

CeTZ

ein Typst
Zeichenpaket

Johannes Wolf
fenjalien

Version 0.2.0

1 Introduction	3
2 Usage	3
2.1 CeTZ Unique Argument Types	3
2.2 Anchors	3
2.2.1 Compass Anchors	3
3 Draw Function Reference	4
3.1 Canvas	4
3.2 Styling	4
3.3 Shapes	6
3.3.1 circle	6
3.3.2 circle-through	6
3.3.3 arc	7
3.3.4 mark	8
3.3.5 line	9
3.3.6 grid	10
3.3.7 content	11
3.3.8 rect	12
3.3.9 bezier	12
3.3.10 bezier-through	13
3.3.11 catmull	14
3.3.12 hobby	14
3.3.13 merge-path	15
3.4 Grouping	17
3.4.1 intersections	17
3.4.2 group	17
3.4.3 anchor	18
3.4.4 copy-anchors	19
3.4.5 place-anchors	19
3.4.6 set-ctx	19
3.4.7 get-ctx	20
3.4.8 for-each-anchor	20
3.4.9 on-layer	21
3.4.10 place-marks	21
3.5 Transformations	23
3.5.1 set-transform	23
3.5.2 rotate	23
3.5.3 translate	23
3.5.4 scale	24
3.5.5 set-origin	24
3.5.6 move-to	25
3.5.7 set-viewport	25

1 Introduction

This package provides a way to draw stuff using a similar API to [Processing](#) but with relative coordinates and anchors from [TikZ](#). You also won't have to worry about accidentally drawing over other content as the canvas will automatically resize. And remember: up is positive!

The name CeTZ is a recursive acronym for “CeTZ, ein Typst Zeichenpaket” (german for “CeTZ, a Typst drawing package”) and is pronounced like the word “Cats”.

2 Usage

This is the minimal starting point:

```
#import "@preview/cetz:0.2.0"
#cetz.canvas({
  import cetz.draw: *
  ...
})
```

Note that draw functions are imported inside the scope of the canvas block. This is recommended as draw functions override Typst's functions such as `line`.

2.1 CeTZ Unique Argument Types

Many CeTZ functions expect data in certain formats which we will call types. Note that these are actually made up of Typst primitives.

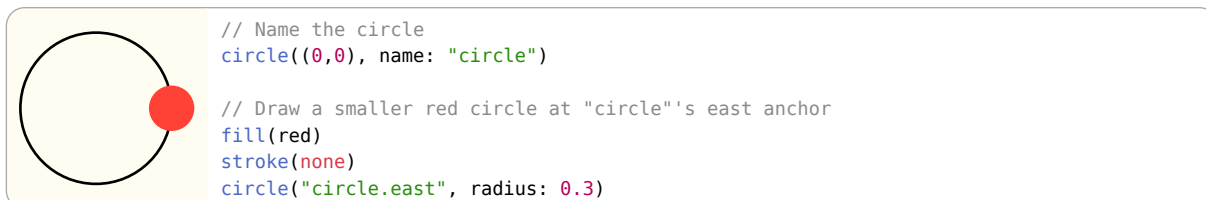
coordinate Any coordinate system. See coordinate-systems.

number Any of `float`, `integer` or `length`.

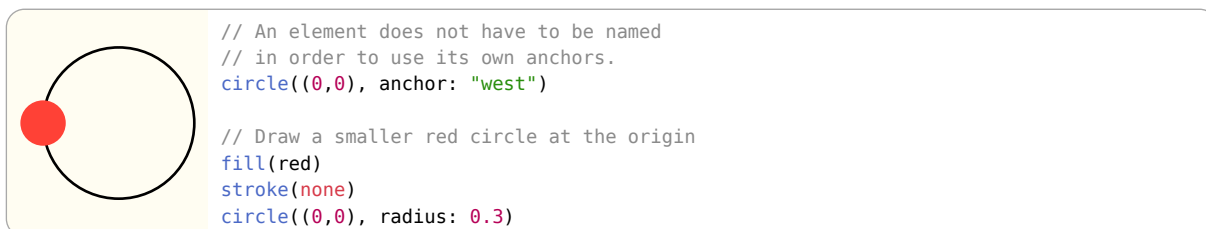
style Named arguments (or a dictionary if used for a single argument) of style key-values

2.2 Anchors

Anchors are named positions relative to named elements. To use an anchor of an element, you must give the element a name using the `name` argument. All elements with the `name` argument allow anchors.

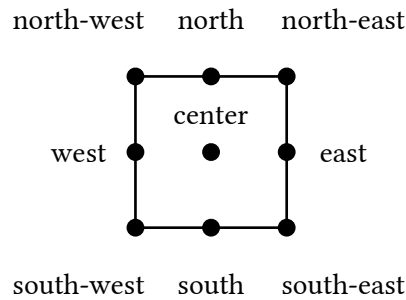


Elements can be placed relative to their own anchors if they have an argument called `anchor`:



2.2.1 Compass Anchors

Some elements support compass anchors. TODO



3 Draw Function Reference

3.1 Canvas

`canvas`(background: `none`, length: `1cm`, debug: `false`, body)

background `color` (default: `none`)

A color to be used for the background of the canvas.

length `length` (default: `1cm`)

Used to specify what 1 coordinate unit is.

debug `bool` (default: `false`)

Shows the bounding boxes of each element when ``true``.

body

A code block in which functions from `draw.typ` have been called.

3.2 Styling

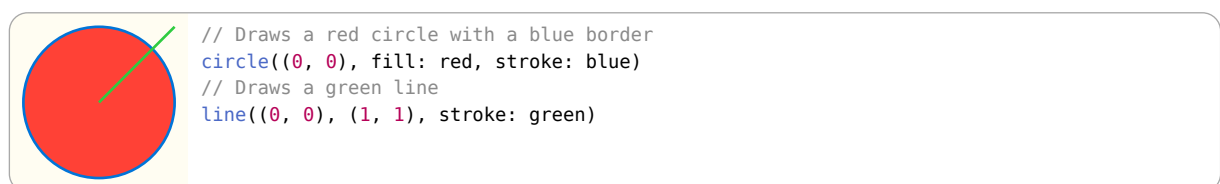
You can style draw elements by passing the relevant named arguments to their draw functions. All elements that draw something have stroke and fill styling unless said otherwise.

fill `color` or `none` Default: `none`

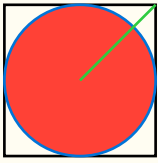
How to fill the drawn element.

stroke `none` or `auto` or `length` or `color` or `dictionary` or `stroke` Default: `1pt + luma(0%)`

How to stroke the border or the path of the draw element. See Typst's line documentation for more details: <https://typst.app/docs/reference/visualize/line/#parameters-stroke>



Instead of having to specify the same styling for each time you want to draw an element, you can use the `set-style` function to change the style for all elements after it. You can still pass styling to a draw function to override what has been set with `set-style`. You can also use the `fill()` and `stroke()` functions as a shorthand to set the fill and stroke respectively.



```
// Draws an empty square with a black border
rect((-1, -1), (1, 1))

// Sets the global style to have a fill of red and a stroke of blue
set-style(stroke: blue, fill: red)
circle((0,0))

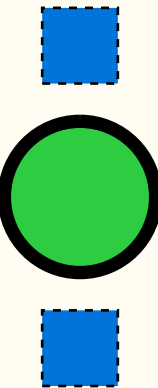
// Draws a green line despite the global stroke is blue
line((0, (1,1), stroke: green)
```

When using a dictionary for a style, it is important to note that they update each other instead of overriding the entire option like a non-dictionary value would do. For example, if the stroke is set to (paint: red, thickness: 5pt) and you pass (paint: blue), the stroke would become (paint: blue, thickness: 5pt).



```
// Sets the stroke to red with a thickness of 5pt
set-style(stroke: (paint: red, thickness: 5pt))
// Draws a line with the global stroke
line((0,0), (1,0))
// Draws a blue line with a thickness of 5pt because dictionaries update the style
line((0,0), (1,1), stroke: (paint: blue))
// Draws a yellow line with a thickness of 1pt because other values override the style
line((0,0), (0,1), stroke: yellow)
```

You can also specify styling for each type of element. Note that dictionary values will still update with its global value, the full hierarchy is function > element type > global. When the value of a style is auto, it will become exactly its parent style.



```
set-style(
  // Global fill and stroke
  fill: green,
  stroke: (thickness: 5pt),
  // Stroke and fill for only rectangles
  rect: (stroke: (dash: "dashed"), fill: blue),
)
rect((0,0), (1,1))
circle((0.5, -1.5))
rect((0,-3), (1, -4), stroke: (thickness: 1pt))
```

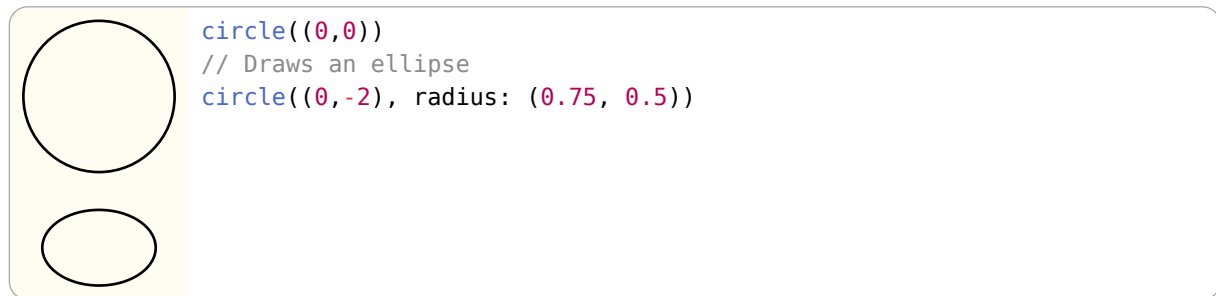


```
// Its a nice drawing okay
set-style(
  rect: (
    fill: red,
    stroke: none
  ),
  line: (
    fill: blue,
    stroke: (dash: "dashed")
  ),
)
rect((0,0), (1,1))
line((0, -1.5), (0.5, -0.5), (1, -1.5), close: true)
circle((0.5, -2.5), radius: 0.5, fill: green)
```

3.3 Shapes

3.3.1 circle

Draws a circle or ellipse.



Parameters

```
circle(
  position: coordinate,
  name: none string,
  anchor: none string,
  ..style: style
)
```

position coordinate

The position to place the circle on.

Style Root circle

Style Keys

radius number or array

Default: 1

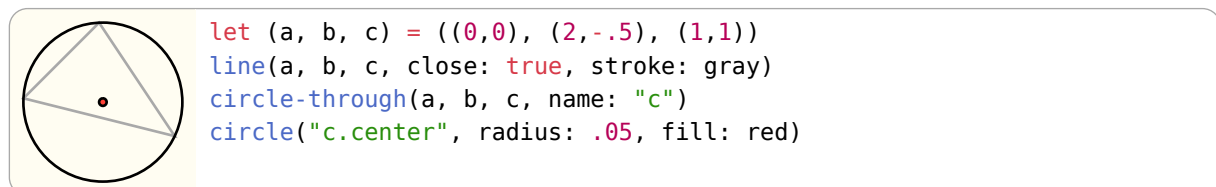
A number that defines the size of the circle's radius. Can also be set to a tuple of two numbers to define the radii of an ellipse, the first number is the x radius and the second is the y radius.

Anchors

Supports compass anchors. The "center" anchor is the default.

3.3.2 circle-through

Draws a circle through three coordinates



Parameters

```
circle-through(
  a: coordinate,
  b: coordinate,
  c: coordinate,
  name: none string,
  anchor: none string,
  ..style: style
)
```

- a** `coordinate`
Coordinate a
- b** `coordinate`
Coordinate b
- c** `coordinate`
Coordinate c

Style Root circle

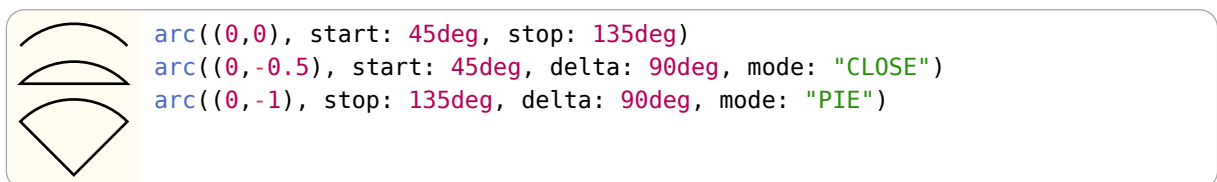
Anchors

Supports the same anchors as `circle` as well as:

- a** Coordinate a
- b** Coordinate b
- c** Coordinate c

3.3.3 arc

Draws a circular segment.



Note that two of the three angle arguments (start, stop and delta) must be set.

Parameters

```
arc(
  position: coordinate,
  start: auto angle,
  stop: auto angle,
  delta: auto angle,
  name: none string,
  anchor: none string,
  ..style: style
)
```

position `coordinate`

Position to place the arc at.

start `auto` or `angle`

Default: `"auto"`

The angle at which the arc should start. Remember that `0deg` points directly towards the right and `90deg` points up.

stop `auto` or `angle`

Default: `"auto"`

The angle at which the arc should stop.

delta `auto` or `angle`

Default: `"auto"`

The change in angle away start or stop.

Style Root arc

Style Keys

radius `number` or `array`Default: `1`

The radius of the arc. An elliptical arc can be created by passing a tuple of numbers where the first element is the x radius and the second element is the y radius.

mode `string`Default: `"OPEN"`

The options are: "OPEN" no additional lines are drawn so just the arc is shown; "CLOSE" a line is drawn from the start to the end of the arc creating a circular segment; "PIE" lines are drawn from the start and end of the arc to the origin creating a circular sector.

Anchors

Supports compass anchors when mode is "PIE"

center The center of the arc, this is the default anchor.

arc-center The midpoint of the arc's curve.

chord-center Center of chord of the arc drawn between the start and end point.

origin The origin of the arc's circle.

arc-start The position at which the arc's curve starts.

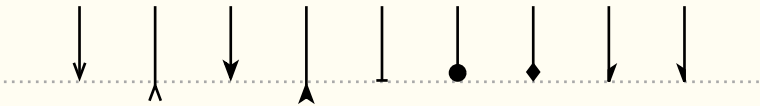
arc-end The position of the arc's curve end.

3.3.4 mark

Draws a single mark pointing at a target coordinate

```
➤ mark((0,0), (1,0), symbol: ">", fill: black)
➤ mark((0,0), (1,1), symbol: ">", scale: 3, fill: black)
```

Or as part of a path based element that supports the mark style key:



```
rotate(-90deg)
set-style(mark: (fill: black))
line((1, -1), (1, 9), stroke: (paint: gray, dash: "dotted"))
line((0, 8), (rel: (1, 0)), mark: (end: "left-harpoon"))
line((0, 7), (rel: (1, 0)), mark: (end: "right-harpoon"))
line((0, 6), (rel: (1, 0)), mark: (end: "<>"))
line((0, 5), (rel: (1, 0)), mark: (end: "o"))
line((0, 4), (rel: (1, 0)), mark: (end: "|"))
line((0, 3), (rel: (1, 0)), mark: (end: "<"))
line((0, 2), (rel: (1, 0)), mark: (end: ">"))
set-style(mark: (fill: none))
line((0, 1), (rel: (1, 0)), mark: (end: "<"))
line((0, 0), (rel: (1, 0)), mark: (end: ">"))
```

Parameters

```
mark(
  from: coordinate,
  to: coordinate,
  ..style: style
)
```


from `coordinate`

The position to place the mark.

to `coordinate`

The position the mark should point towards.

Style Root `mark`

Style Keys

symbol `string`

Default: `">"`

The type of mark to draw when using the mark function.

start `string` or `none` or `array`

Default: `none`

The type of mark to draw at the start of a path.

end `string` or `none` or `array`

Default: `none`

The type of mark to draw at the end of a path.

length `number`

Default: `0.2`

The length of the mark along its direction it is pointing.

width `number`

Default: `0.15`

The width of the mark along the normal of its direction.

inset `number`

Default: `0.05`

The distance by which something inside the arrow tip is set inwards.

scale `float`

Default: `1`

A factor that is applied to the mark's length, width and inset.

sep `number`

Default: `1`

The distance between multiple marks along their path.

flex `boolean`

Default: `true`

Only applicable when marks are used on curves such as bezier and hobby. If true, the mark will point along the secant of the curve. If false, the tangent at the marks tip is used.

position-samples `integer`

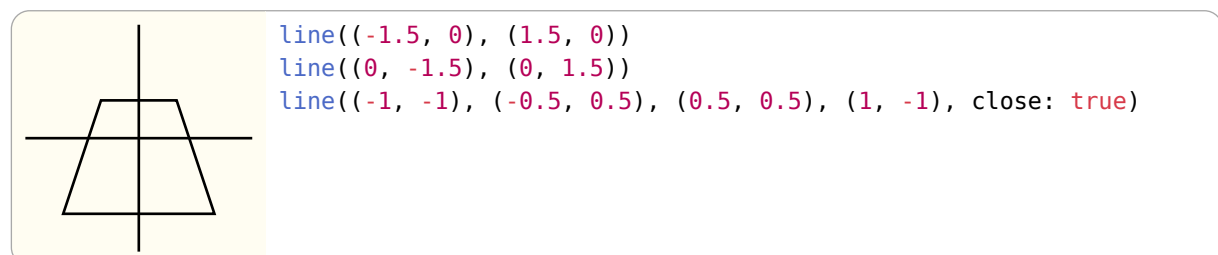
Default: `30`

Only applicable when marks are used on curves such as bezier and hobby. The maximum number of samples to use for calculating curve positions. A higher number gives better results but may slow down compilation.

Note: The size of the mark depends on its style values, not the distance between `from` and `to`, which only determine its orientation.

3.3.5 line

Draws a line, more than two points can be given to create a line-strip.



Parameters

```
line(
  ..pts-style: coordinates style,
  close: bool,
  name: none string
)
```

..pts-style coordinates or style

Positional two or more coordinates to draw lines between. Accepts style key-value pairs.

close bool

Default: "false"

If true, the line-strip gets closed to form a polygon

Style Root line

Style Keys

Supports marks

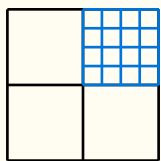
Anchors

start The line's start position

end The line's end position

3.3.6 grid

Draw a grid between two coordinates



```
// Draw a grid
grid((0,0), (2,2))

// Draw a smaller blue grid
grid((1,1), (2,2), stroke: blue, step: .25)
```

Style Root grid

Anchors

Supports compass anchors.

Parameters

```
grid(
  from: coordinate,
  to: coordinate,
  step: number,
  name: none string,
  help-lines,
  ..style: style
)
```

from coordinate

The top left of the grid

to coordinate

The bottom right of the grid

step number

Default: "1"

Grid spacing.

help-lines

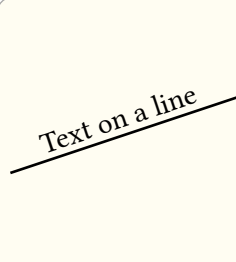
Default: "false"

3.3.7 content

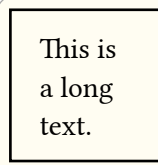
Positions Typst content in the canvas. Note that the content itself is not transformed only its position is.

```
Hello World! content((0,0), [Hello World!])
```

To put text on a line you can let the function calculate the angle between its position and a second coordinate by passing it to angle:



```
line((0, 0), (3, 1), name: "line")
content(
  ("line.start", 0.5, "line.end"),
  angle: "line.end",
  padding: .1,
  anchor: "south",
  [Text on a line]
)
```



```
// Place content in a rect between two coordinates
content((0, 0), (2, 2), box(par(justify: false)[This is a long text.],
stroke: 1pt, width: 100%, height: 100%, inset: 1em))
```

Parameters

```
content(
  ..args-style: coordinate content style,
  angle: angle coordinate,
  anchor: none string,
  name: none string
)
```

..args-style coordinate or content or style

When one coordinate is given as a positional argument, the content will be placed at that position. When two coordinates are given as positional arguments, the content will be placed inside a rectangle between the two positions. All named arguments are styling and any additional positional arguments will panic.

angle angle or coordinate

Default: "0deg"

Rotates the content by the given angle. A coordinate can be given to rotate the content by the angle between it and the first coordinate given in args. This effectively points the right hand side of the content towards the coordinate. This currently exists because Typst's rotate function does not change the width and height of content.

Style Root content

Style Keys

padding number or dictionary

Default: 0

Sets the spacing around content. Can be a single number to set padding on all sides or a dictionary to specify each side specifically. The dictionary follows Typst's pad function: <https://typst.app/docs/reference/layout/pad/>

frame string or none

Default: none

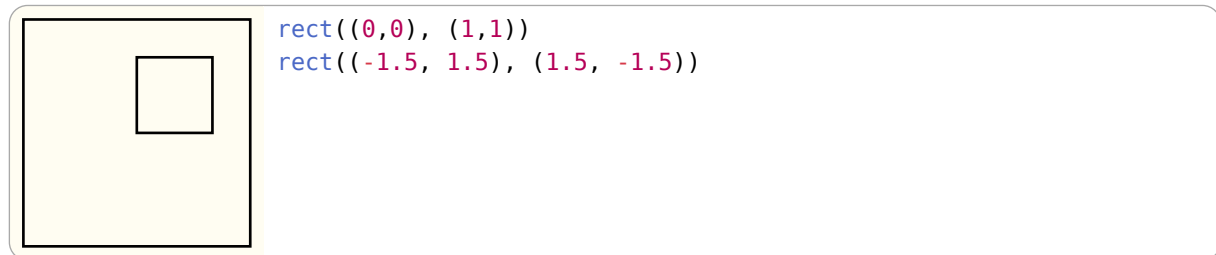
Sets the frame style. Can be none, "rect" or "circle" and inherits the stroke and fill style.

Anchors

Supports compass anchors.

3.3.8 rect

Draws a rectangle between two coordinates.



Style Root rect

Anchors

Supports compass anchors.

Parameters

```
rect(
  a: coordinate,
  b: coordinate,
  name: none string,
  anchor: none string,
  ..style: style
)
```

a coordinate

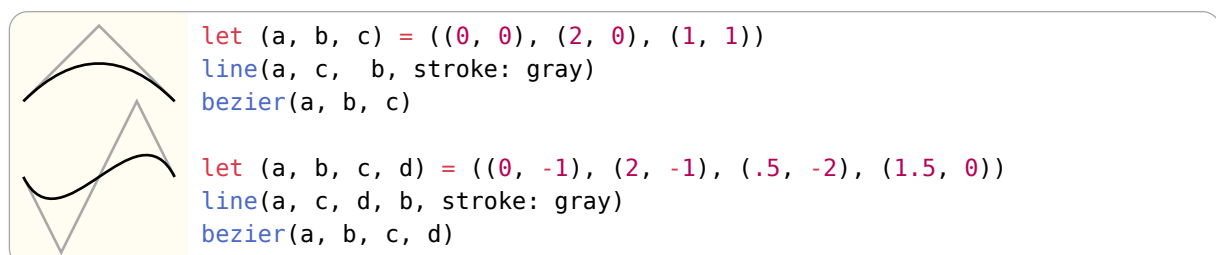
Coordinate of the top left corner of the rectangle.

b coordinate

Coordinate of the bottom right corner of the rectangle. You can draw a rectangle with a specified width and height by using relative coordinates for this parameter (rel: (width, height)).

3.3.9 bezier

Draws a quadratic or cubic bezier curve



Parameters

```
bezier(
  start: coordinate,
  end: coordinate,
  ..ctrl-style: coordinate style,
  name: none string
)
```

start coordinate

Start position

end coordinate

End position (last coordinate)

..ctrl-style coordinate or style

The first two positional arguments are taken as cubic bezier control points, where the first is the start control point and the second is the end control point. One control point can be given for a quadratic bezier curve instead. Named arguments are for styling.

Style Root bezier

Style Keys

Supports marks.

Anchors

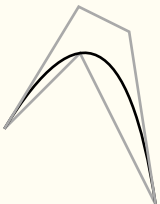
ctrl-n nth control point where n is an integer starting at 0

start The start position of the curve.

end The end position of the curve.

3.3.10 bezier-through

Draw a cubic bezier curve through a set of three points. See bezier for style and anchor details.



```
let (a, b, c) = ((0, 0), (1, 1), (2, -1))
line(a, b, c, stroke: gray)
bezier-through(a, b, c, name: "b")

// Show calculated control points
line(a, "b.ctrl-0", "b.ctrl-1", c, stroke: gray)
```

Parameters

```
bezier-through(
  start: coordinate,
  pass-through: coordinate,
  end: coordinate,
  name: none string,
  ..style: style
)
```

start coordinate

Start position

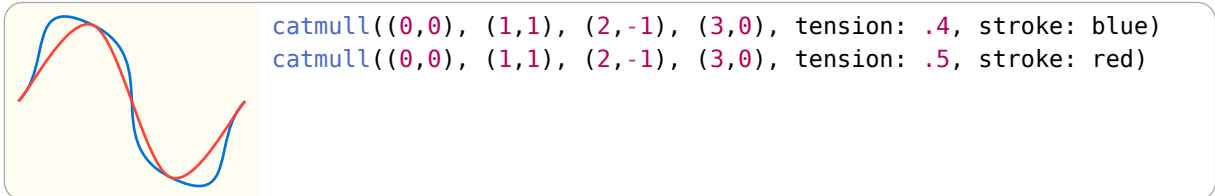
pass-through coordinate

Curve mid-point

end `coordinate`
End coordinate

3.3.11 catmull

Draw a Catmull-Rom curve through a set of points.



Parameters

```
catmull(
  ..pts-style: coordinate style,
  close: bool,
  name: none string
)
```

..pts-style `coordinate` or `style`

Positional arguments should be coordinates that the curve should pass through. Named arguments are for styling.

close `bool`

Default: `"false"`

Closes the curve with a straight line between the start and end of the curve.

Style Root `catmull`

Style Keys

tension `float`

Default: `0.5`

I need a description

Supports marks.

Anchors

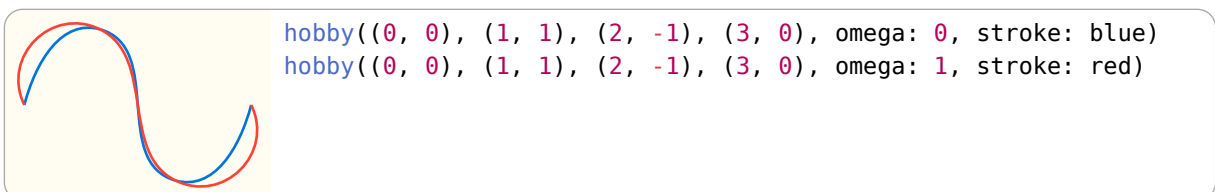
start The position of the start of the curve.

end The position of the end of the curve.

pt-n The nth given position (0 indexed so "pt-0" is equal to "start")

3.3.12 hobby

Draws a Hobby curve through a set of points.



Parameters

```
hobby(
  ..pts-style: coordinate style,
  ta: auto array,
  tb: auto array,
  close: bool,
  name: none string
)
```

..pts-style coordinate or style

Positional arguments are the coordinates to use to draw the curve with, a minimum of two is required. Named arguments are for styling.

ta auto or array Default: "auto"

Outgoing tension at `pts.at(n)` from `pts.at(n)` to `pts.at(n+1)`. The number given must be one less than the number of points.

tb auto or array Default: "auto"

Incoming tension at `pts.at(n+1)` from `pts.at(n)` to `pts.at(n+1)`. The number given must be one less than the number of points.

close bool Default: "false"

Closes the curve with a straight line between the start and end of the curve.

Style Root hobby

Style Keys

Supports marks.

omega idk Default: none

The curve's curliness

rho idk Default: none

Anchors

start The position of the start of the curve.

end The position of the end of the curve.

pt-n The nth given position (0 indexed, so "pt-0" is equal to "start")

3.3.13 merge-path

Merges two or more paths by concatenating their elements. Anchors and visual styling, such as `stroke` and `fill`, are not preserved. When an element's path does not start at the same position the previous element's path ended, a straight line is drawn between them so that the final path is continuous. You must then pay attention to the direction in which element paths are drawn.



```
merge-path(fill: white, {
  line((0, 0), (1, 0))
  bezier((0, 0), (0, 0), (1,1), (0,1))
})
```

Parameters

```
merge-path(  
  body: elements,  
  close: bool,  
  name: none string,  
  ..style: style  
)
```

body `elements`

Elements with paths to be merged together.

close `bool`

Close the path with a straight line from the start of the path to its end.

Default: `"false"`

Aliases

start The start of the merged path.

end The end of the merged path.

Parameters

```
group(
  body: elements function,
  name: none string,
  anchor: none string,
  ..style: style
)
```

body elements or function

Elements to group together. A least one is required. A function that accepts ctx and returns elements is also accepted.

Style Root group

Style Keys

padding none or number or array or dictionary

Default: none

How much padding to add around the group's bounding box. none applies no padding. A number applies padding to all sides equally. A dictionary applies padding following Typst's pad function: <https://typst.app/docs/reference/layout/pad/>. An array follows CSS like padding: (y, x), (top, x, bottom) or (top, right, bottom, left).

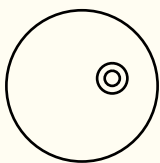
anchors Supports compass anchors. These are created based on the axis aligned bounding box of all the child elements of the group.

You can add custom anchors to the group by using the anchor element while in the scope of said group, see anchor for more details. You can also copy over anchors from named child element by using the copy-anchors element as they are not accessible from outside the group.

The default anchor is "center" but this can be overridden by using anchor to place a new anchor called "default".

3.4.3 anchor

Creates a new anchor for the current group. This element can only be used inside a group otherwise it will panic. The new anchor will be accessible from inside the group by using just the anchor's name as a coordinate.



```
// Create group
group(name: "g", {
  circle((0,0))
  anchor("x", (.4, .1))
  circle("x", radius: .2)
})
circle("g.x", radius: .1)
```

Parameters

```
anchor(
  name: string,
  position: coordinate
)
```

name string

The name of the anchor

position `coordinate`

The position of the anchor

3.4.4 copy-anchors

Copies multiple anchors from one element into the current group. Panics when used outside of a group. Copied anchors will be accessible in the same way anchors created by the anchor element are.

Parameters

```
copy-anchors(
  element: string,
  filter: auto array
)
```

element `string`

The name of the element to copy anchors from.

filter `auto` or `array`

Default: `"auto"`

When set to `auto` all anchors will be copied to the group. An array of anchor names can instead be given so only the anchors that are in the element and the list will be copied over.

3.4.5 place-anchors

TODO: Not writing the docs for this as it should be removed in place of better anchors before 0.2 Place multiple anchors along a path

Parameters

```
place-anchors(
  path: drawable,
  ..anchors: array,
  name
)
```

path `drawable`

Single drawable

..anchors `array`

List of anchor dictionaries of the form `(pos: <float>, name: <string>)`, where `pos` is a relative position on the path from 0 to 1.

- `name: (auto,string)`: If `auto`, take the name of the passed drawable. Otherwise sets the elements name

name

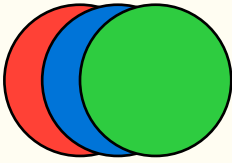
Default: `"auto"`

3.4.6 set-ctx

An advanced element that allows you to modify the current canvas context.

A context object holds the canvas' state, such as the element dictionary, the current transformation matrix, group and canvas unit length. The following fields are considered stable:

- `length (length)`: Length of one canvas unit as typst length
- `transform (cetz.matrix)`: Current 4x4 transformation matrix
- `debug (bool)`: True if the canvas' debug flag is set



```
// Setting a custom transformation matrix
set-ctx(ctx => {
  let mat = ((1, 0, .5, 0),
             (0, 1, 0, 0),
             (0, 0, 1, 0),
             (0, 0, 0, 1))
  ctx.transform = mat
  return ctx
})
circle((z: 0), fill: red)
circle((z: 1), fill: blue)
circle((z: 2), fill: green)
```

Parameters

`set-ctx`(callback: function)

callback function

A function that accepts the context dictionary and only returns a new one.

3.4.7 get-ctx

An advanced element that allows you to read the current canvas context through a callback and return elements based on it.

```
(
  (1, 0, 0.5, 0),
  (0, -1, -0.5, 0),
  (0, 0, 1, 0),
  (0, 0, 0, 1),
)

// Print the transformation matrix
get-ctx(ctx => {
  content(), [#repr(ctx.transform)]
})
```

Parameters

`get-ctx`(callback: function)

callback function

A function that accepts the context dictionary and can return elements.

3.4.8 for-each-anchor

Iterates through all anchors of an element and calls a callback for each one.

```
// Label nodes anchors
rect((0, 0), (2,2), name: "my-rect")
for-each-anchor("my-rect", (name) => {
  content(), box(inset: 1pt, fill: white, text(8pt, [#name])),
  angle: -30deg
})
```

Parameters

```
for-each-anchor(
  name: string,
  callback: function
)
```

name `string`

The name of the element with the anchors to loop through.

callback `function`


A function that takes the anchor name and can return elements.

3.4.9 on-layer

Places elements on a specific layer.

A layer determines the position of an element in the draw queue. A lower layer is drawn before a higher layer.

Layers can be used to draw behind or in front of other elements, even if the other elements were created before or after. An example would be drawing a background behind a text, but using the text's calculated bounding box for positioning the background.



```

// Draw something behind text
set-style(stroke: none)
content((0, 0), [This is an example.], name: "text")
on-layer(-1, {
  circle("text.north-east", radius: .3, fill: red)
  circle("text.south", radius: .4, fill: green)
  circle("text.north-west", radius: .2, fill: blue)
})
```

Parameters

```
on-layer(
  layer: float integer,
  body: elements
)
```

layer `float` or `integer`

The layer to place the elements on. Elements placed without on-layer are always placed on layer 0.

body `elements`

Elements to draw on the layer specified.

3.4.10 place-marks

TODO: Not writing the docs for this as it should be removed in place of better anchors before 0.2 Place one or more marks along a path

Mark items must get passed as positional arguments. A mark-item is an dictionary of the format: (mark: "<symbol>", pos: <float>), where the position pos is a relative position from 0 to 1 along the path.

Parameters

```
place-marks(  
  path: drawable,  
  ..marks-style: mark-item style,  
  name: none string  
)
```

path drawable

A single drawable

..marks-style mark-item or style

Positional mark-items and style key-value pairs

name none or string

Element name

Default: "none"

3.5 Transformations

All transformation functions push a transformation matrix onto the current transform stack. To apply transformations scoped use a `group(...)` object.

Transformation matrices get multiplied in the following order:

$$M_{\text{world}} = M_{\text{world}} \cdot M_{\text{local}}$$

3.5.1 set-transform

Sets the transformation matrix.

Parameters

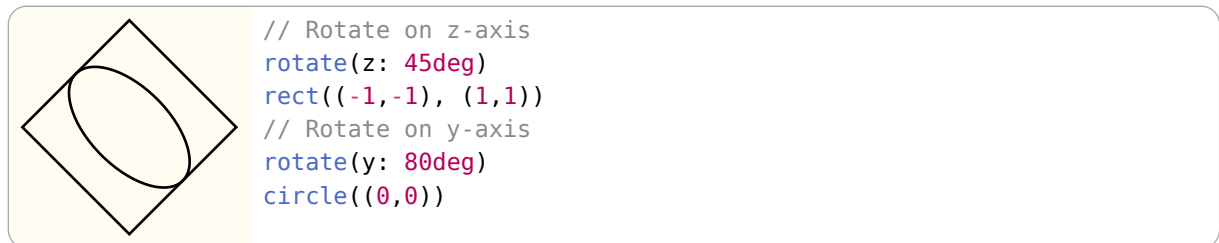
`set-transform(mat: none matrix)`

mat **none** or **matrix**

The 4x4 transformation matrix to set. If none is passed, the transformation matrix is set to the identity matrix (`matrix.ident()`).

3.5.2 rotate

Rotates the transformation matrix on the z-axis by a given angle or other axes when specified.



Parameters

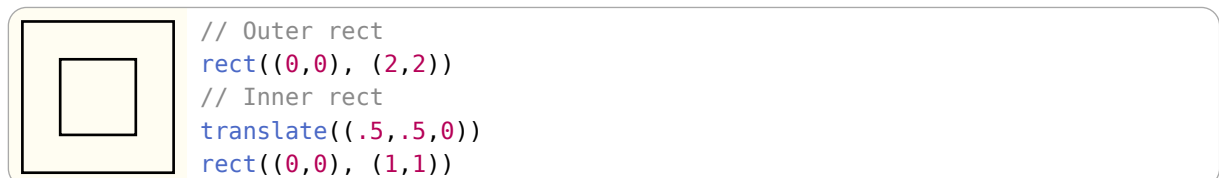
`rotate(..angles: angle)`

..angles **angle**

A single angle as a positional argument to rotate on the z-axis by. Named arguments of x, y or z can be given to rotate on their respective axis. You can give named arguments of yaw, pitch or roll to TODO

3.5.3 translate

Translates the transformation matrix by the given vector or dictionary.



Parameters

```
translate(
  vec: vector dictionary,
  pre: bool
)
```

vec vector or dictionary

The vector to translate by. A dictionary can be given instead with optional keys x, y and z to translate in the relevant axis.

pre bool

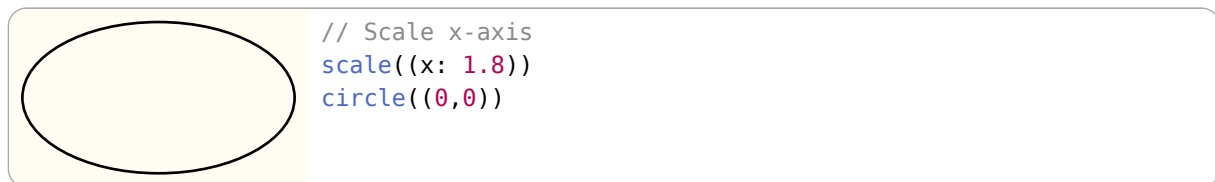
Default: "true"

Specify matrix multiplication order

- false: World = World * Translate
- true: World = Translate * World

3.5.4 scale

Scales the transformation matrix by the given factor(s).



Parameters

```
scale(factor: float dictionary)
```

factor float or dictionary

A float to scale the transformation matrix by. A dictionary with optional keys x, y and z can also be given to scale in the respective directions.

3.5.5 set-origin

Sets the given position as the origin



Parameters

```
set-origin(origin: coordinate)
```

origin coordinate

Coordinate to set as new origin (0,0,0)

3.5.6 move-to

Sets the previous coordinate.

The previous coordinate can be used via `()` (empty coordinate). It is also used as base for relative coordinates if not specified otherwise.

Parameters

```
move-to(pt: coordinate)
```

pt coordinate

The coordinate to move to.

3.5.7 set-viewport

Span viewport between two coordinates and set-up scaling and translation

Parameters

```
set-viewport(
  from: coordinate,
  to: coordinate,
  bounds: vector
)
```

from coordinate

Bottom-Left corner coordinate

to coordinate

Top right corner coordinate

bounds vector

Default: "(1, 1, 1)"

Viewport bounds vector that describes the inner width, height and depth of the viewport