

# The canvas package

[Johannes Wolf](#) and [fenjalien](#)  
<https://github.com/johannes-wolf/typst-canvas>

## Contents

1. Introduction .....	2
2. Usage .....	2
2.1. Argument Types .....	2
2.2. Anchors .....	2
3. Draw Function Reference .....	3
3.1. Canvas .....	3
3.2. Styling .....	3
3.3. Elements .....	4
3.3.1. Line .....	4
3.3.2. Rectangle .....	4
3.3.3. Arc .....	5
3.3.4. Circle .....	5
3.3.5. Bezier .....	5
3.3.6. Content .....	6
3.3.7. Grid .....	6
3.3.8. Arrow Heads .....	7
3.4. Groups .....	7
3.5. Transformations .....	7
3.5.1. Translate .....	7
3.5.2. Set Origin .....	7
3.5.3. Rotate .....	8
3.5.4. Scale .....	8
4. Coordinate Systems .....	8
4.1. XYZ .....	8
4.2. Previous .....	9
4.3. Relative .....	9
4.4. Polar .....	10
4.5. Barycentric .....	10
4.6. Anchor .....	11
4.7. Tangent .....	12
4.8. Perpendicular .....	13
4.9. Interpolation .....	13
4.10. Function .....	15

## 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!

## 2. Usage

This is the minimal starting point:

```
#import "typst-canvas/canvas.typ": canvas

#canvas({
  import "typst-canvas/draw.typ": *
  ...
})
```

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. Argument Types

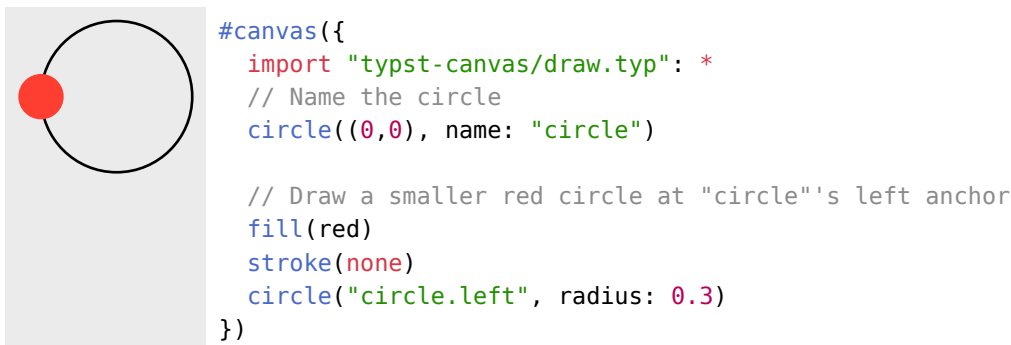
Argument types in this document are formatted in monospace and encased in angle brackets `<>`. Types such as `<integer>` and `<content>` are the same as Typst but additional are required:

**<coordinate>** Any coordinate system. See Section 4.  
**<number>** `<integer>` or `<float>`

### 2.2. Anchors

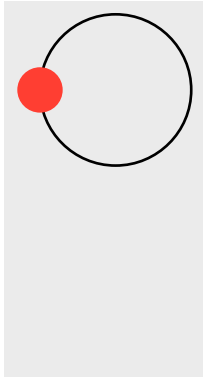
Anchors are named positions relative to named elements.

To use an anchor of an element, you must give the element a name using the `name` argument.



All elements will have default anchors based on their bounding box, they are: center, left, right, above/top and below/bottom, top-left, top-right, bottom-left, bottom-right. Some elements will have their own anchors.

Elements can be placed relative to their own anchors.



```
#canvas({
  import "typst-canvas/draw.typ": *
  // An element does not have to be named
  // in order to use its own anchors.
  circle((0,0), anchor: "left")

  // Draw a smaller red circle at the origin
  fill(red)
  stroke(none)
  circle((0,0), radius: 0.3)
})
```

## 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`.

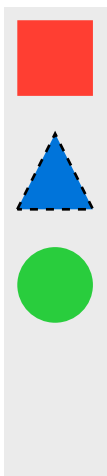
**body**

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

### 3.2. Styling

The fill and stroke of drawn elements can be set globally by using the `fill(color)` and `stroke(stroke)` functions. See the fill and stroke parameters of Typst's path function to see the types accepted (<https://typst.app/docs/reference/visualize/path/>). You can set the fill and stroke of individual elements by using their fill and stroke arguments.

Styling options of drawn elements can be set globally by using the `set-style()` function.



```
#canvas({
  import "typst-canvas/draw.typ": *
  // Set the global fill to red
  fill(red)
  // Set no global stroke
  stroke(none)
  // Draws a red rectangle
  rect((0,0), (1,1))
  // Draws a blue triangle with dashed edges
  line((0, -1.5), (0.5, -0.5), (1, -1.5),
    close: true, fill: blue, stroke: (dash: "dashed"))
  // Draws a green circle
  circle((0.5, -2.5), radius: 0.5, fill: green)
})
```

### 3.3. Elements

#### 3.3.1. Line

Draws a line (a direct path between two points) to the canvas. If multiple coordinates are given, a line is drawn between each consecutive one.

```
#line(..pts, mark-begin: none, mark-end: none, mark-size: auto, name: none, fill: auto, stroke: auto)
```

**..pts** <arguments of coordinates>

Coordinates to draw the lines between. A minimum of two must be given.

**mark-begin** <string>

The type of arrow to draw at the start of the line. See Section 3.3.8.

**mark-end** <string>

The type of arrow to draw at the end of the line.

**mark-size** <number>

The size of the marks.

(default: 0.15)

**name** <string>

Sets the name of element for use with anchors.

**fill** none or auto or <color>

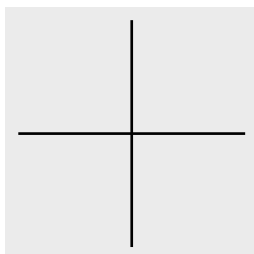
Sets the fill of the path of lines. If auto the global fill is used. See Section 3.2.

(default: **auto**)

**stroke** none or auto or <length> or <color> or <color> or <dictionary> or <stroke>

Sets the stroke of the lines. If auto the global stroke is used. See Section 3.2.

(default: **auto**)



```
#canvas({
  import "typst-canvas/draw.typ": *
  line((-1.5, 0), (1.5, 0))
  line((0, -1.5), (0, 1.5))
})
```

#### 3.3.2. Rectangle

Draws a rectangle to the canvas.

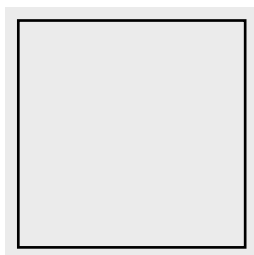
```
#rect(a, b, name: none, fill: auto, stroke: auto)
```

**a** <coordinate>

The top left coordinate of the rectangle.

**b** <coordinate>

The bottom right coordinate of the rectangle.



```
#canvas({
  import "typst-canvas/draw.typ": *
  rect((-1.5, 1.5), (1.5, -1.5))
})
```

### 3.3.3. Arc

Draws an arc to the canvas. Exactly two of the three values start, stop, and delta should be defined.

```
#arc(position, start: auto, stop: auto, delta: auto, radius: 1, name: none, anchor: none, fill: auto, stroke: auto, mode: "OPEN")
```

**position** <coordinate>

The coordinate to start drawing the arc from.

**start** <angle>

The angle to start the arc.

**stop** <angle>

The angle to stop the arc.

**delta** <angle>

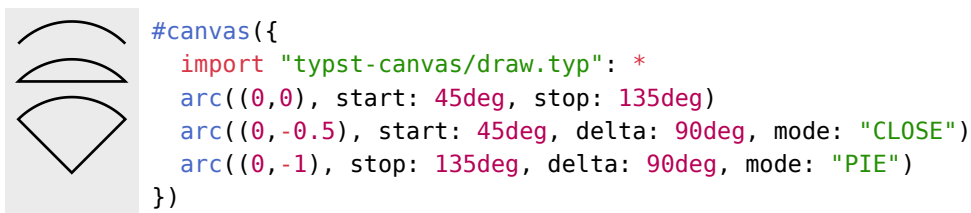
The angle that is added to start or removed from stop.

**radius** <number>

The radius of the arc.

**mode** <string>

The options are “OPEN” (the default, just the arc), “CLOSE” (a circular segment) and “PIE” (a circular sector).



### 3.3.4. Circle

Draws a circle to the canvas. An ellipse can be drawn by passing an array of length two to the radius argument to specify its x and y radii.

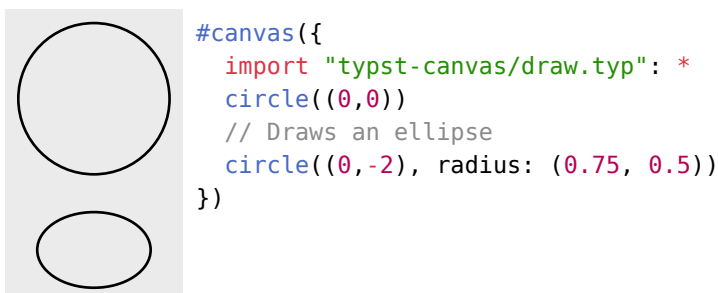
```
#circle(center, radius: 1, name: none, anchor: none, fill: auto, stroke: auto)
```

**center** <coordinate>

The coordinate of the circle's origin.

**radius** <number> or <length> or <array of <number> or <length>> (default: 1)

The circle's radius. If an array is given an ellipse will be drawn where the first item is the x radius and the second item is the y radius.



### 3.3.5. Bezier

Draws a bezier curve with 1 or 2 control points to the canvas.

```
#bezier(start, end, ..ctrl, samples: 100, name: none, fill: auto, stroke: auto)
```

**start** <coordinate>

The coordinate to start drawing the bezier curve from.

**end** <coordinate>


The coordinate to draw the bezier curve to.

**..ctrl** <coordinate>

An array of one or two coordinates to specify the control points of the bezier curve.

**samples** <integer>

The number of lines used to construct the curve.



```
#canvas({
  import "typst-canvas/draw.typ": *
  bezier((0, 0), (2, 0), (1, 1))
  bezier((0, -1), (2, -1), (.5, -2), (1.5, 0))
})
```

### 3.3.6. Content

Draws a content block to the canvas.

```
#content(pt, ct, angle: 0deg, name: none, anchor: none)
```

**pt** <coordinate>

The coordinate of the center of the content block.

**ct** <content>

The content block.

**angle** <angle>

The angle to rotate the content block by. Uses Typst's rotate function.



```
#canvas({
  import "typst-canvas/draw.typ": *
  content((0,0), [Hello World!])
})
```

### 3.3.7. Grid

Draws a grid to the canvas.

```
#grid(from, to, step: 1, help-lines: false, name: none, fill: auto, stroke: auto)
```

**from** <coordinate>

Specifies the bottom left position of the grid.

**to** <coordinate>

Specifies the top right position of the grid.

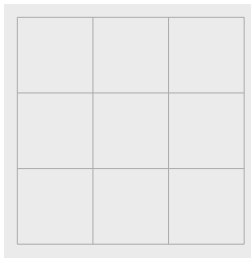
**step** <number> or <length> or <array of <number> or <length>>

The stepping in both  $x$  and  $y$  directions. An array can be given to specify the stepping for each direction.

**help-lines** <bool>

(default: false)

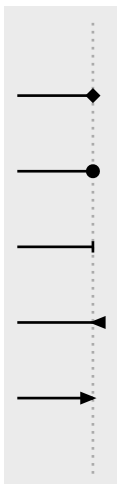
Styles the grid to look “subdued” by using thin gray lines (0.2pt + gray)



```
#canvas({
  import "typst-canvas/draw.typ": *
  grid((0,0), (3,2), help-lines: true)
})
```

### 3.3.8. Arrow Heads

Arrow heads – *marks* – can be drawn using the arrow-head function or as start/end marks of paths (line). Arrow heads are filled using the current fill color.

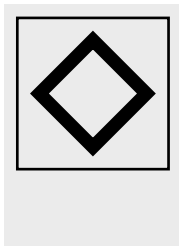


```
#canvas({
  import "typst-canvas/draw.typ": *
  line((1, -1), (1, 5), stroke: (paint: gray, dash: "dotted"))
  set-style(mark: (fill: black))
  line((0, 4), (1, 4), mark: (end: "<>"))
  line((0, 3), (1, 3), mark: (end: "o"))
  line((0, 2), (1, 2), mark: (end: "|"))
  line((0, 1), (1, 1), mark: (end: "<"))
  line((0, 0), (1, 0), mark: (end: ">"))
})
```

## 3.4. Groups

Groups allow scoping context changes such as setting stroke-style, fill and transformations.

```
#group(content, name: none)
```



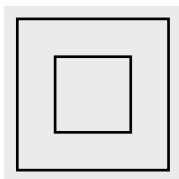
```
// Create group
group({
  stroke(5pt)
  scale(.5); rotate(45deg)
  rect((-1,-1),(1,1))
})
rect((-1,-1),(1,1))
```

## 3.5. Transformations

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

### 3.5.1. Translate

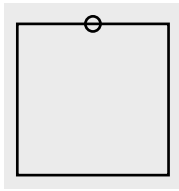
```
#translate(coordinate)
```



```
// Outer rect
rect((0,0), (2,2))
// Inner rect
translate((.5,.5,0))
rect((0,0), (1,1))
```

### 3.5.2. Set Origin

```
#set-origin(position)
```

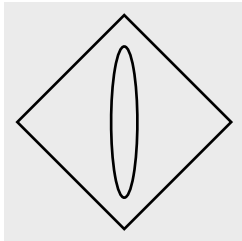


```
// Outer rect
rect((0,0), (2,2), name: "r")
// Move origin to top edge
set-origin("r.above")
circle((0, 0), radius: .1)
```

### 3.5.3. Rotate

```
#rotate(axis-dictionary)
```

```
#rotate(z-angle)
```

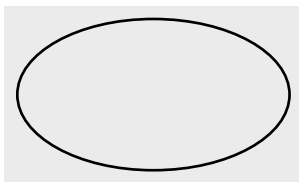


```
// Rotate on z-axis
rotate((z: 45deg))
rect((-1,-1), (1,1))
// Rotate on y-axis
rotate((y: 80deg))
circle((0,0))
```

### 3.5.4. Scale

```
#scale(axis-dictionary)
```

```
#scale(factor)
```



```
// Scale x-axis
scale((x: 1.8))
circle((0,0))
```

## 4. Coordinate Systems

A *coordinate* is a position on the canvas on which the picture is drawn. They take the form of dictionaries and the following sub-sections define the key value pairs for each system. Some systems have a more implicit form as an array of values and `typst-canvas` attempts to infer the system based on the element types.

### 4.1. XYZ

Defines a point  $x$  units right,  $y$  units upward, and  $z$  units away.

**x** <number> or <length> (default: 0)

The number of units in the  $x$  direction.

**y** <number> or <length> (default: 0)

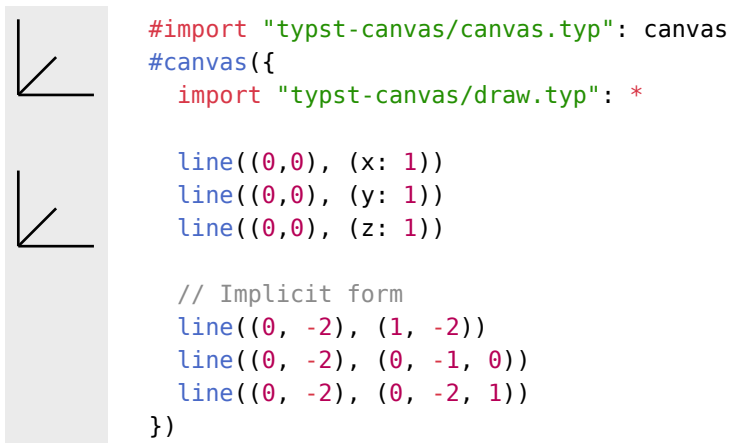
The number of units in the  $y$  direction.

**z** <number> or <length> (default: 0)

The number of units in the  $z$  direction.

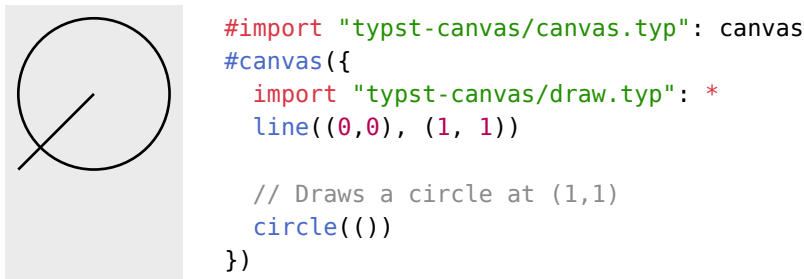
The implicit form can be given as an array of two or three <number> or <length>, as in  $(x,y)$  and  $(x,y,z)$ .





## 4.2. Previous

Use this to reference the position of the previous coordinate passed to a draw function. This will never reference the position of a coordinate used in to define another coordinate. It takes the form of an empty array (). The previous position initially will be (0, 0, 0).



## 4.3. Relative

Places the given coordinate relative to the previous coordinate. Or in other words, for the given coordinate, the previous coordinate will be used as the origin. Another coordinate can be given to act as the previous coordinate instead.

**rel** <coordinate>

The coordinate to be place relative to the previous coordinate.

**update** <bool>

(default: **true**)

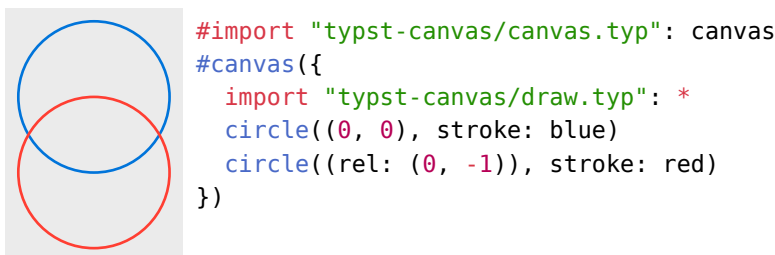
When false the previous position will not be updated.

**to** <coordinate>

(default: ())

The coordinate to treat as the previous coordinate.

In the example below, the red circle is placed one unit below the blue circle. If the blue circle was to be moved to a different position, the red circle will move with the blue circle to stay one unit below.



#### 4.4. Polar

Defines a point a radius distance away from the origin at the given angle. An angle of zero degrees. An angle of zero degrees is to the right, a degree of 90 is upward.

**angle** <angle>

The angle of the coordinate.

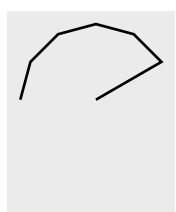
**radius** <number> or <length> or <array of length or number>

The distance from the origin. An array can be given, in the form (x, y) to define the x and y radii of an ellipse instead of a circle.



```
#import "typst-canvas/canvas.typ": canvas
#canvas({
  import "typst-canvas/draw.typ": *
  line((0,0), (angle: 30deg, radius: 1cm))
})
```

The implicit form is an array of the angle then the radius (angle, radius) or (angle, (x, y)).



```
#import "typst-canvas/canvas.typ": canvas
#canvas({
  import "typst-canvas/draw.typ": *
  line((0,0), (30deg, 1), (60deg, 1),
    (90deg, 1), (120deg, 1), (150deg, 1), (180deg, 1))
})
```

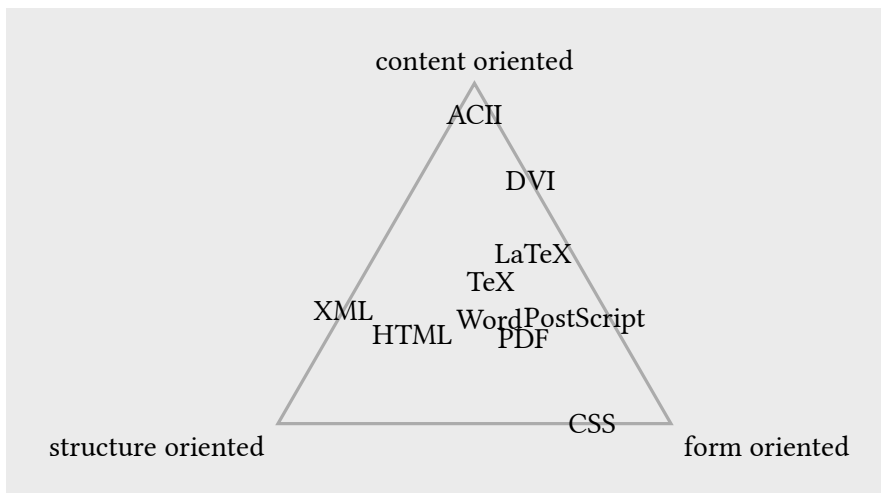
#### 4.5. Barycentric

In the barycentric coordinate system a point is expressed as the linear combination of multiple vectors. The idea is that you specify vectors  $v_1, v_2, \dots, v_n$  and numbers  $\alpha_1, \alpha_2, \dots, \alpha_n$ . Then the barycentric coordinate specified by these vectors and numbers is

$$\frac{\alpha_1 v_1 + \alpha_2 v_2 + \dots + \alpha_n v_n}{\alpha_1 + \alpha_2 + \dots + \alpha_n}$$

**bary** <dictionary>

A dictionary where the key is a named element and the value is a <float>. The center anchor of the named element is used as  $v$  and the value is used as  $a$ .



```
#import "typst-canvas/canvas.typ": canvas
#canvas({
  import "typst-canvas/draw.typ": *
  circle((90deg, 3), radius: 0, name: "content")
  circle((210deg, 3), radius: 0, name: "structure")
  circle((-30deg, 3), radius: 0, name: "form")

  for (c, a) in (
    ("content", "bottom"),
    ("structure", "top-right"),
    ("form", "top-left")
  ) {
    content(c, box(c + " oriented", inset: 5pt), anchor: a)
  }

  stroke(gray + 1.2pt)
  line("content", "structure", "form", close: true)

  for (c, s, f, cont) in (
    (0.5, 0.1, 1, "PostScript"),
    (1, 0, 0.4, "DVI"),
    (0.5, 0.5, 1, "PDF"),
    (0, 0.25, 1, "CSS"),
    (0.5, 1, 0, "XML"),
    (0.5, 1, 0.4, "HTML"),
    (1, 0.2, 0.8, "LaTeX"),
    (1, 0.6, 0.8, "TeX"),
    (0.8, 0.8, 1, "Word"),
    (1, 0.05, 0.05, "ACII")
  ) {
    content((bary: (content: c, structure: s, form: f)), cont)
  }
})
```

## 4.6. Anchor

Defines a point relative to a named element using anchors, see Section 2.2.

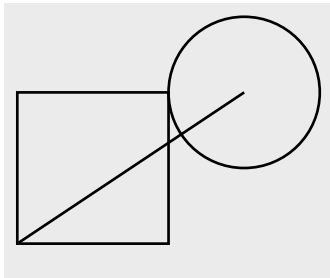
**name** <string>

The name of the element that you wish to use to specify a coordinate.

**anchor** <string>

An anchor of the element. If one is not given a default anchor will be used. On most elements this is center but it can be different.

You can also use implicit syntax of a dot separated string in the form "name.anchor".



```
#import "typst-canvas/canvas.typ": canvas

#canvas({
  import "typst-canvas/draw.typ": *
  line((0,0), (3,2), name: "line")
  circle("line.end", name: "circle")
  rect("line.start", "circle.left")
})
```

## 4.7. Tangent

This system allows you to compute the point that lies tangent to a shape. In detail, consider an element and a point. Now draw a straight line from the point so that it “touches” the element (more formally, so that it is *tangent* to this element). The point where the line touches the shape is the point referred to by this coordinate system.

**element** <string>

The name of the element on whose border the tangent should lie.

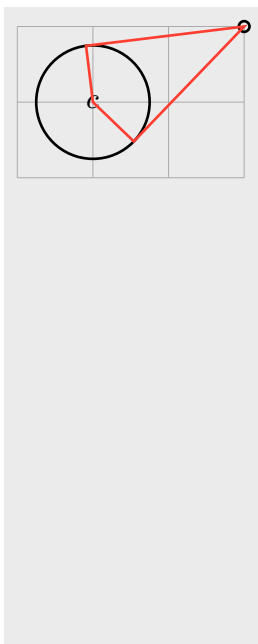
**point** <coordinate>

The point through which the tangent should go.

**solution** <integer>

Which solution should be used if there are more than one.

A special algorithm is needed in order to compute the tangent for a given shape. Currently it does this by assuming the distance between the center and top anchor (See Section 2.2) is the radius of a circle.



```
#import "typst-canvas/canvas.typ": canvas

#canvas({
  import "typst-canvas/draw.typ": *
  grid((0,0), (3,2), help-lines: true)

  circle((3,2), name: "a", radius: 2pt)
  circle((1,1), name: "c", radius: 0.75)
  content("c", $ c $)

  stroke(red)
  line(
    "a",
    (element: "c", point: "a", solution: 1),
    "c",
    (node: "c", point: "a", solution: 2),
    close: true
  )
})
```

## 4.8. Perpendicular

Can be used to find the intersection of a vertical line going through a point  $p$  and a horizontal line going through some other point  $q$ .

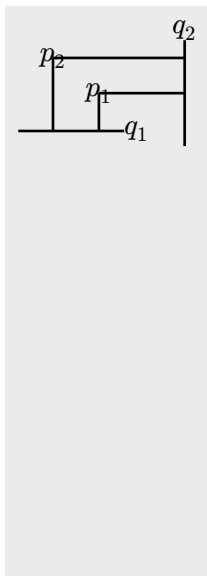
**horizontal** <coordinate>

The coordinate through which the horizontal line passes.

**vertical** <coordinate>

The coordinate through which the vertical line passes.

You can use the implicit syntax of (horizontal, "-|", vertical) or (vertical, "|-", horizontal)



```
#import "typst-canvas/canvas.typ": canvas

#canvas({
  import "typst-canvas/draw.typ": *
  content((30deg, 1), $ p_1 $, name: "p1")
  content((75deg, 1), $ p_2 $, name: "p2")

  line((-0.2, 0), (1.2, 0), name: "xline")
  content("xline.end", $ q_1 $, anchor: "left")
  line((2, -0.2), (2, 1.2), name: "yline")
  content("yline.end", $ q_2 $, anchor: "bottom")

  line("p1", (horizontal: (), vertical: "xline"))
  line("p2", (horizontal: (), vertical: "xline"))
  line("p1", (vertical: (), horizontal: "yline"))
  line("p2", (vertical: (), horizontal: "yline"))
})
```

## 4.9. Interpolation

Use this to linearly interpolate between two coordinates  $a$  and  $b$  with a given factor number. If number is a <length> the position will be at the given distance away from  $a$  towards  $b$ . An angle can also be given for the general meaning: “First consider the line from  $a$  to  $b$ . Then rotate this line by angle around point  $a$ . Then the two endpoints of this line will be  $a$  and some point  $c$ . Use this point  $c$  for the subsequent computation.”

**a** <coordinate>

The coordinate to interpolate from.

**b** <coordinate>

The coordinate to interpolate to.

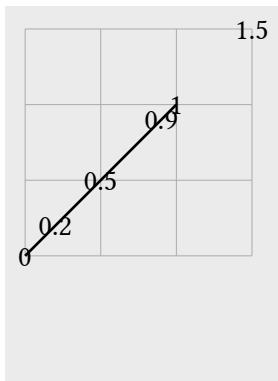
**number** <number> or <length>

The factor to interpolate by or the distance away from  $a$  towards  $b$ .

**angle** <angle>

(default: 0deg)

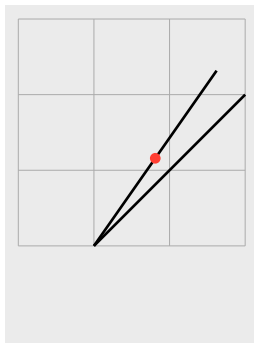
Can be used implicitly as an array in the form ( $a$ , number,  $b$ ) or ( $a$ , number, angle,  $b$ ).



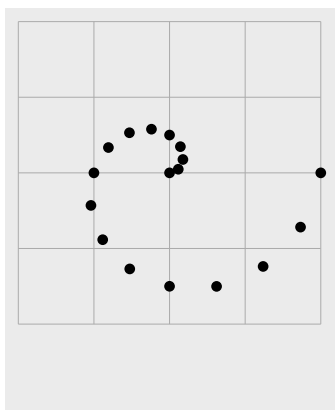
```
#import "typst-canvas/canvas.typ": canvas
#canvas({
  import "typst-canvas/draw.typ": *
  grid((0,0), (3,3), help-lines: true)

  line((0,0), (2,2))

  for i in (0, 0.2, 0.5, 0.9, 1, 1.5) {
    content((0,0), i, (2,2)), [#i]
  }
})
```



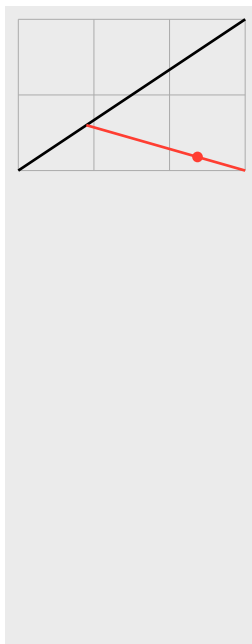
```
#import "typst-canvas/canvas.typ": canvas
#canvas({
  import "typst-canvas/draw.typ": *
  grid((0,0), (3,3), help-lines: true)
  line((1,0), (3,2))
  line((1,0), ((1, 0), 1, 10deg, (3,2)))
  fill(red)
  stroke(none)
  circle(((1, 0), 0.5, 10deg, (3, 2)), radius: 2pt)}
})
```



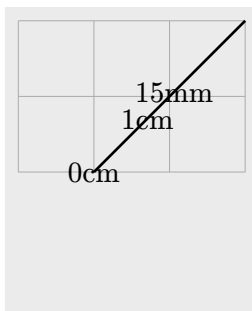
```
#import "typst-canvas/canvas.typ": canvas
#canvas({
  import "typst-canvas/draw.typ": *
  grid((0,0), (4,4), help-lines: true)

  fill(black)
  stroke(none)
  let n = 16
  for i in range(0, n+1) {
    circle(((2,2), i / 8, i * 22.5deg, (3,2)), radius: 2pt)
  }
})
```

You can even chain them together!



```
#import "typst-canvas/canvas.typ": canvas
#canvas({
  import "typst-canvas/draw.typ": *
  grid((0,0), (3, 2), help-lines: true)
  line((0,0), (3,2))
  stroke(red)
  line(((0,0), 0.3, (3,2)), (3,0))
  fill(red)
  stroke(none)
  circle(
    (
      // a
      (((0, 0), 0.3, (3, 2))),
      0.7,
      (3,0)
    ),
    radius: 2pt
  )
})
```

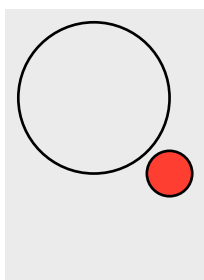


```
#import "typst-canvas/canvas.typ": canvas
#canvas({
  import "typst-canvas/draw.typ": *
  grid((0,0), (3, 2), help-lines: true)
  line((1,0), (3,2))
  for (l, c) in ((0cm, "0cm"), (1cm, "1cm"), (15mm, "15mm")) {
    content(((1,0), l, (3,2)), $ #c $)
  }
})
```

## 4.10. Function

An array where the first element is a function and the rest are coordinates will cause the function to be called with the resolved coordinates. The resolved coordinates have the same format as the implicit form of the 3-D XYZ coordinate system, Section 4.1.

The example below shows how to use this system to create an offset from an anchor, however this could easily be replaced with a relative coordinate with the `to` argument set, Section 4.3.



```
#import "typst-canvas/canvas.typ": canvas
#import "typst-canvas/vector.typ"
#canvas({
  import "typst-canvas/draw.typ": *
  circle((0, 0), name: "c")
  fill(red)
  circle((v => vector.add(v, (0, -1)), "c.right"), radius: 0.3)
})
```