- line() (pygal.svg.Svg method), 99
- load\_module() (pygal.PluginImportFixer method), 70
- logarithmic (pygal.config.Config attribute), 88
- LogView (class in pygal.view), 102
- ## M
- major\_guide\_stroke\_dasharray (pygal.style.Style attribute), 97
- major\_label\_font\_family (pygal.style.Style attribute), 97
- major\_label\_font\_size (pygal.style.Style attribute), 97
- majorize() (in module pygal.util), 100
- make\_data() (in module pygal.test), 76

Margin (class in pygal.view), 102  
margin (pygal.config.Config attribute), 88  
margin (pygal.view.Box attribute), 101  
margin\_bottom (pygal.config.Config attribute), 88  
margin\_left (pygal.config.Config attribute), 88  
margin\_right (pygal.config.Config attribute), 88  
margin\_top (pygal.config.Config attribute), 88  
max\_scale (pygal.config.Config attribute), 88  
merge() (in module pygal.util), 100  
mergextend() (in module pygal.util), 100  
MetaConfig (class in pygal.config), 89  
min\_scale (pygal.config.Config attribute), 88  
minify\_css() (in module pygal.util), 100  
missing\_value\_fill\_truncation (pygal.config.Config attribute), 88

## N

needle() (pygal.graph.gauge.Gauge method), 71  
needle\_width (pygal.graph.gauge.Gauge attribute), 71  
NeonStyle (class in pygal.style), 96  
no\_data\_font\_family (pygal.style.Style attribute), 97  
no\_data\_font\_size (pygal.style.Style attribute), 97  
no\_data\_text (pygal.config.Config attribute), 88  
no\_prefix (pygal.config.Config attribute), 88  
node() (pygal.svg.Svg method), 99  
none\_to\_zero() (in module pygal.adapters), 85  
norm\_ppf() (in module pygal.stats), 92  
normalize\_float() (in module pygal.colors), 86  
not\_zero() (in module pygal.adapters), 86  
ns (pygal.svg.Svg attribute), 99

## O

opacity (pygal.style.BlueStyle attribute), 92  
opacity (pygal.style.DarkColorizedStyle attribute), 93  
opacity (pygal.style.DarkGreenBlueStyle attribute), 93  
opacity (pygal.style.DarkGreenStyle attribute), 93  
opacity (pygal.style.DarkSolarizedStyle attribute), 94  
opacity (pygal.style.DarkStyle attribute), 94  
opacity (pygal.style.LightColorizedStyle attribute), 95  
opacity (pygal.style.LightGreenStyle attribute), 95  
opacity (pygal.style.NeonStyle attribute), 96  
opacity (pygal.style.RedBlueStyle attribute), 96  
opacity (pygal.style.SolidColorStyle attribute), 97  
opacity (pygal.style.Style attribute), 97  
opacity (pygal.style.TurquoiseStyle attribute), 98  
opacity\_hover (pygal.style.BlueStyle attribute), 92  
opacity\_hover (pygal.style.DarkColorizedStyle attribute), 93  
opacity\_hover (pygal.style.DarkGreenBlueStyle attribute), 93  
opacity\_hover (pygal.style.DarkGreenStyle attribute), 94  
opacity\_hover (pygal.style.DarkSolarizedStyle attribute), 94  
opacity\_hover (pygal.style.DarkStyle attribute), 94

opacity\_hover (pygal.style.LightColorizedStyle attribute), 95  
opacity\_hover (pygal.style.LightGreenStyle attribute), 95  
opacity\_hover (pygal.style.NeonStyle attribute), 96  
opacity\_hover (pygal.style.RedBlueStyle attribute), 96  
opacity\_hover (pygal.style.SolidColorStyle attribute), 97  
opacity\_hover (pygal.style.Style attribute), 97  
opacity\_hover (pygal.style.TurquoiseStyle attribute), 98  
order\_min (pygal.config.Config attribute), 88  
ORDERS (pygal.formatters.HumanReadable attribute), 90

## P

ParametricStyleBase (class in pygal.style), 96  
parse\_color() (in module pygal.colors), 86  
Pie (class in pygal.graph.pie), 73  
plot\_background (pygal.style.BlueStyle attribute), 92  
plot\_background (pygal.style.CleanStyle attribute), 93  
plot\_background (pygal.style.DarkColorizedStyle attribute), 93  
plot\_background (pygal.style.DarkGreenBlueStyle attribute), 93  
plot\_background (pygal.style.DarkGreenStyle attribute), 94  
plot\_background (pygal.style.DarkSolarizedStyle attribute), 94  
plot\_background (pygal.style.DarkStyle attribute), 94  
plot\_background (pygal.style.LightColorizedStyle attribute), 95  
plot\_background (pygal.style.LightGreenStyle attribute), 95  
plot\_background (pygal.style.LightSolarizedStyle attribute), 95  
plot\_background (pygal.style.LightStyle attribute), 96  
plot\_background (pygal.style.RedBlueStyle attribute), 96  
plot\_background (pygal.style.SolidColorStyle attribute), 97  
plot\_background (pygal.style.Style attribute), 97  
plot\_background (pygal.style.TurquoiseStyle attribute), 98  
PluginImportFixer (class in pygal), 69  
PolarLogView (class in pygal.view), 102  
PolarThetaLogView (class in pygal.view), 102  
PolarThetaView (class in pygal.view), 102  
PolarView (class in pygal.view), 102  
positive() (in module pygal.adapters), 86  
ppf() (in module pygal.stats), 92  
pre\_render() (pygal.svg.Svg method), 99  
prepare\_values() (pygal.graph.base.BaseGraph method), 70  
pretty\_print (pygal.config.Config attribute), 88  
print\_labels (pygal.config.Config attribute), 88  
print\_values (pygal.config.Config attribute), 88  
print\_values\_position (pygal.config.Config attribute), 88

- print\_zeroes (pygal.config.Config attribute), 88
  - PublicApi (class in pygal.graph.public), 73
  - pygal (module), 69
  - pygal.adapters (module), 85
  - pygal.colors (module), 86
  - pygal.config (module), 86
  - pygal.etree (module), 90
  - pygal.formatters (module), 90
  - pygal.graph (module), 70
  - pygal.graph.bar (module), 70
  - pygal.graph.base (module), 70
  - pygal.graph.box (module), 70
  - pygal.graph.dot (module), 71
  - pygal.graph.dual (module), 71
  - pygal.graph.funnel (module), 71
  - pygal.graph.gauge (module), 71
  - pygal.graph.graph (module), 71
  - pygal.graph.histogram (module), 71
  - pygal.graph.horizontal (module), 72
  - pygal.graph.horizontalbar (module), 72
  - pygal.graph.horizontalline (module), 72
  - pygal.graph.horizontalstackedbar (module), 72
  - pygal.graph.horizontalstackedline (module), 72
  - pygal.graph.line (module), 72
  - pygal.graph.map (module), 73
  - pygal.graph.pie (module), 73
  - pygal.graph.public (module), 73
  - pygal.graph.pyramid (module), 74
  - pygal.graph.radar (module), 74
  - pygal.graph.solidgauge (module), 74
  - pygal.graph.stackedbar (module), 74
  - pygal.graph.stackedline (module), 74
  - pygal.graph.time (module), 75
  - pygal.graph.treemap (module), 75
  - pygal.graph.xy (module), 75
  - pygal.interpolate (module), 91
  - pygal.maps (module), 76
  - pygal.serie (module), 91
  - pygal.state (module), 92
  - pygal.stats (module), 92
  - pygal.style (module), 92
  - pygal.svg (module), 98
  - pygal.table (module), 99
  - pygal.test (module), 76
  - pygal.test.conftest (module), 76
  - pygal.test.test\_bar (module), 76
  - pygal.test.test\_box (module), 76
  - pygal.test.test\_colors (module), 76
  - pygal.test.test\_config (module), 79
  - pygal.test.test\_date (module), 80
  - pygal.test.test\_formatters (module), 80
  - pygal.test.test\_graph (module), 81
  - pygal.test.test\_histogram (module), 82
  - pygal.test.test\_interpolate (module), 82
  - pygal.test.test\_line (module), 83
  - pygal.test.test\_maps (module), 83
  - pygal.test.test\_pie (module), 83
  - pygal.test.test\_serie\_config (module), 83
  - pygal.test.test\_sparktext (module), 84
  - pygal.test.test\_stacked (module), 84
  - pygal.test.test\_style (module), 84
  - pygal.test.test\_table (module), 84
  - pygal.test.test\_util (module), 85
  - pygal.test.test\_view (module), 85
  - pygal.test.test\_xml\_filters (module), 85
  - pygal.test.utils (module), 85
  - pygal.util (module), 99
  - pygal.view (module), 101
  - Pyramid (class in pygal.graph.pyramid), 74
  - pytest\_generate\_tests() (in module pygal.test.conftest), 76
- ## Q
- quadratic\_interpolate() (in module pygal.interpolate), 91
- ## R
- rad() (in module pygal.util), 100
  - Radar (class in pygal.graph.radar), 74
  - range (pygal.config.Config attribute), 88
  - Raw (class in pygal.formatters), 90
  - RedBlueStyle (class in pygal.style), 96
  - render() (pygal.graph.public.PublicApi method), 73
  - render() (pygal.svg.Svg method), 99
  - render() (pygal.table.Table method), 99
  - render\_data\_uri() (pygal.graph.public.PublicApi method), 73
  - render\_django\_response() (pygal.graph.public.PublicApi method), 73
  - render\_in\_browser() (pygal.graph.public.PublicApi method), 73
  - render\_pyquery() (pygal.graph.public.PublicApi method), 73
  - render\_response() (pygal.graph.public.PublicApi method), 73
  - render\_sparkline() (pygal.graph.public.PublicApi method), 73
  - render\_sparktext() (pygal.graph.public.PublicApi method), 73
  - render\_table() (pygal.graph.public.PublicApi method), 73
  - render\_to\_file() (pygal.graph.public.PublicApi method), 74
  - render\_to\_png() (pygal.graph.public.PublicApi method), 74
  - render\_tree() (pygal.graph.public.PublicApi method), 74
  - reverse\_text\_len() (in module pygal.util), 100
  - ReverseView (class in pygal.view), 102
  - rgb\_to\_hsl() (in module pygal.colors), 86
  - rotate() (in module pygal.colors), 86
  - RotateStyle (class in pygal.style), 96

round\_to\_float() (in module pygal.util), 100  
 round\_to\_int() (in module pygal.util), 100  
 round\_to\_scale() (in module pygal.util), 101  
 rounded\_bars (pygal.config.CommonConfig attribute), 87  
 rounded\_bars (pygal.config.Config attribute), 88

## S

safe\_enumerate() (in module pygal.util), 101  
 safe\_values (pygal.serie.Serie attribute), 91  
 saturate() (in module pygal.colors), 86  
 SaturateStyle (class in pygal.style), 96  
 secondary (pygal.config.SerieConfig attribute), 90  
 secondary\_range (pygal.config.Config attribute), 88  
 seconds\_to\_time() (in module pygal.graph.time), 75  
 Serie (class in pygal.serie), 91  
 serie() (pygal.svg.Svg method), 99  
 SerieConfig (class in pygal.config), 90  
 set\_polar\_box() (pygal.view.Box method), 101  
 setup() (pygal.graph.base.BaseGraph method), 70  
 show\_dots (pygal.config.CommonConfig attribute), 87  
 show\_legend (pygal.config.Config attribute), 88  
 show\_minor\_x\_labels (pygal.config.Config attribute), 88  
 show\_minor\_y\_labels (pygal.config.Config attribute), 88  
 show\_only\_major\_dots (pygal.config.CommonConfig attribute), 87  
 show\_x\_guides (pygal.config.Config attribute), 88  
 show\_x\_labels (pygal.config.Config attribute), 88  
 show\_y\_guides (pygal.config.Config attribute), 88  
 show\_y\_labels (pygal.config.Config attribute), 88  
 Significant (class in pygal.formatters), 91  
 slice() (pygal.graph.pie.Pie method), 73  
 slice() (pygal.svg.Svg method), 99  
 solid\_gauge() (pygal.svg.Svg method), 99  
 SolidColorStyle (class in pygal.style), 96  
 SolidGauge (class in pygal.graph.solidgauge), 74  
 spacing (pygal.config.Config attribute), 88  
 split\_title() (in module pygal.util), 101  
 stack\_from\_top (pygal.config.Config attribute), 88  
 StackedBar (class in pygal.graph.stackedbar), 74  
 StackedLine (class in pygal.graph.stackedline), 74  
 State (class in pygal.state), 92  
 strict (pygal.config.Config attribute), 88  
 stroke (pygal.config.CommonConfig attribute), 87  
 stroke\_opacity (pygal.style.Style attribute), 97  
 stroke\_opacity\_hover (pygal.style.Style attribute), 97  
 stroke\_style (pygal.config.CommonConfig attribute), 87  
 Style (class in pygal.style), 97  
 style (pygal.config.Config attribute), 88  
 Svg (class in pygal.svg), 98  
 swap() (in module pygal.util), 101  
 swap() (pygal.view.Box method), 101

## T

Table (class in pygal.table), 99

teardown() (pygal.graph.base.BaseGraph method), 70  
 template() (in module pygal.util), 101  
 test\_all\_logarithmic() (in module pygal.test.test\_view), 85  
 test\_all\_sparktext() (in module pygal.test.test\_sparktext), 84  
 test\_another\_sparktext() (in module pygal.test.test\_sparktext), 84  
 test\_basic\_sparktext() (in module pygal.test.test\_sparktext), 84  
 test\_classes() (in module pygal.test.test\_config), 79  
 test\_config\_alterations\_class() (in module pygal.test.test\_config), 79  
 test\_config\_alterations\_instance() (in module pygal.test.test\_config), 79  
 test\_config\_alterations\_kwargs() (in module pygal.test.test\_config), 79  
 test\_config\_behaviours() (in module pygal.test.test\_config), 79  
 test\_css() (in module pygal.test.test\_config), 79  
 test\_cubic() (in module pygal.test.test\_interpolate), 82  
 test\_cubic\_prec() (in module pygal.test.test\_interpolate), 82  
 test\_darken() (in module pygal.test.test\_colors), 76  
 test\_date() (in module pygal.test.test\_date), 80  
 test\_date\_labels() (in module pygal.test.test\_date), 80  
 test\_date\_xrange() (in module pygal.test.test\_date), 80  
 test\_datetime() (in module pygal.test.test\_date), 80  
 test\_desaturate() (in module pygal.test.test\_colors), 76  
 test\_donut() (in module pygal.test.test\_pie), 83  
 test\_empty\_lists() (in module pygal.test.test\_graph), 81  
 test\_empty\_lists\_with\_nones() (in module pygal.test.test\_graph), 81  
 test\_fill() (in module pygal.test.test\_config), 79  
 test\_format() (in module pygal.test.test\_util), 85  
 test\_formatters() (in module pygal.test.test\_config), 79  
 test\_global\_config() (in module pygal.test.test\_serie\_config), 83  
 test\_half\_pie() (in module pygal.test.test\_pie), 83  
 test\_hermite() (in module pygal.test.test\_interpolate), 82  
 test\_hermite\_cardinal() (in module pygal.test.test\_interpolate), 82  
 test\_hermite\_catmull\_rom() (in module pygal.test.test\_interpolate), 82  
 test\_hermite\_finite() (in module pygal.test.test\_interpolate), 82  
 test\_hermite\_kochanek\_bartels() (in module pygal.test.test\_interpolate), 82  
 test\_histogram() (in module pygal.test.test\_histogram), 82  
 test\_hsl\_to\_rgb\_part\_0() (in module pygal.test.test\_colors), 76  
 test\_hsl\_to\_rgb\_part\_1() (in module pygal.test.test\_colors), 77

test_hsl_to_rgb_part_10()	(in module gal.test.test_colors), 77	py-	test_legend_at_bottom()	(in module pygal.test.test_config), 79
test_hsl_to_rgb_part_11()	(in module gal.test.test_colors), 77	py-	test_lighten()	(in module pygal.test.test_colors), 77
test_hsl_to_rgb_part_12()	(in module gal.test.test_colors), 77	py-	test_line()	(in module pygal.test.test_line), 83
test_hsl_to_rgb_part_13()	(in module gal.test.test_colors), 77	py-	test_line_secondary()	(in module pygal.test.test_line), 83
test_hsl_to_rgb_part_14()	(in module gal.test.test_colors), 77	py-	test_logarithmic()	(in module pygal.test.test_config), 79
test_hsl_to_rgb_part_15()	(in module gal.test.test_colors), 77	py-	test_logarithmic_bad_interpolation()	(in module pygal.test.test_config), 79
test_hsl_to_rgb_part_16()	(in module gal.test.test_colors), 77	py-	test_logarithmic_big_scale()	(in module pygal.test.test_config), 79
test_hsl_to_rgb_part_17()	(in module gal.test.test_colors), 77	py-	test_logarithmic_small_scale()	(in module pygal.test.test_config), 79
test_hsl_to_rgb_part_18()	(in module gal.test.test_colors), 77	py-	test_long_title()	(in module pygal.test.test_graph), 81
test_hsl_to_rgb_part_2()	(in module gal.test.test_colors), 77	py-	test_majorize()	(in module pygal.test.test_util), 85
test_hsl_to_rgb_part_3()	(in module gal.test.test_colors), 77	py-	test_mergextend()	(in module pygal.test.test_util), 85
test_hsl_to_rgb_part_4()	(in module gal.test.test_colors), 77	py-	test_meta_config()	(in module pygal.test.test_config), 79
test_hsl_to_rgb_part_5()	(in module gal.test.test_colors), 77	py-	test_metadata()	(in module pygal.test.test_graph), 81
test_hsl_to_rgb_part_6()	(in module gal.test.test_colors), 77	py-	test_minify_css()	(in module pygal.test.test_util), 85
test_hsl_to_rgb_part_7()	(in module gal.test.test_colors), 77	py-	test_multi_render()	(in module pygal.test.test_graph), 81
test_hsl_to_rgb_part_8()	(in module gal.test.test_colors), 77	py-	test_multiseries_donut()	(in module pygal.test.test_pie), 83
test_hsl_to_rgb_part_9()	(in module gal.test.test_colors), 77	py-	test_negative_and_float_sparktext()	(in module pygal.test.test_sparktext), 84
test_human_readable()	(in module pygal.test.test_config), 79	py-	test_no_data()	(in module pygal.test.test_config), 79
test_human_readable()	(in module pygal.test.test_formatters), 80	py-	test_no_data_interpolation()	(in module pygal.test.test_config), 80
test_human_readable_custom()	(in module pygal.test.test_formatters), 80	py-	test_no_data_sparktext()	(in module pygal.test.test_sparktext), 84
test_include_x_axis()	(in module pygal.test.test_config), 79	py-	test_no_data_with_empty_serie()	(in module pygal.test.test_graph), 81
test_inline_css()	(in module pygal.test.test_config), 79	py-	test_no_data_with_empty_serie_interpolation()	(in module pygal.test.test_config), 80
test_int_x_labels()	(in module pygal.test.test_line), 83	py-	test_no_data_with_empty_series()	(in module pygal.test.test_graph), 81
test_interpolation()	(in module pygal.test.test_config), 79	py-	test_no_data_with_list_of_none()	(in module pygal.test.test_graph), 81
test_ipython_notebook()	(in module pygal.test.test_graph), 81	py-	test_no_data_with_lists_of_nones()	(in module pygal.test.test_graph), 81
test_iterable_types()	(in module pygal.test.test_graph), 81	py-	test_no_data_with_no_values()	(in module pygal.test.test_graph), 81
test_label_rotation()	(in module pygal.test.test_config), 79	py-	test_no_data_with_no_values_with_include_x_axis()	(in module pygal.test.test_graph), 81
test_labels_with_links()	(in module pygal.test.test_graph), 81	py-	test_no_data_with_none()	(in module pygal.test.test_graph), 81
test_lagrange()	(in module pygal.test.test_interpolate), 82	py-	test_no_dot()	(in module pygal.test.test_line), 83
		py-	test_no_dot_at_all()	(in module pygal.test.test_line), 83
		py-	test_no_serie_config()	(in module pygal.test.test_serie_config), 83
		py-	test_no_y_labels()	(in module pygal.test.test_config), 80
		py-	test_non_iterable_value()	(in module pygal.test.test_graph), 81
		py-	test_not_equal_x_labels()	(in module pygal.test.test_line), 83
		py-	test_one_dot()	(in module pygal.test.test_line), 83

test\_only\_major\_dots() (in module pygal.test.test\_line), 83

test\_only\_major\_dots\_count() (in module pygal.test.test\_line), 83

test\_only\_major\_dots\_every() (in module pygal.test.test\_line), 83

test\_only\_major\_dots\_no\_labels() (in module pygal.test.test\_line), 83

test\_only\_one\_value() (in module pygal.test.test\_graph), 81

test\_only\_one\_value\_intrap() (in module pygal.test.test\_graph), 81

test\_only\_one\_value\_log() (in module pygal.test.test\_graph), 81

test\_parametric\_styles() (in module pygal.test.test\_style), 84

test\_parametric\_styles\_with\_parameters() (in module pygal.test.test\_style), 84

test\_parse\_color() (in module pygal.test.test\_colors), 77

test\_pie\_table() (in module pygal.test.test\_table), 84

test\_quadratic() (in module pygal.test.test\_interpolate), 82

test\_quartiles() (in module pygal.test.test\_box), 76

test\_quartiles\_min\_extremes() (in module pygal.test.test\_box), 76

test\_quartiles\_stddev() (in module pygal.test.test\_box), 76

test\_quartiles\_tukey() (in module pygal.test.test\_box), 76

test\_range() (in module pygal.test.test\_config), 80

test\_render\_data\_uri() (in module pygal.test.test\_config), 80

test\_render\_to\_file() (in module pygal.test.test\_graph), 82

test\_render\_to\_png() (in module pygal.test.test\_graph), 82

test\_rgb\_to\_hsl\_part\_0() (in module pygal.test.test\_colors), 77

test\_rgb\_to\_hsl\_part\_1() (in module pygal.test.test\_colors), 78

test\_rgb\_to\_hsl\_part\_10() (in module pygal.test.test\_colors), 78

test\_rgb\_to\_hsl\_part\_11() (in module pygal.test.test\_colors), 78

test\_rgb\_to\_hsl\_part\_12() (in module pygal.test.test\_colors), 78

test\_rgb\_to\_hsl\_part\_13() (in module pygal.test.test\_colors), 78

test\_rgb\_to\_hsl\_part\_14() (in module pygal.test.test\_colors), 78

test\_rgb\_to\_hsl\_part\_15() (in module pygal.test.test\_colors), 78

test\_rgb\_to\_hsl\_part\_16() (in module pygal.test.test\_colors), 78

test\_rgb\_to\_hsl\_part\_17() (in module pygal.test.test\_colors), 78

test\_rgb\_to\_hsl\_part\_18() (in module pygal.test.test\_colors), 78

test\_rgb\_to\_hsl\_part\_2() (in module pygal.test.test\_colors), 78

test\_rgb\_to\_hsl\_part\_3() (in module pygal.test.test\_colors), 78

test\_rgb\_to\_hsl\_part\_4() (in module pygal.test.test\_colors), 78

test\_rgb\_to\_hsl\_part\_5() (in module pygal.test.test\_colors), 78

test\_rgb\_to\_hsl\_part\_6() (in module pygal.test.test\_colors), 78

test\_rgb\_to\_hsl\_part\_7() (in module pygal.test.test\_colors), 78

test\_rgb\_to\_hsl\_part\_8() (in module pygal.test.test\_colors), 78

test\_rgb\_to\_hsl\_part\_9() (in module pygal.test.test\_colors), 78

test\_rotate() (in module pygal.test.test\_colors), 78

test\_round\_to\_float() (in module pygal.test.test\_util), 85

test\_round\_to\_int() (in module pygal.test.test\_util), 85

test\_same\_max\_and\_relative\_values\_sparktext() (in module pygal.test.test\_sparktext), 84

test\_saturate() (in module pygal.test.test\_colors), 78

test\_secondary() (in module pygal.test.test\_graph), 82

test\_serie\_config() (in module pygal.test.test\_serie\_config), 84

test\_serie\_precedence\_over\_global\_config() (in module pygal.test.test\_serie\_config), 84

test\_shifted\_sparktext() (in module pygal.test.test\_sparktext), 84

test\_show\_dots() (in module pygal.test.test\_config), 80

test\_show\_legend() (in module pygal.test.test\_config), 80

test\_significant() (in module pygal.test.test\_formatters), 81

test\_simple\_bar() (in module pygal.test.test\_bar), 76

test\_simple\_box() (in module pygal.test.test\_box), 76

test\_simple\_line() (in module pygal.test.test\_line), 83

test\_sparkline() (in module pygal.test.test\_graph), 82

test\_stacked\_line() (in module pygal.test.test\_stacked), 84

test\_stacked\_line\_interpolate() (in module pygal.test.test\_stacked), 84

test\_stacked\_line\_log() (in module pygal.test.test\_stacked), 84

test\_stacked\_line\_reverse() (in module pygal.test.test\_stacked), 84

test\_swap\_curly() (in module pygal.test.test\_util), 85

test\_time() (in module pygal.test.test\_date), 80

test\_timedelta() (in module pygal.test.test\_date), 80

test\_trigonometric() (in module pygal.test.test\_interpolate), 82

test\_truncate() (in module pygal.test.test\_util), 85

test\_unicode\_labels\_decode() (in module pygal.test.test\_graph), 82

- test\_unicode\_labels\_python2() (in module pygal.test.test\_graph), 82
- test\_unicode\_labels\_python3() (in module pygal.test.test\_graph), 82
- test\_unparse\_color() (in module pygal.test.test\_colors), 78
- test\_utc\_timestamping() (in module pygal.test.test\_date), 80
- test\_value\_formatter() (in module pygal.test.test\_config), 80
- test\_values\_by\_dict() (in module pygal.test.test\_graph), 82
- test\_x\_label\_major() (in module pygal.test.test\_config), 80
- test\_x\_y\_title() (in module pygal.test.test\_config), 80
- test\_xml\_filters\_change\_bars() (in module pygal.test.test\_xml\_filters), 85
- test\_xml\_filters\_round\_trip() (in module pygal.test.test\_xml\_filters), 85
- test\_y\_label\_major() (in module pygal.test.test\_config), 80
- text\_len() (in module pygal.util), 101
- texts() (in module pygal.test.utils), 85
- time\_to\_datetime() (in module pygal.graph.time), 75
- time\_to\_seconds() (in module pygal.graph.time), 75
- timedelta\_to\_seconds() (in module pygal.graph.time), 75
- TimeDeltaLine (class in pygal.graph.time), 75
- TimeLine (class in pygal.graph.time), 75
- title (pygal.config.Config attribute), 88
- title (pygal.config.SerieConfig attribute), 90
- title\_font\_family (pygal.style.Style attribute), 97
- title\_font\_size (pygal.style.Style attribute), 97
- to\_dict() (pygal.config.BaseConfig method), 87
- to\_dict() (pygal.style.Style method), 97
- to\_etree() (pygal.etree.Etree method), 90
- to\_lxml() (pygal.etree.Etree method), 90
- tooltip\_border\_radius (pygal.config.Config attribute), 88
- tooltip\_fancy\_mode (pygal.config.Config attribute), 88
- tooltip\_font\_family (pygal.style.Style attribute), 98
- tooltip\_font\_size (pygal.style.Style attribute), 98
- transition (pygal.style.BlueStyle attribute), 92
- transition (pygal.style.DarkColorizedStyle attribute), 93
- transition (pygal.style.DarkGreenBlueStyle attribute), 93
- transition (pygal.style.DarkGreenStyle attribute), 94
- transition (pygal.style.DarkSolarizedStyle attribute), 94
- transition (pygal.style.DarkStyle attribute), 94
- transition (pygal.style.LightColorizedStyle attribute), 95
- transition (pygal.style.LightGreenStyle attribute), 95
- transition (pygal.style.NeonStyle attribute), 96
- transition (pygal.style.SolidColorStyle attribute), 97
- transition (pygal.style.Style attribute), 98
- transition (pygal.style.TurquoiseStyle attribute), 98
- transposable\_node() (pygal.svg.Svg method), 99
- Treemap (class in pygal.graph.treemap), 75
- trigonometric\_interpolate() (in module pygal.interpolate), 91
- truncate() (in module pygal.util), 101
- truncate\_label (pygal.config.Config attribute), 89
- truncate\_legend (pygal.config.Config attribute), 89
- TurquoiseStyle (class in pygal.style), 98
- ## U
- unparse\_color() (in module pygal.colors), 86
- ## V
- value\_background (pygal.style.Style attribute), 98
- value\_colors (pygal.style.Style attribute), 98
- value\_font\_family (pygal.style.Style attribute), 98
- value\_font\_size (pygal.style.Style attribute), 98
- value\_formatter (pygal.config.Config attribute), 89
- value\_label\_font\_family (pygal.style.Style attribute), 98
- value\_label\_font\_size (pygal.style.Style attribute), 98
- VerticalPyramid (class in pygal.graph.pyramid), 74
- View (class in pygal.view), 103
- ## W
- width (pygal.config.Config attribute), 89
- width (pygal.view.Box attribute), 101
- ## X
- x (pygal.view.Margin attribute), 102
- x() (pygal.view.HorizontalLogView method), 102
- x() (pygal.view.HorizontalView method), 102
- x() (pygal.view.View method), 103
- x() (pygal.view.XLogView method), 103
- x\_label\_rotation (pygal.config.Config attribute), 89
- x\_labels (pygal.config.Config attribute), 89
- x\_labels\_major (pygal.config.Config attribute), 89
- x\_labels\_major\_count (pygal.config.Config attribute), 89
- x\_labels\_major\_every (pygal.config.Config attribute), 89
- x\_title (pygal.config.Config attribute), 89
- x\_value\_formatter (pygal.config.Config attribute), 89
- xlink\_ns (pygal.svg.Svg attribute), 99
- XLogView (class in pygal.view), 103
- xmax (pygal.view.Box attribute), 101
- xmin (pygal.view.Box attribute), 101
- xrange (pygal.config.Config attribute), 89
- xvals (pygal.graph.histogram.Histogram attribute), 72
- xvals (pygal.graph.xy.XY attribute), 75
- XY (class in pygal.graph.xy), 75
- XYLogView (class in pygal.view), 103
- ## Y
- y (pygal.view.Margin attribute), 102
- y() (pygal.view.HorizontalLogView method), 102
- y() (pygal.view.HorizontalView method), 102
- y() (pygal.view.LogView method), 102

y() (pygal.view.ReverseView method), 102  
y() (pygal.view.View method), 103  
y\_label\_rotation (pygal.config.Config attribute), 89  
y\_labels (pygal.config.Config attribute), 89  
y\_labels\_major (pygal.config.Config attribute), 89  
y\_labels\_major\_count (pygal.config.Config attribute), 89  
y\_labels\_major\_every (pygal.config.Config attribute), 89  
y\_title (pygal.config.Config attribute), 89  
ymax (pygal.view.Box attribute), 101  
ymin (pygal.view.Box attribute), 101  
yvals (pygal.graph.histogram.Histogram attribute), 72  
yvals (pygal.graph.xy.XY attribute), 75

## Z

zero (pygal.config.Config attribute), 89