

CeTZ Plot

Johannes Wolf
fenjalien

Version 0.1.0

1 Introduction	3
2 Usage	3
3 Plot	3
3.0.1 plot	3
3.0.2 add-anchor	7
3.0.3 add	7
3.0.4 add-hline	10
3.0.5 add-vline	10
3.0.6 add-fill-between	11
3.0.7 add-bar	12
3.0.8 add-boxwhisker	14
3.0.9 add-contour	15
3.0.10 add-errorbar	16
3.0.11 annotate	17
3.0.12 fraction	18
3.0.13 multiple-of	19
3.0.14 sci	20
3.0.15 decimal	20
3.0.16 add-violin	21
3.0.17 item	22
3.0.18 legend	23
3.0.19 add-legend	23
3.1 Styling	23
3.1.1 default-style	23
3.1.2 Example	25
4 Chart	25
4.0.1 barchart	25
4.0.2 boxwhisker	27
4.0.3 columnchart	28
4.0.4 piechart	29
4.0.5 pyramid	32
4.1 Processes	34
4.1.1 CHEVRON-CAPS	34
4.1.2 basic	34
4.1.3 chevron	36
4.1.4 bending	38

1 Introduction

CeTZ-Plot is a simple plotting library for use with CeTZ.

2 Usage

This is the minimal starting point:

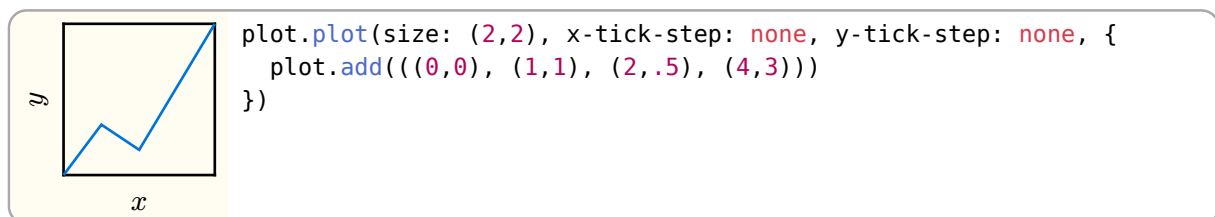
```
#import "@preview/cetz:0.3.1"
#import "@preview/cetz-plot:0.1.0"
#cezt.canvas({
  import cetz.draw: *
  import cetz-plot: *
  ...
})
```

Note that plot functions are imported inside the scope of the canvas block. All following example code is expected to be inside a canvas block, with the plot module imported into the namespace.

3 Plot

3.0.1 plot

Create a plot environment. Data to be plotted is given by passing it to the `plot.add` or other plotting functions. The plot environment supports different axis styles to draw, see its parameter `axis-style`.



To draw elements inside a plot, using the plots coordinate system, use the `plot.annotate(..)` function.

Parameters

```
plot(
  body: body,
  size: array,
  axis-style: none string,
  name: string,
  plot-style: style function,
  mark-style: style function,
  fill-below: bool,
  legend: none auto coordinate,
  legend-anchor: auto string,
  legend-style: style,
  ..options: any
)
```

body: body

Calls of `plot.add` or `plot.add-*` commands. Note that normal drawing commands like `line` or `rect` are not allowed inside the plots body, instead wrap them in `plot.annotate`, which lets you select the axes used for drawing.

size: array

Default: (1, 1)

Plot size tuple of (<width>, <height>) in canvas units. This is the plots inner plotting size without axes and labels.

axis-style: `none` or `string`

Default: `"scientific"`

How the axes should be styled:

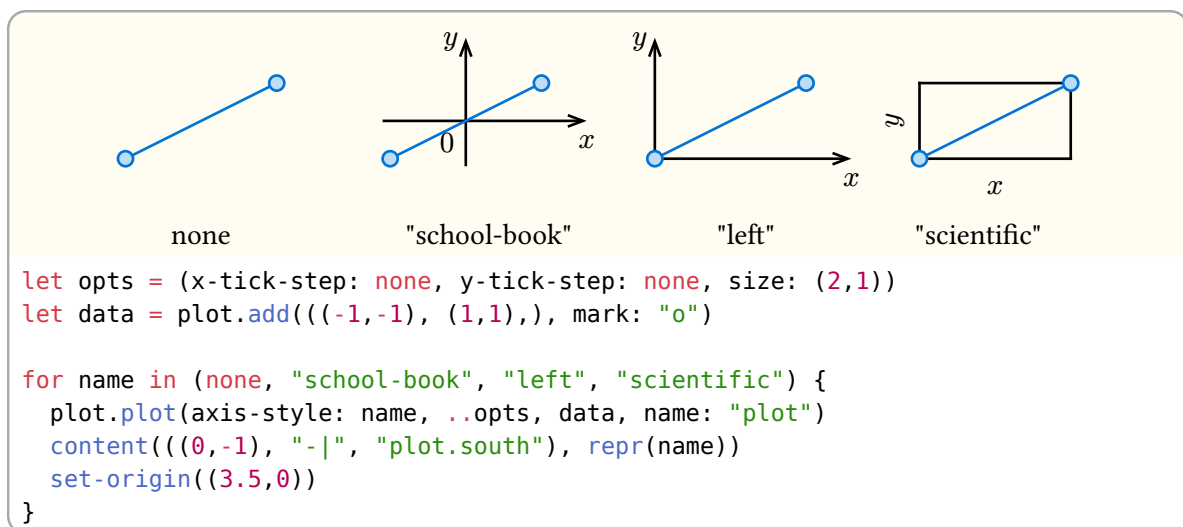
scientific Frames plot area using a rectangle and draw axes x (bottom), y (left), x2 (top), and y2 (right) around it. If x2 or y2 are unset, they mirror their opposing axis.

scientific-auto Draw set (used) axes x (bottom), y (left), x2 (top) and y2 (right) around the plotting area, forming a rect if all axes are in use or a L-shape if only x and y are in use.

school-book Draw axes x (horizontal) and y (vertical) as arrows pointing to the right/top with both crossing at (0, 0)

left Draw axes x and y as arrows, while the y axis stays on the left (at x.min) and the x axis at the bottom (at y.min)

none Draw no axes (and no ticks).



name: `string`

Default: `none`

The plots element name to be used when referring to anchors

plot-style: `style` or `function`

Default: `default-plot-style`

Styling to use for drawing plot graphs. This style gets inherited by all plots and supports palette functions. The following style keys are supported:

stroke: `none` or `stroke`

Default: `1pt`

Stroke style to use for stroking the graph.

fill: `none` or `paint`

Default: `none`

Paint to use for filled graphs. Note that not all graphs may support filling and that you may have to enable filling per graph, see `plot.add(fill: ..)`.

mark-style: `style` or `function`

Default: `default-mark-style`

Styling to use for drawing plot marks. This style gets inherited by all plots and supports palette functions. The following style keys are supported:

stroke: `none` or `stroke`

Default: `1pt`

Stroke style to use for stroking the mark.

fill: `none` or `paint`

Default: `none`

Paint to use for filling marks.

fill-below: `bool`

Default: `true`

If true, the filled shape of plots is drawn *below* axes.

legend: `none` or `auto` or `coordinate`

Default: `auto`

The position the legend will be drawn at. See plot-legends for information about legends. If set to `<auto>`, the legend's "default-placement" styling will be used. If set to a `<coordinate>`, it will be taken as relative to the plot's origin.

legend-anchor: `auto` or `string`

Default: `auto`

Anchor of the legend group to use as its origin. If set to `auto` and `legend` is one of the predefined legend anchors, the opposite anchor to `legend` gets used.

legend-style: `style`

Default: `(:)`

Style key-value overwrites for the legend style with style root `legend`.

..options: `any`

Axis options, see *options* below.

Options

You can use the following options to customize each axis of the plot. You must pass them as named arguments prefixed by the axis name followed by a dash (-) they should target. Example: `x-min: 0`, `y-ticks: (...)` or `x2-label: [...]`.

label: `none` or `content`

Default: `"none"`

The axis' label. If and where the label is drawn depends on the `axis-style`.

min: `auto` or `float`

Default: `"auto"`

Axis lower domain value. If this is set greater than `max`, the axis' direction is swapped

max: `auto` or `float`

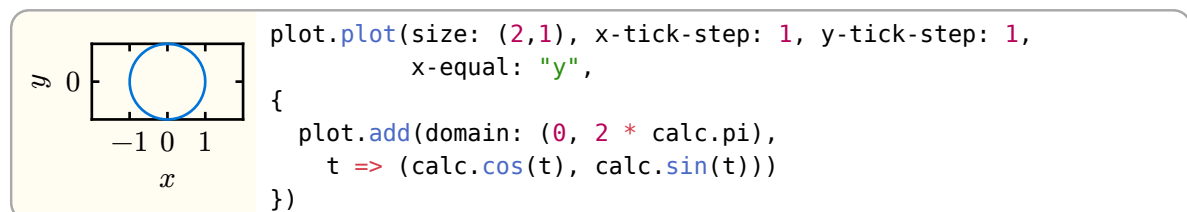
Default: `"auto"`

Axis upper domain value. If this is set to a lower value than `min`, the axis' direction is swapped

equal: `string`

Default: `"none"`

Set the axis domain to keep a fixed aspect ratio by multiplying the other axis domain by the plots aspect ratio, depending on the other axis orientation (see `horizontal`). This can be useful to force one axis to grow or shrink with another one. You can only "lock" two axes of different orientations.



horizontal: `bool`

Default: `"axis name dependant"`

If true, the axis is considered an axis that gets drawn horizontally, vertically otherwise. The default value depends on the axis name on axis creation. Axes which name start with `x` have this set to `true`, all others have it set to `false`. Each plot has to use one horizontal and one vertical axis for plotting, a combination of two y-axes will panic: ("`y`", "`y2`").

tick-step: `none` or `auto` or `float`

Default: `"auto"`

The increment between tick marks on the axis. If set to auto, an increment is determined. When set to none, incrementing tick marks are disabled.

minor-tick-step: none or float

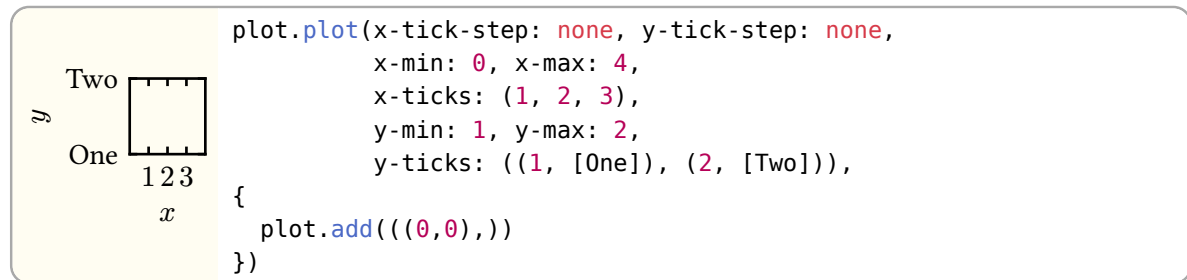
Default: "none"

Like tick-step, but for minor tick marks. In contrast to ticks, minor ticks do not have labels.

ticks: none or array

Default: "none"

A List of custom tick marks to additionally draw along the axis. They can be passed as an array of <float> values or an array of (<float>, <content>) tuples for setting custom tick mark labels per mark.



Examples: (1, 2, 3) or ((1, [One]), (2, [Two]), (3, [Three]))

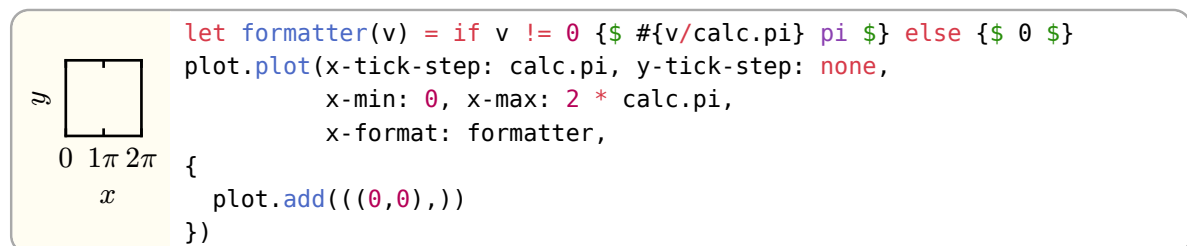
format: none or string or function

Default: "float"

How to format the tick label: You can give a function that takes a <float> and return <content> to use as the tick label. You can also give one of the predefined options:

float Floating point formatting rounded to two digits after the point (see decimals)

sci Scientific formatting with $\times 10^n$ used as exponent syntax



decimals: int

Default: "2"

Number of decimals digits to display for tick labels, if the format is set to "float".

mode: none or string

Default: "none"

The scaling function of the axis. Takes lin (default) for linear scaling, and log for logarithmic scaling.

base: none or number

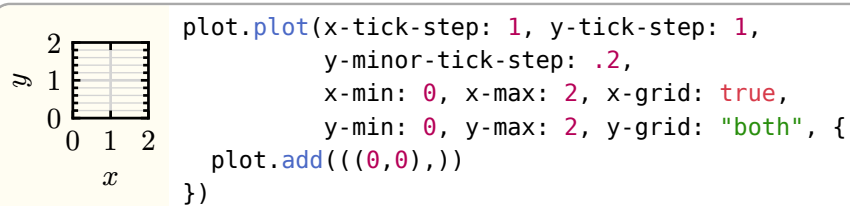
Default: "none"

The base to be used when labeling axis ticks in logarithmic scaling

grid: bool or string

Default: "false"

If true or "major", show grid lines for all major ticks. If set to "minor", show grid lines for minor ticks only. The value "both" enables grid lines for both, major- and minor ticks.

**break:** bool

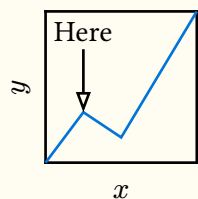
Default: "false"

If true, add a “sawtooth” at the start or end of the axis line, depending on the axis bounds. If the axis min. value is > 0 , a sawtooth is added to the start of the axes, if the axis max. value is < 0 , a sawtooth is added to its end.

3.0.2 add-anchor

Add an anchor to a plot environment

This function is similar to `draw.anchor` but it takes an additional axis tuple to specify which axis coordinate system to use.



```

plot.plot(size: (2,2), name: "plot",
          x-tick-step: none, y-tick-step: none, {
    plot.add(((0,0), (1,1), (2,.5), (4,3)))
    plot.add-anchor("pt", (1,1))
  })

line("plot.pt", (((), "|-", (0,1.5))), mark: (start: ">"), name: "line")
content("line.end", [Here], anchor: "south", padding: .1)

```

Parameters

```

add-anchor(
  name: string,
  position: tuple,
  axes: tuple
)

```

name: string

Anchor name

position: tuple

Tuple of x and y values. Both values can have the special values “min” and “max”, which resolve to the axis min/max value. Position is in axis space defined by the axes passed to axes.

axes: tuple

Default: ("x", "y")

Name of the axes to use ("x", "y") as coordinate system for position. Note that both axes must be used, as add-anchors does not create them on demand.

3.0.3 add

Add data to a plot environment.

Note: You can use this for scatter plots by setting the stroke style to none: `add(..., style: (stroke: none))`.

Must be called from the body of a `plot(...)` command.

Parameters

```

add(
  domain: domain,
  hypograph: bool,
  epigraph: bool,
  fill: bool,
  fill-type: string,
  style: style,
  mark: string,
  mark-size: float,
  mark-style,
  samples: int,
  sample-at: array,
  line: string dictionary,
  axes: axes,
  label: none content,
  data: array function
)

```

domain: domain	Default: auto
Domain of data, if data is a function. Has no effect if data is not a function.	
hypograph: bool	Default: false
Fill hypograph; uses the hypograph style key for drawing	
epigraph: bool	Default: false
Fill epigraph; uses the epigraph style key for drawing	
fill: bool	Default: false
Fill the shape of the plot	
fill-type: string	Default: "axis"
Fill type:	
"axis" Fill the shape to $y = 0$	
"shape" Fill the complete shape	
style: style	Default: (:)
Style to use, can be used with a palette function	
mark: string	Default: none
Mark symbol to place at each distinct value of the graph. Uses the mark style key of style for drawing.	
mark-size: float	Default: .2
Mark size in canvas units	
mark-style:	Default: (:)
samples: int	Default: 50
Number of times the data function gets called for sampling y-values. Only used if data is of type function. This parameter gets passed onto sample-fn.	
sample-at: array	Default: ()

Array of x-values the function gets sampled at in addition to the default sampling. This parameter gets passed to `sample-fn`.

line: `string` or `dictionary`

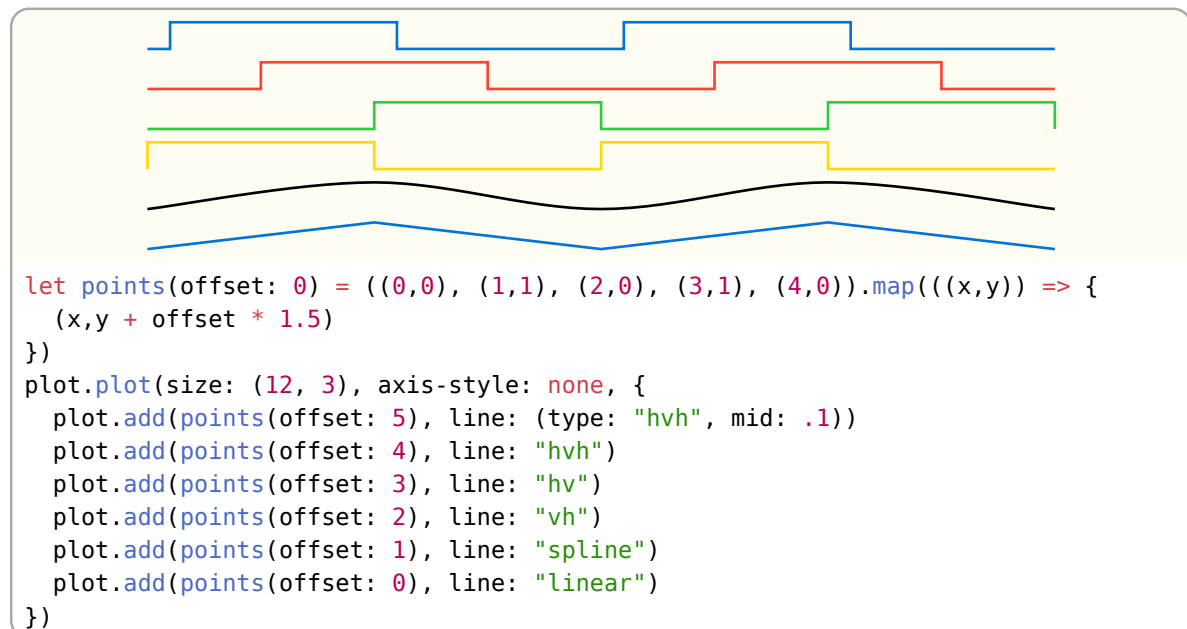
Default: `"raw"`

Line type to use. The following types are supported:

- `"raw"` Plot raw data
- `"linear"` Linearize data
- `"spline"` Calculate a Catmull-Rom curve through all points
- `"vh"` Move vertical and then horizontal
- `"hv"` Move horizontal and then vertical
- `"hvh"` Add a vertical step in the middle

If the value is a dictionary, the type must be supplied via the `type` key. The following extra attributes are supported:

- `"samples"` `<int>` Samples of splines
- `"tension"` `<float>` Tension of splines
- `"mid"` `<float>` Mid-Point of hvh lines (0 to 1)
- `"epsilon"` `<float>` Linearization slope epsilon for use with `"linear"`, defaults to 0.



axes: `axes`

Default: `("x", "y")`

Name of the axes to use for plotting. Reversing the axes means rotating the plot by 90 degrees.

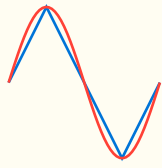
label: `none` or `content`

Default: `none`

Legend label to show for this plot.

data: `array` or `function`

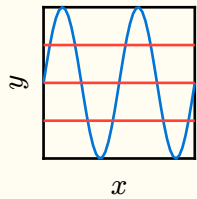
Array of 2D data points (numeric) or a function of the form $x \Rightarrow y$, where x is a value in domain and y must be numeric or a 2D vector (for parametric functions).



```
plot.plot(size: (2, 2), axis-style: none, {
  // Using an array of points:
  plot.add(((0,0), (calc.pi/2,1),
              (1.5*calc.pi,-1), (2*calc.pi,0)))
  // Sampling a function:
  plot.add(domain: (0, 2*calc.pi), calc.sin)
})
```

3.0.4 add-hline

Add horizontal lines at one or more y-values. Every lines start and end points are at their axis bounds.



```
plot.plot(size: (2,2), x-tick-step: none, y-tick-step: none, {
  plot.add(domain: (0, 4*calc.pi), calc.sin)
  // Add 3 horizontal lines
  plot.add-hline(-.5, 0, .5)
})
```

Parameters

```
add-hline(
  ..y: float,
  min: auto float,
  max: auto float,
  axes: array,
  style: style,
  label: none content
)
```

..y: float

Y axis value(s) to add a line at

min: auto or float

Default: auto

X axis minimum value or auto to take the axis minimum

max: auto or float

Default: auto

X axis maximum value or auto to take the axis maximum

axes: array

Default: ("x", "y")

Name of the axes to use for plotting

style: style

Default: (:)

Style to use, can be used with a palette function

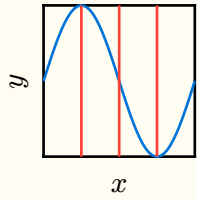
label: none or content

Default: none

Legend label to show for this plot.

3.0.5 add-vline

Add vertical lines at one or more x-values. Every lines start and end points are at their axis bounds.



```
plot.plot(size: (2,2), x-tick-step: none, y-tick-step: none, {
  plot.add(domain: (0, 2*calc.pi), calc.sin)
  // Add 3 vertical lines
  plot.add-vline(calc.pi/2, calc.pi, 3*calc.pi/2)
})
```

Parameters

```
add-vline(
  ..x: float,
  min: auto float,
  max: auto float,
  axes: array,
  style: style,
  label: none content
)
```

..x: float

X axis values to add a line at

min: auto or float

Default: auto

Y axis minimum value or auto to take the axis minimum

max: auto or float

Default: auto

Y axis maximum value or auto to take the axis maximum

axes: array

Default: ("x", "y")

Name of the axes to use for plotting, note that not all plot styles are able to display a custom axis!

style: style

Default: (:)

Style to use, can be used with a palette function

label: none or content

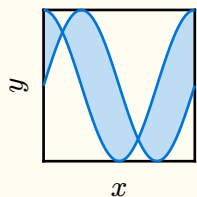
Default: none

Legend label to show for this plot.

3.0.6 add-fill-between

Fill the area between two graphs. This behaves same as add but takes a pair of data instead of a single data array/function. The area between both function plots gets filled. For a more detailed explanation of the arguments, see [add\(\)](#).

This can be used to display an error-band of a function.



```
plot.plot(size: (2,2), x-tick-step: none, y-tick-step: none, {
  plot.add-fill-between(domain: (0, 2*calc.pi),
    calc.sin, // First function/data
    calc.cos) // Second function/data
})
```

Parameters

```

add-fill-between(
  data-a: array function ,
  data-b: array function ,
  domain: domain ,
  samples: int ,
  sample-at: array ,
  line: string dictionary ,
  axes: array ,
  label: none content ,
  style: style
)

```

data-a: array or functionData of the first plot, see `add()`.**data-b:** array or functionData of the second plot, see `add()`.**domain:** domainDefault: `auto`

Domain of both data-a and data-b. The domain is used for sampling functions only and has no effect on data arrays.

samples: intDefault: `50`

Number of times the data-a and data-b function gets called for sampling y-values. Only used if data-a or data-b is of type function.

sample-at: arrayDefault: `()`

Array of x-values the function(s) get sampled at in addition to the default sampling.

line: string or dictionaryDefault: `"raw"`Line type to use, see `add()`.**axes:** arrayDefault: `("x", "y")`

Name of the axes to use for plotting.

label: none or contentDefault: `none`

Legend label to show for this plot.

style: styleDefault: `(:)`

Style to use, can be used with a palette function.

3.0.7 add-bar

Add a bar- or column-chart to the plot

A bar- or column-chart is a chart where values are drawn as rectangular boxes.

Parameters

```

add-bar(
  data: array,
  x-key,
  y-key,
  error-key,
  mode: string,
  labels: none content array,
  bar-width: float,
  bar-position: string,
  cluster-gap: float,
  whisker-size,
  error-style,
  style: dictionary,
  axes: axes
)

```

data: array

Array of data items. An item is an array containing a x and one or more y values. For example (0, 1) or (0, 10, 5, 30). Depending on the mode, the data items get drawn as either clustered or stacked rects.

- **x-key:** (int,string): Key to use for retrieving a bars x-value from a single data entry. This value gets passed to the `.at(...)` function of a data item.
- **y-key:** (auto,int,string,array): Key to use for retrieving a bars y-value. For clustered/stacked data, this must be set to a list of keys (e.g. `range(1, 4)`). If set to auto, all but the first array-values of a data item are used as y-values.
- **error-key:** (none,int,string,array): Key(s) to use for retrieving a bars y-error.

x-key: Default: 0**y-key:** Default: auto**error-key:** Default: none**mode:** string Default: "basic"

The mode on how to group data items into bars:

basic Add one bar per data value. If the data contains multiple values, group those bars next to each other.

clustered Like “basic”, but take into account the maximum number of values of all items and group each cluster of bars together having the width of the widest cluster.

stacked Stack bars of subsequent item values onto the previous bar, generating bars with the height of the sum of all an items values.

stacked100 Like “stacked”, but scale each bar to height 100, making the different bars percentages of the sum of an items values.

labels: none or content or array Default: none

A single legend label for “basic” bar-charts, or a a list of legend labels per bar category, if the mode is one of “clustered”, “stacked” or “stacked100”.

bar-width: float Default: 1

Width of one data item on the y axis

bar-position: `string` Default: `"center"`

Positioning of data items relative to their x value.

- “start”: The lower edge of the data item is on the x value (left aligned)
- “center”: The data item is centered on the x value
- “end”: The upper edge of the data item is on the x value (right aligned)

cluster-gap: `float` Default: `0`

Spacing between bars insides a cluster.

whisker-size: Default: `.25`

error-style: Default: `(:)`

style: `dictionary` Default: `(:)`

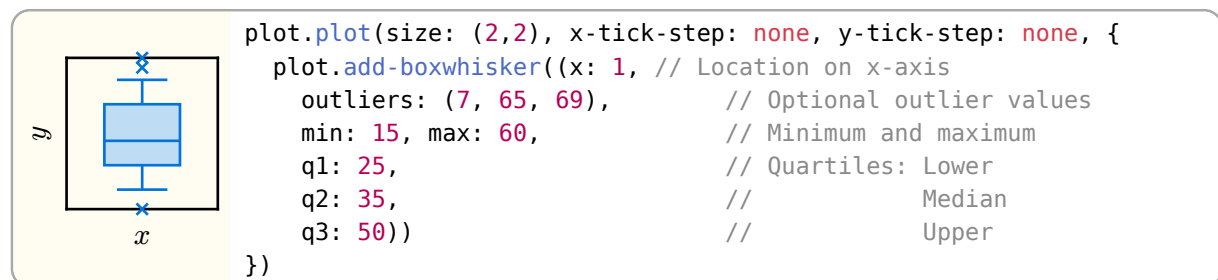
Plot style

axes: `axes` Default: `("x", "y")`

Plot axes. To draw a horizontal growing bar chart, you can swap the x and y axes.

3.0.8 add-boxwhisker

Add one or more box or whisker plots



Parameters

```

add-boxwhisker(
  data: array dictionary,
  label: none content,
  axes: array,
  style: style,
  box-width: float,
  whisker-width: float,
  mark: string,
  mark-size: float
)
  
```

data: `array` or `dictionary`

dictionary or array of dictionaries containing the needed entries to plot box and whisker plot.

The following fields are supported:

- x (number) X-axis value
- min (number) Minimum value
- max (number) Maximum value

- q1, q2, q3 (number) Quartiles from lower to upper
- outliers (array of number) Optional outliers

label: `none` or `content` Default: `none`

Legend label to show for this plot.

axes: `array` Default: `("x", "y")`

Name of the axes to use ("x", "y"), note that not all plot styles are able to display a custom axis!

style: `style` Default: `(:)`

Style to use, can be used with a palette function

box-width: `float` Default: `0.75`

Width from edge-to-edge of the box of the box and whisker in plot units. Defaults to 0.75

whisker-width: `float` Default: `0.5`

Width from edge-to-edge of the whisker of the box and whisker in plot units. Defaults to 0.5

mark: `string` Default: `"*"`

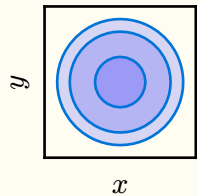
Mark to use for plotting outliers. Set none to disable. Defaults to "x"

mark-size: `float` Default: `0.15`

Size of marks for plotting outliers. Defaults to 0.15

3.0.9 add-contour

Add a contour plot of a sampled function or a matrix.



```
plot.plot(size: (2,2), x-tick-step: none, y-tick-step: none, {
  plot.add-contour(x-domain: (-3, 3), y-domain: (-3, 3),
    style: (fill: rgb(50,50,250,50)),
    fill: true,
    op: "<", // Find contours where data < z
    z: (2.5, 2, 1), // Z values to find contours for
    (x, y) => calc.sqrt(x * x + y * y))
})
```

Parameters

```
add-contour(
  data: array function,
  label: none content,
  z: float array,
  x-domain: domain,
  y-domain: domain,
  x-samples: int,
  y-samples: int,
  interpolate: bool,
  op: auto string function,
  axes: axes,
  style: style,
  fill: bool,
  limit: int
)
```

data: `array` or `function`

A function of the signature $(x, y) \Rightarrow z$ or an array of arrays of floats (a matrix) where the first index is the row and the second index is the column.

label: `none` or `content` Default: `none`

Plot legend label to show. The legend preview for contour plots is a little rectangle drawn with the contours style.

z: `float` or `array` Default: `(1,)`

Z values to plot. Contours containing values above z ($z \geq 0$) or below z ($z < 0$) get plotted. If you specify multiple z values, they get plotted in the order of specification.

x-domain: `domain` Default: `(0, 1)`

X axis domain used if `data` is a function, that is the domain inside the function gets sampled.

y-domain: `domain` Default: `(0, 1)`

Y axis domain used if `data` is a function, see `x-domain`.

x-samples: `int` Default: `25`

X axis domain samples ($2 < n$). Note that contour finding can be quite slow. Using a big sample count can improve accuracy but can also lead to bad compilation performance.

y-samples: `int` Default: `25`

Y axis domain samples ($2 < n$)

interpolate: `bool` Default: `true`

Use linear interpolation between sample values which can improve the resulting plot, especially if the contours are curved.

op: `auto` or `string` or `function` Default: `auto`

Z value comparison operator:

`">"`, `">="`, `"<"`, `"<="`, `"!="`, `"=="` Use the operator for comparison of z to the values from data.

auto Use `">="` for positive z values, `"<="` for negative z values.

<function> Call comparison function of the format $(\text{plot-}z, \text{data-}z) \Rightarrow \text{boolean}$, where `plot-z` is the z -value from the plots z argument and `data-z` is the z -value of the data getting plotted. The function must return true if at the combinations of arguments a contour is detected.

axes: `axes` Default: `("x", "y")`

Name of the axes to use for plotting.

style: `style` Default: `(:)`

Style to use for plotting, can be used with a palette function. Note that all z -levels use the same style!

fill: `bool` Default: `false`

Fill each contour

limit: `int` Default: `50`

Limit of contours to create per z value before the function panics

3.0.10 add-errorbar

Add x- and/or y-error bars

Parameters

```
add-errorbar(
    pt: tuple,
    x-error,
    y-error,
    label,
    mark,
    mark-size,
    mark-style,
    whisker-size: float,
    style: dictionary,
    axes: axes
)
```

pt: tuple

Error-bar center coordinate tuple: (x, y)

- x-error: (float,tuple): Single error or tuple of errors along the x-axis
- y-error: (float,tuple): Single error or tuple of errors along the y-axis
- mark: (none,string): Mark symbol to show at the error position (pt).
- mark-size: (number): Size of the mark symbol.
- mark-style: (style): Extra style to apply to the mark symbol.

x-error: Default: 0**y-error:** Default: 0**label:** Default: none**mark:** Default: "o"**mark-size:** Default: .2**mark-style:** Default: (:)**whisker-size:** float Default: .5

Width of the error bar whiskers in canvas units.

style: dictionary Default: (:)

Style for the error bars

- label: (none,content): Label to tsh

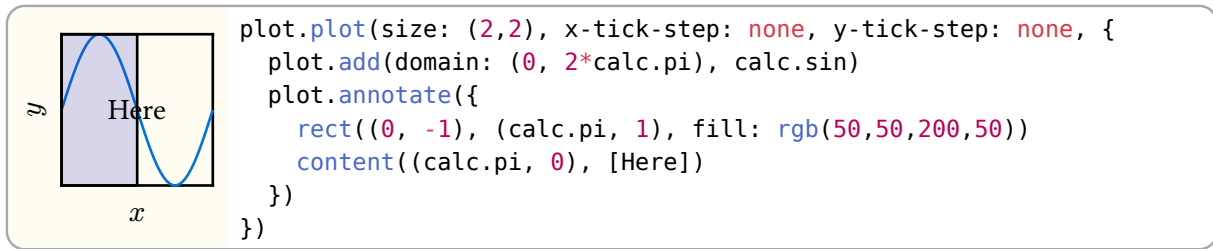
axes: axes Default: ("x", "y")

Plot axes. To draw a horizontal growing bar chart, you can swap the x and y axes.

3.0.11 annotate

Add an annotation to the plot

An annotation is a sub-canvas that uses the plots coordinates specified by its x and y axis.



Bounds calculation is done naively, therefore fixed size content *can* grow out of the plot. You can adjust the padding manually to adjust for that. The feature of solving the correct bounds for fixed size elements might be added in the future.

Parameters

```

annotate(
  body: drawable,
  axes: axes,
  resize: bool,
  padding: none number dictionary,
  background: bool
)

```

body: drawable

Elements to draw

axes: axes

Default: ("x", "y")

X and Y axis names

resize: bool

Default: true

If true, the plots axes get adjusted to contain the annotation

padding: none or number or dictionary

Default: none

Annotation padding that is used for axis adjustment

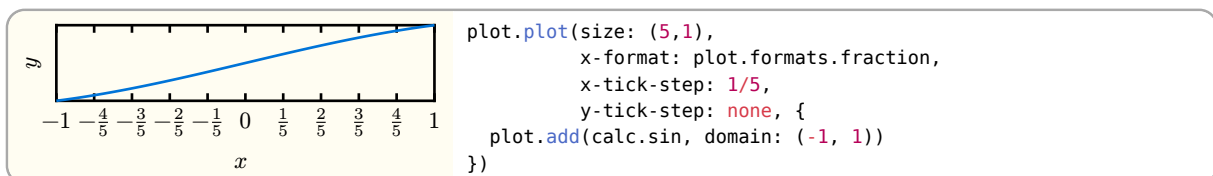
background: bool

Default: false

If true, the annotation is drawn behind all plots, in the background. If false, the annotation is drawn above all plots.

3.0.12 fraction

Fraction tick formatter



Parameters

```

fraction(
  value: number,
  denom: auto int,
  eps: number
) -> Content if a matching fraction could be found or none

```

value: number

Value to format

denom: `auto` or `int`Default: `auto`

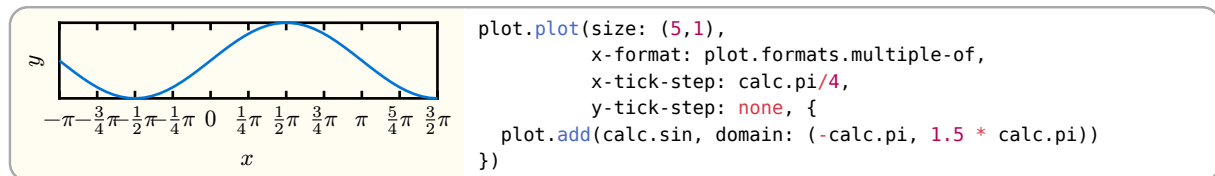
Denominator for result fractions. If set to `auto`, a hardcoded fraction table is used for finding fractions with a denominator ≤ 11 .

eps: `number`Default: `1e-6`

Epsilon used for comparison

3.0.13 multiple-of

Multiple of tick formatter



Parameters

```
multiple-of(
  value: number,
  factor: number,
  symbol: content,
  fraction: none true int,
  digits: int,
  eps: number,
  prefix: content,
  suffix: content
) -> Content if a matching fraction could be found or none
```

value: `number`

Value to format

factor: `number`Default: `calc.pi`

Factor value is expected to be a multiple of.

symbol: `content`Default: `pi`

Suffix symbol. For value = 0, the symbol is not appended.

fraction: `none` or `true` or `int`Default: `true`

If not `none`, try finding matching fractions using the same mechanism as `fraction`. If set to an integer, that integer is used as denominator. If set to `none` or `false`, or if no fraction could be found, a real number with `digits` digits is used.

digits: `int`Default: `2`

Number of digits to use for rounding

eps: `number`Default: `1e-6`

Epsilon used for comparison

prefix: `content`Default: `[]`

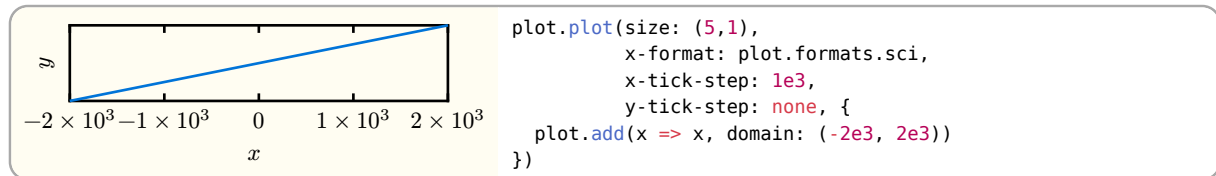
Content to prefix

suffix: `content`Default: `[]`

Content to append

3.0.14 sci

Scientific notation tick formatter



Parameters

```
sci(
  value: number,
  digits: int,
  prefix: content,
  suffix: content
) -> Content
```

value: number

Value to format

digits: int

Default: 2

Number of digits for rounding the factor

prefix: content

Default: []

Content to prefix

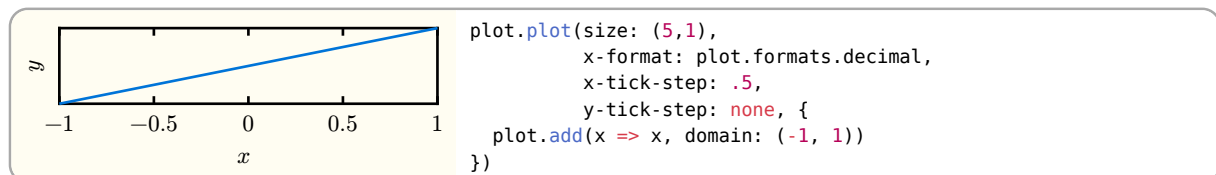
suffix: content

Default: []

Content to append

3.0.15 decimal

Rounded decimal number formatter



Parameters

```
decimal(
  value: number,
  digits: int,
  prefix: content,
  suffix: content
) -> Content
```

value: number

Value to format

digits: `int`Default: `2`

Number of digits to round to

prefix: `content`Default: `[]`

Content to prefix

suffix: `content`Default: `[]`

Content to append

3.0.16 add-violin

Add a violin plot

A violin plot is a chart that can be used to compare the distribution of continuous data between categories.

Parameters

```
add-violin(
    data: array,
    x-key: int string,
    y-key: int string,
    side: string,
    kernel: function,
    bandwidth: float,
    extents: float,
    samples: int,
    style: dictionary,
    mark-style: dictionary,
    axes: axes,
    label: none content
)
```

data: `array`

Array of data items. An item is an array containing an x and one or more y values.

x-key: `int` or `string`Default: `0`

Key to use for retrieving the x position of the violin.

y-key: `int` or `string`Default: `1`

Key to use for retrieving values of points within the category.

side: `string`Default: `"right"`

The sides of the violin to be rendered:

left Plot only the left side of the violin.**right** Plot only the right side of the violin.**both** Plot both sides of the violin.**kernel:** `function`Default: `kernel-normal.with(stdev: 1.5)`

The kernel density estimator function, which takes a single x value relative to the center of a distribution (0) and normalized by the bandwidth

bandwidth: `float`Default: `1`

The smoothing parameter of the kernel.

extents: `float`Default: `0.25`

The extension of the domain, expressed as a fraction of spread.

samples: `int`

Default: `50`

The number of samples of the kernel to render.

style: `dictionary`

Default: `(:)`

Style override dictionary.

mark-style: `dictionary`

Default: `(:)`

(unused, will eventually be used to render interquartile ranges).

axes: `axes`

Default: `("x", "y")`

(unstable, documentation to follow once completed).

label: `none` or `content`

Default: `none`

The name of the category to be shown in the legend.

3.0.17 item

Construct a legend item for use with the `legend` function

Parameters

```
item(
  label: none auto content,
  preview: auto function,
  mark,
  mark-style,
  mark-size,
  ..style: styles
)
```

label: `none` or `auto` or `content`

Legend label or auto to use the enumerated default label

preview: `auto` or `function`

Legend preview icon function of the format `item => elements`. Note that the canvas bounds for drawing the preview are (0,0) to (1,1).

- `mark`: (none,string): Legend mark symbol
- `mark-style`: (none,dictionary): Mark style
- `mark-size`: (number): Mark size

mark:

Default: `none`

mark-style:

Default: `(:)`

mark-size:

Default: `1`

..style: `styles`

Style keys for the single item

3.0.18 legend

Draw a legend

Parameters

```
legend(
    position,
    items,
    name,
    ..style
)
```

position:

items:

name:

Default: "legend"

..style:

3.0.19 add-legend

Function for manually adding a legend item from within a plot environment

Parameters

```
add-legend(
    label: content,
    preview: auto function
)
```

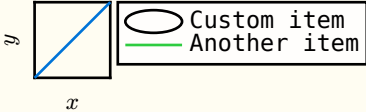
label: content

Legend label

preview: auto or function

Default: auto

Legend preview function of the format () => elements. The preview canvas bounds are between (0,0) and (1,1). If set to auto, a straight line is drawn.



```

plot.plot(size: (1,1), x-tick-step: none, y-tick-step: none, {
  plot.add(((0,0), (1,1))) // Some data
  plot.add-legend([Custom item], preview: () => {
    import cetz.draw: *
    circle(.5,.5), radius: .5 // Draw a custom preview
                                // between (0,0) and (1,1)
  })
  plot.add-legend([Another item])
})

```

3.1 Styling

You can use style root axes with the following keys:

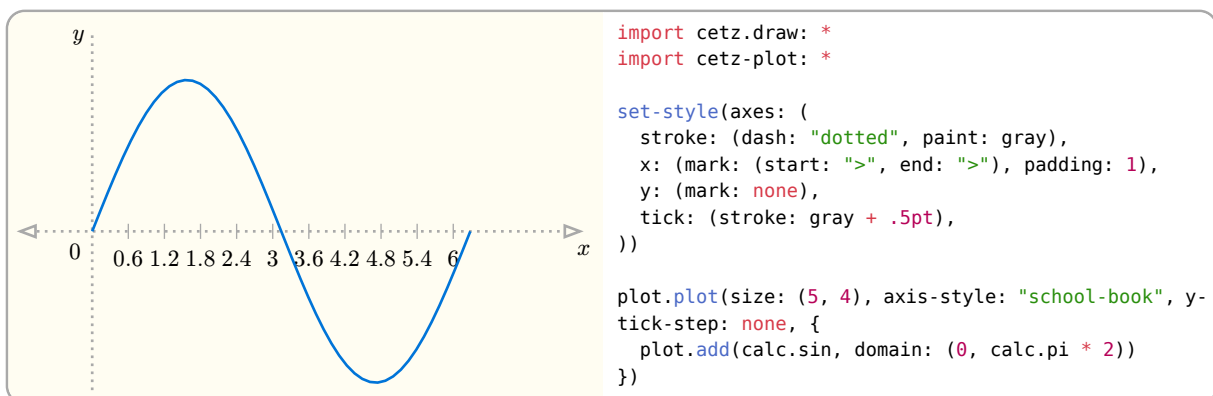
3.1.1 default-style

Default axis style

tick-limit: int	Default: 100
Upper major tick limit.	
minor-tick-limit: int	Default: 1000
Upper minor tick limit.	
auto-tick-factors: array	Default: none
List of tick factors used for automatic tick step determination.	
auto-tick-count: int	Default: none
Number of ticks to generate by default.	
stroke: stroke	Default: none
Axis stroke style.	
label.offset: number	Default: none
Distance to move axis labels away from the axis.	
label.anchor: anchor	Default: none
Anchor of the axis label to use for it's placement.	
label.angle: angle	Default: none
Angle of the axis label.	
axis-layer: float	Default: none
Layer to draw axes on (see cetz' on-layer)	
grid-layer: float	Default: none
Layer to draw the grid on (see cetz' on-layer)	
background-layer: float	Default: none
Layer to draw the background on (see cetz' on-layer)	
padding: number	Default: none
Extra distance between axes and plotting area. For schoolbook axes, this is the length of how much axes grow out of the plotting area.	
overshoot: number	Default: none
School-book style axes only: Extra length to add to the end (right, top) of axes.	
tick.stroke: stroke	Default: none
Major tick stroke style.	
tick.minor-stroke: stroke	Default: none
Minor tick stroke style.	
tick.offset: number or ratio	Default: none
Major tick offset along the tick's direction, can be relative to the length.	
tick.minor-offset: number or ratio	Default: none
Minor tick offset along the tick's direction, can be relative to the length.	
tick.length: number	Default: none
Major tick length.	

tick.minor-length: number or ratio	Default: none
Minor tick length, can be relative to the major tick length.	
tick.label.offset: number	Default: none
Major tick label offset away from the tick.	
tick.label.angle: angle	Default: none
Major tick label angle.	
tick.label.anchor: anchor	Default: none
Anchor of major tick labels used for positioning.	
tick.label.show: auto or bool	Default: auto
Set visibility of tick labels. A value of auto shows tick labels for all but mirrored axes.	
grid.stroke: stroke	Default: none
Major grid line stroke style.	
break-point.width: number	Default: none
Axis break width along the axis.	
break-point.length: number	Default: none
Axis break length.	
minor-grid.stroke: stroke	Default: none
Minor grid line stroke style.	
shared-zero: bool or content	Default: "\$0\$"
School-book style axes only: Content to display at the plots origin (0,0). If set to false, nothing is shown. Having this set, suppresses auto-generated ticks for 0!	

3.1.2 Example



4 Chart

4.0.1 barchart

Draw a bar chart. A bar chart is a chart that represents data with rectangular bars that grow from left to right, proportional to the values they represent.

Styling

Root: barchart.

bar-width: float

Default: 0.8

Width of a single bar (basic) or a cluster of bars (clustered) in the plot.

y-inset: float

Default: 1

Distance of the plot data to the plot's edges on the y-axis of the plot.

You can use any plot or axes related style keys, too.

The barchart function is a wrapper of the plot API. Arguments passed to `..plot-args` are passed to the `plot.plot` function.

Parameters

```
barchart(
  data: array,
  label-key: int string,
  value-key: int string,
  error-key: none int string array,
  mode: string,
  size: array,
  bar-style: style function,
  x-label: content none,
  x-format,
  y-label: content none,
  labels: none content,
  ..plot-args: any
)
```

data: array

Array of data rows. A row can be of type array or dictionary, with `label-key` and `value-key` being the keys to access a rows label and value(s).

Example

```
(([A], 1), ([B], 2), ([C], 3),)
```

label-key: int or string

Default: 0

Key to access the label of a data row. This key is used as argument to the rows `.at(...)` function.

value-key: int or string

Default: 1

Key(s) to access values of a data row. These keys are used as argument to the rows `.at(...)` function.

error-key: none or int or string or array

Default: none

Key(s) to access error values of a data row. These keys are used as argument to the rows `.at(...)` function.

mode: string

Default: "basic"

Chart mode:

basic Single bar per data row

clustered Group of bars per data row

stacked Stacked bars per data row

stacked100 Stacked bars per data row relative to the sum of the row

size: array

Default: (auto, 1)

Chart size as width and height tuple in canvas unist; width can be set to auto.

bar-style: style or function

Default: palette.red

Style or function (idx => style) to use for each bar, accepts a palette function.

x-label: `content` or `none` Default: `none`
 x axis label

x-format: Default: `auto`

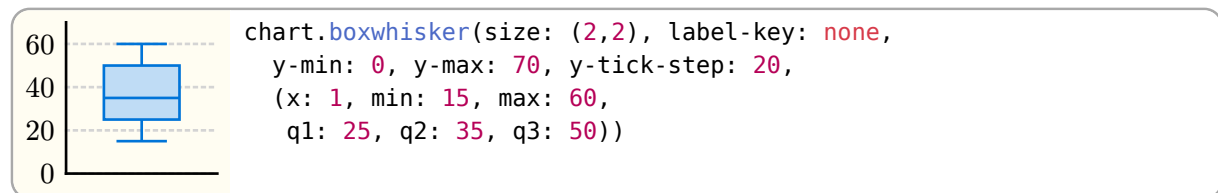
y-label: `content` or `none` Default: `none`
 Y axis label

labels: `none` or `content` Default: `none`
 Legend labels per x value group

..plot-args: `any`
 Arguments to pass to `plot.plot`

4.0.2 boxwhisker

Add one or more box or whisker plots.



Styling

Root boxwhisker

box-width: `float` Default: `0.75`
 The width of the box. Since boxes are placed 1 unit next to each other, a width of 1 would make neighbouring boxes touch.

whisker-width: `float` Default: `0.5`
 The width of the whisker, that is the horizontal bar on the top and bottom of the box.

mark-size: `float` Default: `0.15`
 The scaling of the mark for the boxes outlier values in canvas units.

You can use any plot or axes related style keys, too.

Parameters

```
boxwhisker(
  data: array dictionary,
  size,
  label-key: integer string,
  mark: string,
  ..plot-args: any
)
```

data: `array` or `dictionary`

Dictionary or array of dictionaries containing the needed entries to plot box and whisker plot.

See `plot.add-boxwhisker` for more details.

Examples:

- `(x: 1` `// Location on x-axis`
`outliers: (7, 65, 69),` `// Optional outliers`
`min: 15, max: 60` `// Minimum and maximum`
`q1: 25,` `// Quartiles: Lower`
`q2: 35,` `// Median`
`q3: 50)` `// Upper`
- `size (array)` : Size of chart. If the second entry is `auto`, it automatically scales to accommodate the number of entries plotted

size: Default: `(1, auto)`

label-key: `integer` or `string` Default: `0`

Index in the array where labels of each entry is stored

mark: `string` Default: `"*"`

Mark to use for plotting outliers. Set `none` to disable. Defaults to `"x"`

..plot-args: `any`

Additional arguments are passed to `plot.plot`

4.0.3 columnchart

Draw a column chart. A column chart is a chart that represents data with rectangular bars that grow from bottom to top, proportional to the values they represent.

Styling

Root: `columnchart`.

bar-width: `float` Default: `0.8`

Width of a single bar (basic) or a cluster of bars (clustered) in the plot.

x-inset: `float` Default: `1`

Distance of the plot data to the plot's edges on the x-axis of the plot.

You can use any plot or axes related style keys, too.

The `columnchart` function is a wrapper of the `plot` API. Arguments passed to `..plot-args` are passed to the `plot.plot` function.

Parameters

```
columnchart(
  data: array,
  label-key: int string,
  value-key: int string,
  error-key: none int string array,
  mode: string,
  size: array,
  bar-style: style function,
  x-label: content none,
  y-format,
  y-label: content none,
  labels: none content,
  ..plot-args: any
)
```

data: array

Array of data rows. A row can be of type array or dictionary, with `label`-key and `value`-key being the keys to access a rows label and value(s).

Example

```
(([A], 1), ([B], 2), ([C], 3),)
```

label-key: int or string Default: 0

Key to access the label of a data row. This key is used as argument to the rows `.at(...)` function.

value-key: int or string Default: 1

Key(s) to access value(s) of data row. These keys are used as argument to the rows `.at(...)` function.

error-key: none or int or string or array Default: none

Key(s) to access error values of a data row. These keys are used as argument to the rows `.at(...)` function.

mode: string Default: "basic"

Chart mode:

basic Single bar per data row

clustered Group of bars per data row

stacked Stacked bars per data row

stacked100 Stacked bars per data row relative to the sum of the row

size: array Default: (auto, 1)

Chart size as width and height tuple in canvas unist; width can be set to auto.

bar-style: style or function Default: palette.red

Style or function (`idx => style`) to use for each bar, accepts a palette function.

x-label: content or none Default: none

x axis label

y-format: Default: auto**y-label:** content or none Default: none

Y axis label

labels: none or content Default: none

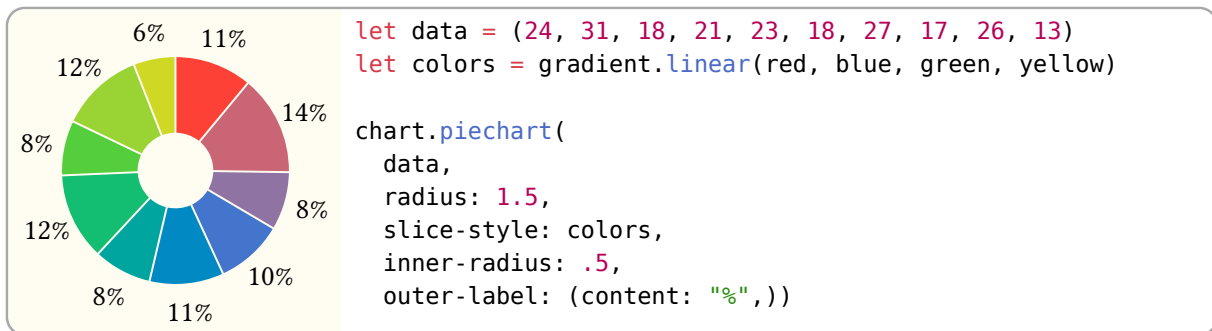
Legend labels per y value group

..plot-args: any

Arguments to pass to `plot.plot`

4.0.4 piechart

Draw a pie- or donut-chart



Styling

Root piechart

radius: number

Default: 1

Outer radius of the chart.

inner-radius: number

Default: 0

Inner radius of the chart slices. If greater than zero, the chart becomes a “donut-chart”.

gap: number or angle

Default: 0.5deg

Gap between chart slices to leave empty. This does not increase the charts radius by pushing slices outwards, but instead shrinks the slice. Big values can result in slices becoming invisible if no space is left.

outset-offset: number or ratio

Default: 10%

Absolute, or radius relative distance to push slices marked for “outsetting” outwards from the center of the chart.

outset-offset: string

Default: "OFFSET"

The mode of how to perform “outsetting” of slices:

- “OFFSET”: Offset slice position by outset-offset, increasing their gap to their siblings
- “RADIUS”: Offset slice radius by outset-offset, which scales the slice and leaves the gap unchanged

start: angle

Default: 90deg

The pie-charts start angle (ccw). You can use this to draw charts not forming a full circle.

stop: angle

Default: 450deg

The pie-charts stop angle (ccw).

clockwise: bool

Default: true

The pie-charts rotation direction.

outer-label.content: none or string or function

Default: "LABEL"

Content to display outside the charts slices. There are the following predefined values:

LABEL Display the slices label (see label-key)

% Display the percentage of the items value in relation to the sum of all values, rounded to the next integer

VALUE Display the slices value

If passed a <function> of the format (value, label) => content, that function gets called with each slices value and label and must return content, that gets displayed.

outer-label.radius: `number` or `ratio` Default: `125%`

Absolute, or radius relative distance from the charts center to position outer labels at.

outer-label.angle: `angle` or `auto` Default: `0deg`

The angle of the outer label. If passed `auto`, the label gets rotated, so that the baseline is parallel to the slices secant.

outer-label.anchor: `string` Default: `"center"`

The anchor of the outer label to use for positioning.

inner-label.content: `none` or `string` or `function` Default: `none`

Content to display insides the charts slices. See `outer-label.content` for the possible values.

inner-label.radius: `number` or `ratio` Default: `150%`

Distance of the inner label to the charts center. If passed a `<ratio>`, that ratio is relative to the mid between the inner and outer radius (`inner-radius` and `radius`) of the chart

inner-label.angle: `angle` or `auto` Default: `0deg`

See `outer-label.angle`.

inner-label.anchor: `string` Default: `"center"`

See `outer-label.anchor`.

legend.label: `none` or `string` or `function` Default: `"LABEL"`

See `outer-label.content`. The legend gets shown if this key is set `!= none`.

Anchors

The chart places one anchor per item at the radius of it's slice that gets named `"item-<index>"` (outer radius) and `"item-<index>-inner"` (inner radius), where `index` is the index of the slice data in `data`.

Parameters

```
piechart(
  data: array,
  value-key: none int string,
  label-key: none int string,
  outset-key: none int string,
  outset: none int array,
  slice-style: function array gradient,
  name,
  ..style
)
```

data: `array`

Array of data items. A data item can be:

- A number: A number that is used as the fraction of the slice
- An array: An array which is read depending on `value-key`, `label-key` and `outset-key`
- A dictionary: A dictionary which is read depending on `value-key`, `label-key` and `outset-key`

value-key: `none` or `int` or `string` Default: `none`

Key of the "value" of a data item. If for example data items are passed as dictionaries, the `value-key` is the key of the dictionary to access the items chart value.

label-key: `none` or `int` or `string` Default: `none`

Same as the `value-key` but for getting an items label content.

outset-key: `none` or `int` or `string` Default: `none`

Same as the value-key but for getting if an item should get outset (highlighted). The outset can be a bool, float or ratio. If of type bool, the outset distance from the style gets used.

outset: `none` or `int` or `array` Default: `none`

A single or multiple indices of items that should get offset from the center to the outsides of the chart. Only used if outset-key is none!

slice-style: `function` or `array` or `gradient` Default: `palette.red`

Slice style of the following types:

- function: A function of the form `index => style` that must return a style dictionary. This can be a `palette` function.
- array: An array of style dictionaries or fill colors of at least one item. For each slice the style at the slices index modulo the arrays length gets used.
- gradient: A gradient that gets sampled for each data item using the the slices index divided by the number of slices as position on the gradient.

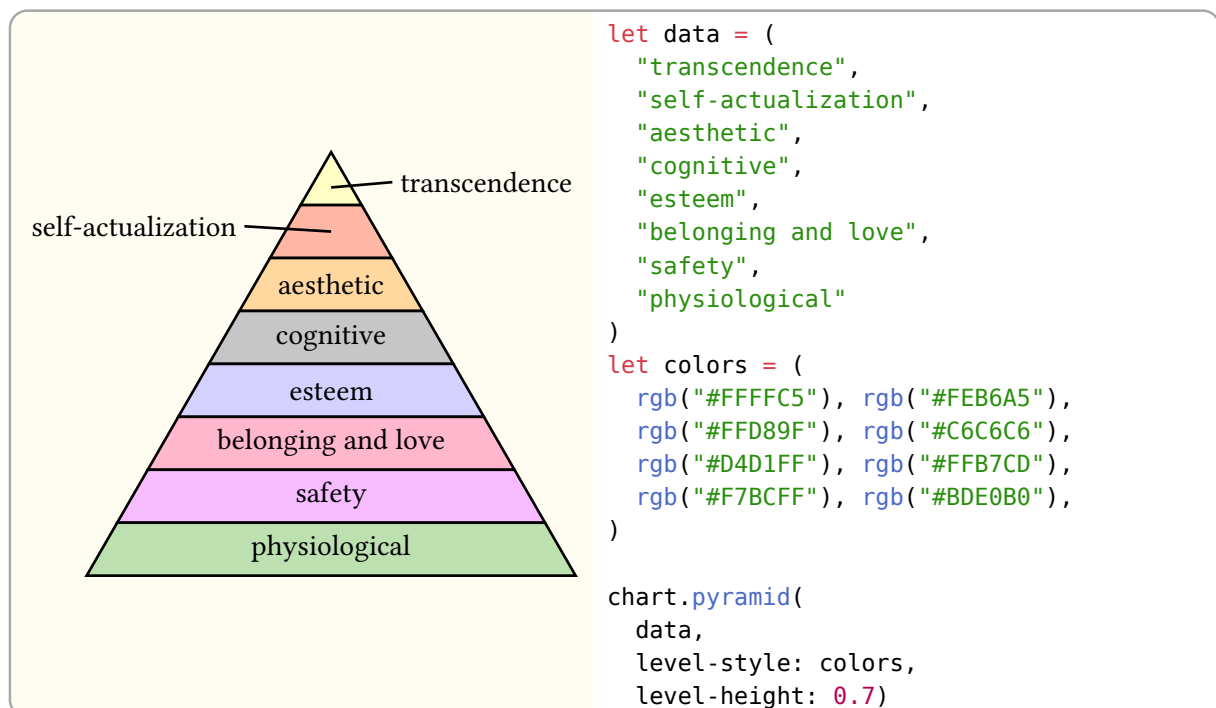
If one of stroke or fill is not in the style dictionary, it is taken from the charts style.

name: Default: `none`

..style:

4.0.5 pyramid

Draw a pyramid chart



Styling

Root pyramid

level-height: `number` Default: `1`

Minimum level height.

gap: `number` or `ratio` Default: `0`

Gap between levels to leave empty. If mode is “AREA-HEIGHT”, the value must be a ratio and will be proportional to the height of the first level.

mode: `string` Default: `"REGULAR"`

The mode of how to shape each level:

- “REGULAR”: All levels have the same height and make a perfectly triangular pyramid
- “AREA-HEIGHT”: The area of each level is proportional to its value. Only the height is adapted, keeping the pyramid triangular
- “HEIGHT”: The height of each level is proportional to its value. The pyramid is kept as a perfect triangle
- “WIDTH”: The height of each level is fixed, but its width is proportional to the value. The pyramid might not be perfectly triangular

side-label.content: `none` or `string` or `function` Default: `none`

Content to display outside the charts levels, on the side. There are the following predefined values:

LABEL Display the levels label (see `label-key`)

% Display the percentage of the items value in relation to the sum of all values, rounded to the next integer

VALUE Display the levels value

If passed a `<function>` of the format `(value, label) => content`, that function gets called with each levels value and label and must return content, that gets displayed.

side-label.side: `string` Default: `"west"`

The side of the chart on which to place side labels, either “west” or “east”

inner-label.content: `none` or `string` or `function` Default: `"LABEL"`

Content to display inside the charts levels. See `side-label.content` for the possible values.

inner-label.force-inside: `boolean` Default: `false`

If false, labels are automatically placed outside their corresponding levels if they don’t fit inside. If true, they are always placed inside.

Anchors

The chart places one anchor per item at the center of its level that gets named `"levels.<index>"`, one on the middle of its left side named `"levels.<index>.west"`, and one on the right side named `"levels.<index>.east"`, where `index` is the index of the level data in `data`.

Parameters

```
pyramid(
  data: array,
  value-key: none | int | string,
  label-key: none | int | string,
  level-style: function | array | gradient,
  name,
  ..style
)
```

data: `array`

Array of data items. A data item can be:

- A number: A number that is used as the fraction of the level
- An array: An array which is read depending on value-key and label-key
- A dictionary: A dictionary which is read depending on value-key and label-key

value-key: `none` or `int` or `string` Default: `none`

Key of the “value” of a data item. If for example data items are passed as dictionaries, the value-key is the key of the dictionary to access the items chart value.

label-key: `none` or `int` or `string` Default: `none`

Same as the value-key but for getting an items label content.

level-style: `function` or `array` or `gradient` Default: `palette.red`

Level style of the following types:

- function: A function of the form `index => style` that must return a style dictionary. This can be a `palette` function.
- array: An array of style dictionaries or fill colors of at least one item. For each level the style at the levels index modulo the arrays length gets used.
- gradient: A gradient that gets sampled for each data item using the the levels index divided by the number of levels as position on the gradient.

If one of stroke or fill is not in the style dictionary, it is taken from the charts style.

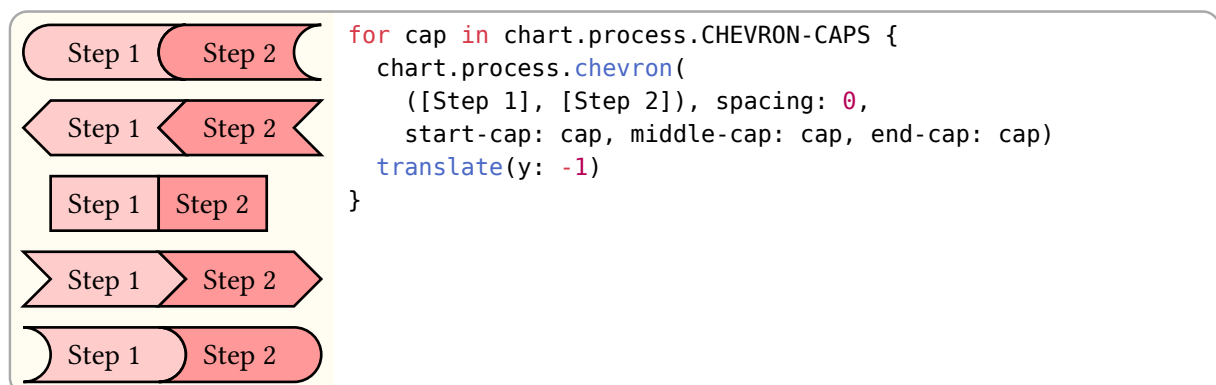
name: Default: `none`

..style:

4.1 Processes

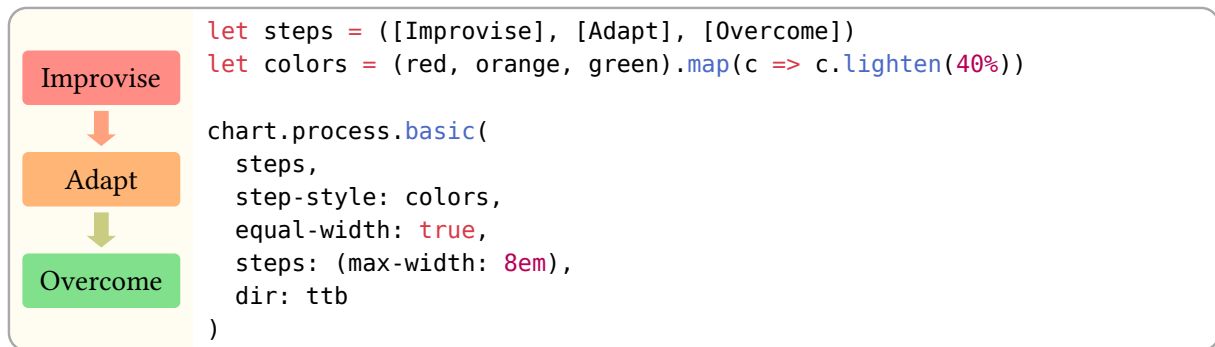
4.1.1 CHEVRON-CAPS

Possible chevron caps: ("`(`", "`<`", "`|`", "`>`", "`)`")



4.1.2 basic

Draw a basic process chart, describing sequential steps



Styling

Root process-basic

spacing: number or length Default: 0.2em

Gap between steps and arrows.

steps.radius: number or length Default: 0.2em

Corner radius of the steps boxes.

steps.padding: number or length Default: 0.6em

Inner padding of the steps boxes.

steps.max-width: number or length Default: 5em

Maximum width of the steps boxes.

arrows.width: number or length Default: 1.2em

Width / length of arrows.

arrows.height: number or length Default: 1em

Height of arrows.

Parameters

```

basic(
  steps: array,
  arrow-style: auto function array gradient,
  step-style: function array gradient,
  equal-width: boolean,
  equal-height: boolean,
  dir: direction,
  name,
  ..style
)

```

steps: array

Array of steps (<content> or <str>)

arrow-style: auto or function or array or gradient Default: auto

Arrow style of the following types:

- auto: If set to auto, the arrows will be filled with a color in between those of the neighboring steps
- function: A function of the form `index => style` that must return a style dictionary. This can be a palette function.
- array: An array of style dictionaries or fill colors of at least one item. For each arrow the style at the arrows index modulo the arrays length gets used.

- **gradient**: A gradient that gets sampled for each data item using the arrows index divided by the number of steps as position on the gradient.

If one of stroke or fill is not in the style dictionary, it is taken from the charts style.

step-style: `function` or `array` or `gradient`

Default: `palette.red`

Step style of the following types:

- **function**: A function of the form `index => style` that must return a style dictionary. This can be a `palette` function.
- **array**: An array of style dictionaries or fill colors of at least one item. For each step the style at the steps index modulo the arrays length gets used.
- **gradient**: A gradient that gets sampled for each data item using the steps index divided by the number of steps as position on the gradient.

If one of stroke or fill is not in the style dictionary, it is taken from the charts style.

equal-width: `boolean`

Default: `false`

If true, all steps will be sized to have the same width

equal-height: `boolean`

Default: `false`

If true, all steps will be sized to have the same height

dir: `direction`

Default: `ltr`

Direction in which the steps are laid out. The first step is always placed at (0, 0)


name:

Default: `none`

..style:

4.1.3 chevron

Draw a chevron process chart, describing sequential steps



```
let steps = ([Improvise], [Adapt], [Overcome])
let colors = (red, orange, green).map(c => c.lighten(40%))

chart.process.chevron(
  steps,
  step-style: colors,
  equal-width: true,
  steps: (max-width: 8em, cap-ratio: 25%),
  dir: btt
)
```

Styling

Root process-chevron

spacing: `number` or `length`

Default: `0.2em`

Gap between steps.

start-cap: `string`

Default: `">"`

Cap at the start of the process (first step). See [CHEVRON-CAPS](#) for possible values.

mid-cap: `string` Default: `">"`

Cap between steps. See [CHEVRON-CAPS](#) for possible values.

end-cap: `string` Default: `">"`

Cap at the end of the process (last step). See [CHEVRON-CAPS](#) for possible values.

start-in-cap: `boolean` Default: `false`

If true, the content of the first step is shifted inside the start cap (useful with “(“ or “<”).

end-in-cap: `boolean` Default: `false`

If true, the content of the last step is shifted inside the end cap (useful with “)” or “>”).

steps.padding: `number` or `length` Default: `0.6em`

Inner padding of the steps boxes.

steps.max-width: `number` or `length` Default: `5em`

Maximum width of the steps boxes.

steps.cap-ratio: `ratio` Default: `50%`

Ratio of the caps width relative to the steps heights (or the opposite if laid out vertically).

Parameters

```
chevron(
  steps: array,
  step-style: function array gradient,
  equal-length: boolean,
  dir: direction,
  name,
  ..style
)
```

steps: `array`

Array of steps (<content> or <str>)

step-style: `function` or `array` or `gradient` Default: `palette.red`

Step style of the following types:

- `function`: A function of the form `index => style` that must return a style dictionary. This can be a `palette` function.
- `array`: An array of style dictionaries or fill colors of at least one item. For each step the style at the steps index modulo the arrays length gets used.
- `gradient`: A gradient that gets sampled for each data item using the steps index divided by the number of steps as position on the gradient.

If one of `stroke` or `fill` is not in the style dictionary, it is taken from the charts style.

equal-length: `boolean` Default: `false`

If true, all steps will be sized to have the same length (in the layout’s direction, the other dimensions always being equal)

dir: `direction` Default: `ltr`

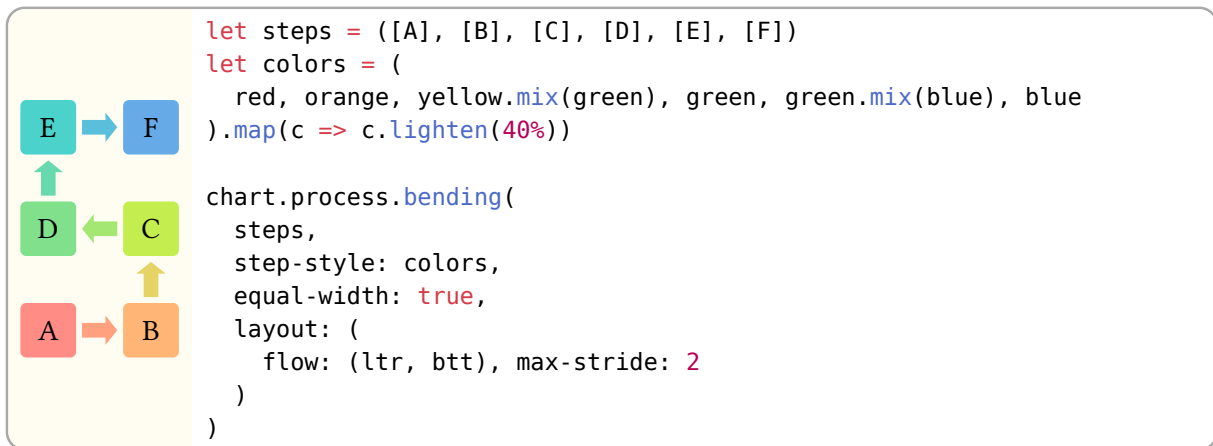
Direction in which the steps are laid out. The first step is always placed at (0, 0)

name: Default: `none`

..style:

4.1.4 bending

Draw a bending process chart, describing sequential steps in a zigzag layout



Styling

Root process-bending

spacing: number or length Default: 0.2em

Gap between steps and arrows.

steps.radius: number or length Default: 0.2em

Corner radius of the steps boxes.

steps.padding: number or length Default: 0.6em

Inner padding of the steps boxes.

steps.max-width: number or length Default: 5em

Maximum width of the steps boxes.

arrows.width: number or length Default: 1.2em

Width / length of arrows.

arrows.height: number or length Default: 1em

Height of arrows.

layout.max-stride: number Default: 3

Maximum number of steps before turning, i.e. making a zigzag.

layout.flow: array Default: (ltr, ttb)

Pair of directions on different axes indicating the primary and secondary layout directions.

Parameters

```

bending(
  steps: array,
  arrow-style: auto function array gradient,
  step-style: function array gradient,
  equal-width: boolean,
  equal-height: boolean,
  name,
  ..style
)

```

steps: array

Array of steps (<content> or <str>)

arrow-style: auto or function or array or gradient

Default: auto

Arrow style of the following types:

- auto: If set to auto, the arrows will be filled with a color in between those of the neighboring steps
- function: A function of the form `index => style` that must return a style dictionary. This can be a palette function.
- array: An array of style dictionaries or fill colors of at least one item. For each arrow the style at the arrows index modulo the arrays length gets used.
- gradient: A gradient that gets sampled for each data item using the arrows index divided by the number of steps as position on the gradient.

If one of stroke or fill is not in the style dictionary, it is taken from the charts style.

step-style: function or array or gradient

Default: palette.red

Step style of the following types:

- function: A function of the form `index => style` that must return a style dictionary. This can be a palette function.
- array: An array of style dictionaries or fill colors of at least one item. For each step the style at the steps index modulo the arrays length gets used.
- gradient: A gradient that gets sampled for each data item using the steps index divided by the number of steps as position on the gradient.

If one of stroke or fill is not in the style dictionary, it is taken from the charts style.

equal-width: boolean

Default: false

If true, all steps will be sized to have the same width

equal-height: boolean

Default: false

If true, all steps will be sized to have the same height

name:

Default: none

..style: