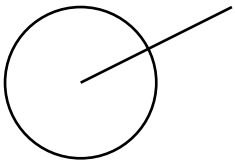
Quick Start

To use the CeTZ, the minimal starting point in a `.typ` file is:

```
#import "@preview/cetz:0.3.3"
#cetz.canvas({
  import cetz.draw: *
  ...
})
```

For example, to draw a circle and a line:



```
#cetz.canvas({
  import cetz.draw: *
  circle((0, 0))
  line((0, 0), (2, 1))
})
```

Custom Typst

Many CeTZ functions expect these types as their parameters.

`coordinate`

Refer to coordinate for more help.

`number`

Any of `float`, `int`, or `length`.

`style`

Normally taken in the form of named arguments.

The Canvas

We draw graphics by calling `canvas` function in this way:

```
#cetz.canvas({
  import cetz.draw: *
  ...
})
```

It evaluates the code block and pass the result to the `canvas`function for rendering. Note, it doesn't have typical `width` and `height` parameters. Instead its size will grow and shrink to fit the drawn graphic. By default 1 coordinate unit is `1cm`, and can be changed by setting the `length` parameters. `length` will be the size of the canvas' parent's width if a ratio if given.

Styling

Here are some styles that can be passed to draw functions with named arguments:

`fill: color or none`

Default: none

`stroke: none or auto or length or color or dictionary or black`

Default: black

How to stroke the border or the path of the draw element. Refer to Typst's line documentation for more details.

`fill-rule: string`

Default: “non-zero”

How to fill self-intersecting paths. Can be “non-zero” or “even-odd”. Refer to Typst’s path documentation for more details.

tips

To avoid specifying the same styling each time, We can use the `set-style` function, which works like a Typst `set` rule. They can be override as well.

When using a dictionary for a style, it update each other instead overriding the entire option. For example:

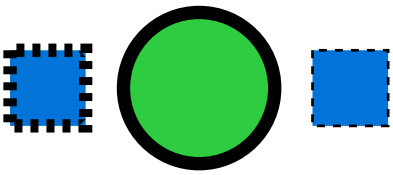
```
#cetz.canvas({
  import cetz.draw: *
  // Sets the stroke to red with a thickness of 5pt
  set-style(stroke: (paint: red, thickness: 5pt))

  // 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.



```
#cetz.canvas({
  import cetz.draw: *
  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((2.5, 0.5))
  rect((4, 0), (5, 1), stroke: (thickness: 1pt))
})
```

Coordinate

A `coordinate` is a position on the canvas on which the picture is draw. They take the form of dictionaries and following sub-sections define the key value pairs for each system.

XYZ

`x: number`
Default: “0”
Points right

`y: number`
Default: “0”
Points upward

`z: number`
Default: “0”
Points away

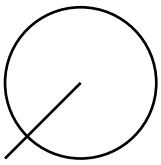
An array like `(x, y)` or `(x, y, z)` is also available.

```
#cetz.canvas({
  import cetz.draw: *
  line((0,0), (x: 1), stroke: blue)
  line((0,0), (y: 1), stroke: green)
  line((0,0), (z: 1), stroke: red)

  // Implicit form
  line((2, 0), (3, 0), stroke: blue)
  line((2, 0), (2, 1, 0), stroke: green)
  line((2, 0), (2, 0, 1), stroke: red)
})
```

Previous

Takes an empty array `()` and passes the previous coordinate to a draw function. The previous position initially will be `(0, 0, 0)`. “This will never reference the position of a coordinate used to define another coordinate.”



```
#cetz.canvas({
  import cetz.draw: *
  line((0,0), (1, 1))

  // Draws a circle at (1,1)
  circle(())
})
```

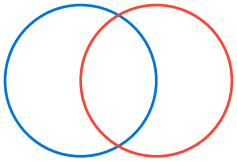
Relative

Places the given coordinate relative to the previous coordinate.

`rel: coordinate`
The coordinate to place relative to the previous coordinate

`update: bool`
Default: `true`
Whether update the previous position.

`to: coordinate`
Default: `()`
The coordinate to treat as the previous coordinate.



```
#cetz.canvas({
  import cetz.draw: *
  circle((0, 0), stroke: blue)
  circle((rel: (1, 0)), stroke: red)
})
```

Polar

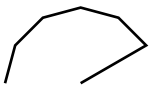
Defines a point that is `radius` distance away from the origin at the given angle.

`angle: angle`

`0deg` is to the right, `90deg` is upward.

`radius: number or array`

The distance from the origin. An array of number can be given in the form `(x, y)` to define the `x` and `y` radii of an ellipse instead of a circle. The implicit form is an array of the angle then the radius `(angle, radius)` or `(angle, (x, y))`.



```
#cetz.canvas({
  import cetz.draw: *
  line(
  (0, 0),
  (30deg, 1),
  (60deg, 1),
  (90deg, 1),
  (120deg, 1),
  (150deg, 1),
  (180deg, 1)
)
})
```

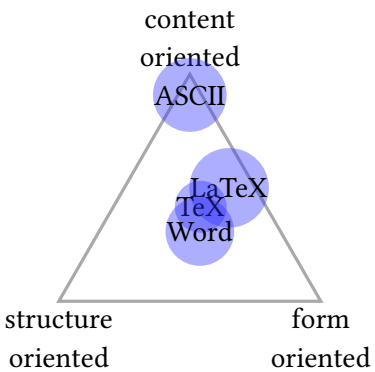
Barycentric

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

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

`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 α.




```
#cetz.canvas({
  import cetz.draw: *
  circle((90deg, 2), radius: 0, name: "content")
  circle((210deg, 2), radius: 0, name: "structure")
  circle((-30deg, 2), radius: 0, name: "form")

  for (c, a) in (
    ("content", "south"),
    ("structure", "north"),
    ("form", "north")
  ) {
    content(c, align(center, c + [\ oriented]),
padding: .1, anchor: a)
  }
```

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

    for (c, s, f, cont) in (
      (1, 0.2, 0.8, "LaTeX"),
      (1, 0.6, 0.8, "TeX"),
      (0.8, 0.8, 1, "Word"),
      (1, 0.05, 0.05, "ASCII")
    ) {
      content(
        (bary: ( content: c, structure: s, form: f)),
        cont,
        fill: rgb(50, 50, 255, 100),
        stroke: none,
        frame: "circle"
      )
    }
  })
```

Anchor

Defines a point relative to a named element using anchors, See anchor.

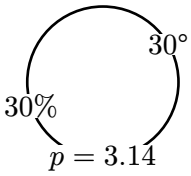
`name: str`

The name of the element.

`anchor: str or angle or number or ratio or none`

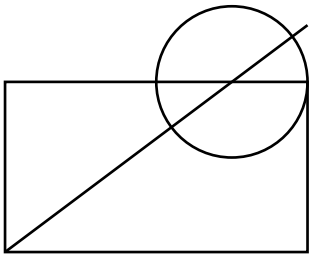
Default: none

The anchor of the element. The default anchor is mostly `center`, but not always.




```
    #cetz.canvas({
      import cetz.draw: *
      circle((0,0), name: "circle")
      // Anchor at 30 degree
      content((name: "circle", anchor: 30deg), box(fill: white, $ 30
degree $))
      // Anchor at 30% of the path length
      content((name: "circle", anchor: 30%), box(fill: white, $ 30 %
$))
      // Anchor at 3.14 of the path
      content((name: "circle", anchor: 3.14), box(fill: white, $ p =
3.14 $))
    })
```

Anchors in the form of `"name.anchor"` are also available.



```
#cetz.canvas({
  import cetz.draw: *
  line((0, 0), (4, 3), name: "line")
  circle("line.75%", name: "circle")
  rect("line.start", "circle.east")
})
```