

mannot v0.2.3

A package for marking and annotating in math blocks in Typst.

Contents

1	Example	1
2	Usage	2
2.1	Marking	2
2.2	Annotations	2
2.2.1	Annotation Positioning	3
2.2.2	Annotation Leader Line	3
2.3	Multi Annotations	4
2.4	Annotations using CeTZ	4
3	Limitations	5
4	API	5
	core-mark	5
	mark	7
	markhl	8
	markrect	9
	markul	11
	core-annot	12
	annot	13
	annot-cetz	15

1 Example

The diagram shows the Boltzmann distribution formula:
$$p_i = \frac{\exp(-\beta E_i)}{\sum_j \exp(-\beta E_j)}$$
 Annotations include: "Inverse temperature" (red) pointing to β , "Boltzmann factor" (blue) pointing to β , "Energy" (green) pointing to E_i , "Probability of state i " (black) pointing to p_i , and "Partition function" (orange) pointing to the denominator.

```
1 #set text(12pt)
2 #v(2em)
3 $
4 markul(p_i, tag: #<p>)
5 = markrect(
6   exp(- mark(beta, tag: #<beta>, color: #red) mark(E_i, tag: #<E>, color: #green)),
7   tag: #<Boltzmann>, color: #blue,
8 ) / markhl(sum_j exp(- beta E_j), tag: #<Z>)
9
10 #annot(<p>, pos: bottom + left)[Probability of \ state $i$]
11 #annot(<beta>, pos: top + left, dy: -1.5em, leader-connect: "elbow")[Inverse
    temperature]
12 #annot(<E>, pos: top + right, dy: -1em)[Energy]
13 #annot(<Boltzmann>, pos: top + left)[Boltzmann factor]
14 #annot(<Z>)[Partition function]
15 $
16 #v(1em)
```

2 Usage

Import the package mannot on the top of your document:

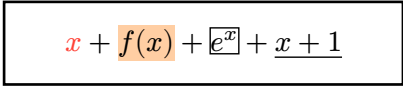
```
1 #import "@preview/mannot:0.2.3": *
```

2.1 Marking

To decorate content within math blocks, use the following marking functions:

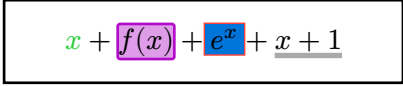
- mark: Changes the text color.
- markhl: Highlights the content.
- markrect: Draws a rectangle around the content.
- markul: Underlines the content.

```
1 $
2 mark(x, color: #red) + markhl(f(x))
3 + markrect(e^x) + markul(x + 1)
4 $
```



You can customize the marking color and other styles:

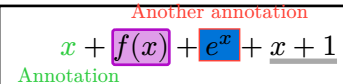
```
1 $
2 mark(x, color: #green)
3 + markhl(f(x), color: #purple, stroke: #1pt, radius:
4   #10%)
5 + markrect(e^x, color: #red, fill: #blue, outset: #.2em)
6 + markul(x + 1, color: #gray, stroke: #2pt)
7 $
```



2.2 Annotations

If you marked content with a tag, you can later annotate it using the annot function:

```
1 $
2 mark(x, tag: #<1>, color: #green)
3 + markhl(f(x), tag: #<2>, color: #purple, stroke: #1pt,
4   radius: #10%)
5 + markrect(e^x, tag: #<3>, color: #red, fill: #blue,
6   outset: #.2em)
7 + markul(x + 1, tag: #<4>, color: #gray, stroke: #2pt)
8
9 #annot(<1>)[Annotation]
10 #annot(<3>, pos: top)[Another annotation]
11 $
```



Markings and annotations do not affect the layout, so you might sometimes need to manually insert spacing before and after the equations to achieve the desired visual appearance:

```
1 Text text text text text:
2 #v(1em)
3 $
4 mark(x, tag: #<1>, color: #green)
5 #annot(<1>, pos: top + right)[Annotation]
6 #annot(<1>, dy: 1em)[Annotation]
7 $
```

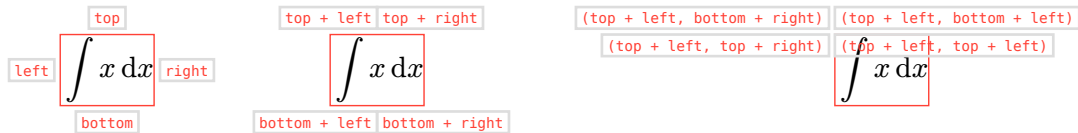


```
8 #v(2em)
9 text text text text text.
```

2.2.1 Annotation Positioning

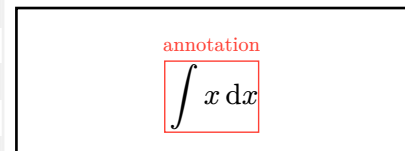
The `annot` function offers the following arguments to control annotation placement:

- `pos`: Where to place the annotation relative to the marked content. This can be:
 - A single alignment for simple relative positioning.
 - A pair of alignments, where the first defines the anchor point on the marked content, and the second defines the anchor point on the annotation.

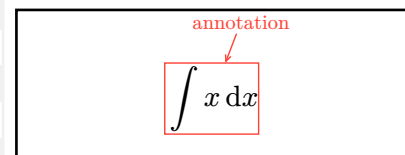


- `dx`, `dy`: the horizontal and vertical displacement of the annotation's anchor from the marked content's anchor.

```
1 #v(1em)
2 $
3 markrect(integral x dif x, tag: #<1>, color: #red)
4 #annot(<1>, pos: top)[annotation]
5 $
```



```
1 #v(1em)
2 $
3 markrect(integral x dif x, tag: #<1>, color: #red)
4 #annot(<1>, pos: top, dx: 1em, dy: -1em)[annotation]
5 $
```

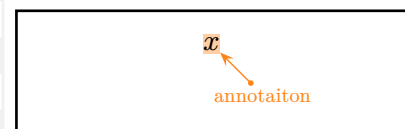


2.2.2 Annotation Leader Line

When the annotation is far from the marked content, a leader line is drawn by default. You can customize its appearance using the following `annot` arguments:

- `leader`: Whether to draw a leader line. Set to `auto` to enable automatic drawing based on distance.
- `leader-stroke`: How to stroke the leader line e.g. `1pt + red`.
- `leader-tip`, `leader-toe`: Define the end and start markers of the leader line. Leader lines are drawn by package `tiptoe`. You can specify markers or `none`:

```
1 $
2 markhl(x, tag: #<1>)
3
4 #annot(
5   <1>, pos: bottom + right, dy: 1em,
6   leader-tip: tiptoe.circle,
7   leader-toe: tiptoe.stealth.with(length: 1000%),
8 ) [annotaiton]
9 $
10 #v(2em)
```



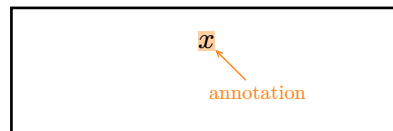
For more options, see the [tiptoe](#) page.

- leader-connect: How the leader line connects to the marked content and the annotation. This can be:
 - A pair of alignments defining the connection points on the marked content and the annotation.
 - “elbow” to create an elbow-shaped leader line.

```

1  $
2  markhl(x, tag: #<1>)
3  #annot(<1>, pos: bottom + right, dy: 1em)[annotation]
4  $
5  #v(2em)

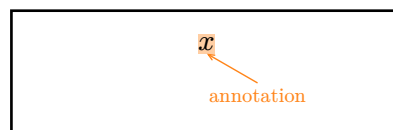
```



```

1  $
2  markhl(x, tag: #<1>)
3  #annot(<1>, pos: bottom + right, dy: 1em, leader-
4  connect: (bottom, top))[annotation]
5  $
6  #v(2em)

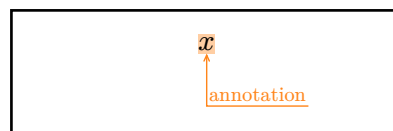
```



```

1  $
2  markhl(x, tag: #<1>)
3  #annot(<1>, pos: bottom + right, dy: 1em, leader-
4  connect: "elbow")[annotation]
5  $
6  #v(2em)

```



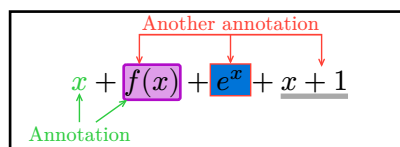
2.3 Multi Annotations

You can also annotate multiple marked elements simultaneously by passing an array of their tags to the annot function.

```

1  #v(1em)
2  $
3  mark(x, tag: #<1>, color: #green)
4  + markhl(f(x), tag: #<2>, color: #purple, stroke: #1pt,
5  radius: #10%)
6  + markrect(e^x, tag: #<3>, color: #red, fill: #blue,
7  outset: #.2em)
8  + markul(x + 1, tag: #<4>, color: #gray, stroke: #2pt)
9  $
10 #annot(<1>, <2>), dy: 1em)[Annotation]
11 #annot(<3>, <2>, <4>), pos: top, dy: -1em, leader-
12 connect: "elbow")[Another annotation]
13 $
14 #v(1em)

```



2.4 Annotations using CeTZ

To create annotations using the CeTZ canvas, use the annot-cetz function. This allows you to embed a CeTZ canvas directly onto previously marked content. Within the CeTZ canvas code block, you can reference the position and dimensions of the marked content using an anchor with the same name as its tag. For elements marked with multiple tags, corresponding anchors will be available.

```

1  #import "@preview/cetz:0.3.4"
2
3  $
4  mark(x, tag: #<x>) + mark(y, tag: #<y>)
5
6  #annot-cetz((<x>, <y>), cetz, {
7      import cetz.draw: *
8      content((0, -.6), [CeTZ], anchor: "north-west", name:
9          "a")
10     line("x", "a") // You can refer the marked content.
11     line("y", "a")
12 })
13 $
14 #v(1em)

```

$$\begin{array}{c} x + y \\ \diagdown \quad \diagup \\ \text{CeTZ} \end{array}$$

3 Limitations

Marking multi-line inline math content is not supported.

```

1  $markhl(x + x + x + x + x + x + x + x + x)$

```

$$\begin{array}{c} x + x + x + x + x + x + \\ x + x \end{array}$$

4 API

- `core-mark()`
- `mark()`
- `markhl()`
- `markrect()`
- `markul()`
- `core-annot()`
- `annot()`
- `annot-cetz()`

core-mark

Marks content within a math block with a custom underlay or overlay.

This function measures the position and size of the marked content, applies a custom underlay or overlay, and generates metadata associated with a given tag. The metadata includes the content's position (x, y), dimensions (width, height), and the color used for the marking. This metadata can be later used for annotations.

Use this function as a foundation for defining custom marking functions.

Example:

```

1  #let mymark(body, tag: none) = {
2    let overlay(width, height, color) = {
3      rect(width: width, height: height, stroke: color)
4    }
5    return core-mark(body, tag: tag, color: red,
6      overlay: overlay, outset: (y: .1em))
7  }
8  $
9  mymark(x, tag: #<e>)
10
11 #context {
12   let info = query(<e>).last()
13   sym.wj
14   box(place(repr(info.value), dx: -5em))
15 }
16 $
17 #v(12em)

```

```

(
  body: [x],
  x: 446.3pt,
  y: 315.92pt,
  width: 6.71pt,
  height: 7.66pt,
  color: rgb("#ff851b"),
  tag: <e>,
  begin-loc: ...,
)

```

Parameters

```

core-mark(
  body: content,
  tag: none or label,
  color: auto or color,
  underlay: none or function,
  overlay: none or function,
  outset: none or length dictionary
) -> content

```

body `content`

The content to be marked within a math block.

tag `none` or `label`

An optional tag to associate with the metadata. This tag can be used to query the marked element.

Default: `none`

color `auto` or `color`

The color used for marking underlay/overlay and later annotations. If set to auto, it defaults to the text fill color.

Default: `auto`

underlay `none` or `function`

An optional function to create a custom underlay. This function receives the marked content's width and height (including outset) and marking color, and should return content to be placed **under** the marked content. The signature is `underlay(width, height, color)`.

Default: `none`

overlay `none` or `function`

An optional function to create a custom overlay. This function receives the marked content's width and height (including outset) and marking color, and should return content to be placed **over** the marked content. The signature is `overlay(width, height, color)`.

Default: `none`

outset `none` or `length` or `dictionary`

How much to expand the marking box size without affecting the layout. This can be specified as a single length value, which applies to all sides, or as a dictionary of length with keys `left`, `right`, `top`, `bottom`, `x`, `y`, or `rest`.

Default: `(:)`

mark

Marks an annotation target within a math block.

If you mark content with a tag, you can annotate it using the `annot` function.

If the `color` argument is provided, it will change the color of the marked text.

Example

```
1 $ mark(x, color: #red) $
```

Parameters

```
mark(  
  body: content,  
  tag: none label,  
  color: auto color,  
  outset: none length dictionary  
) -> content
```

body `content`

The content to be marked within a math block.

tag `none` or `label`

An optional tag used to identify the marked content for later annotations.

Default: `none`

color `auto` or `color`

An optional color for the text and later annotations. If set to `auto`, it defaults to the text fill color.

Default: `auto`

outset `none` or `length` or `dictionary`

How much to expand the marking box size without affecting the layout. This can be specified as a single length value which applies to all sides, or as a dictionary of `length` with keys `left`, `right`, `top`, `bottom`, `x`, `y`, or `rest`.

Default: (`y`: `.1em`)

markhl

Marks and highlights content within a math block.

If you mark content with a tag, you can annotate it using the `annot` function.

Example

```
1 $ markhl(x) $
```



Parameters

```
markhl(  
  body: content,  
  tag: none label,  
  color: auto color,  
  fill: auto none color gradient pattern,  
  stroke: none auto length color gradient stroke pattern dictionary,  
  radius: relative dictionary,  
  outset: none length dictionary  
) -> content
```

body `content`

The content to be highlighted within a math block.

tag `none` or `label`

An optional tag used to identify the marked content for later annotations.

Default: `none`

color `auto` or `color`

The color used for the highlight and later annotations. If both `color` and `fill` are set to `auto`, `color` defaults to orange. Otherwise, if only `color` is `auto`, it defaults to the text fill color.

Default: `auto`

fill `auto` or `none` or `color` or `gradient` or `pattern`

How to fill the highlight rectangle. If set to `auto`, `fill` will be set to `color.transparentize(60%)`.

Default: `auto`

stroke `none` or `auto` or `length` or `color` or `gradient` or `stroke` or `pattern` or `dictionary`

How to stroke the highlight rectangle. If its paint is set to `auto`, it will be set to the color. If its thickness is set to `auto`, it defaults to `.048em`.

Default: `none`

radius `relative` or `dictionary`

How much to round the highlight rectangle's corner.

Default: `(:)`

outset `none` or `length` or `dictionary`

How much to expand the highlight box size without affecting the layout. This can be specified as a single length value which applies to all sides, or as a dictionary of length with keys `left`, `right`, `top`, `bottom`, `x`, `y`, or `rest`.

Default: `(y: .1em)`

markrect

Marks and boxes around content within a math block.

If you mark content with a tag, you can annotate it using the `annot` function.

Example

```
1 $ markrect(x + y) $
```

$$x + y$$

Parameters

```
markrect(  
  body: content,  
  tag: none label,  
  color: color,  
  fill: none color gradient pattern,  
  stroke: none length color gradient stroke pattern dictionary,  
  radius: relative dictionary,  
  outset: none length dictionary  
) -> content
```

body `content`

The content to be boxed around within a math block.

tag `none` or label

An optional tag used to identify the content for later annotations.

Default: `none`

color `color`

The color used for the rectangle's stroke and later annotations. If set to auto, it defaults to the text fill color.

Default: `auto`

fill `none` or `color` or `gradient` or pattern

How to fill the rectangle.

Default: `none`

stroke `none` or `length` or `color` or `gradient` or stroke or pattern or dictionary

How to stroke the rectangle. If its paint is set to auto, it will be set to the color. If its thickness is set to auto, it defaults to `.048em`.

Default: `.048em`

radius `relative` or dictionary

How much to round the highlight rectangle's corner.

Default: `(:)`

outset `none` or `length` or `dictionary`

How much to expand the rectangle size without affecting the layout. This can be specified as a single length value which applies to all sides, or as a dictionary of length with keys left, right, top, bottom, x, y, or rest.

Default: (y: `.1em`)

markul

Marks and underlines content within a math block.

If you mark content with a tag, you can annotate it using the `annot` function.

Example

```
1 $ markul(x + y) $
```


$$\underline{x + y}$$

Parameters

```
markul(  
  body: content ,  
  tag: none label ,  
  color: auto color ,  
  stroke: none length color gradient stroke pattern dictionary ,  
  outset: none length dictionary  
) -> content
```

body `content`

The content to be underlined within a math block.

tag `none` or `label`

An optional tag used to identify the content for later annotations.

Default: `none`

color `auto` or `color`

The color used for the underline and later annotations. If set to `auto`, it defaults to the text fill color.

Default: `auto`

stroke `none` or `length` or `color` or `gradient` or `stroke` or `pattern` or `dictionary`

How to stroke the underline. If its paint is set to auto, it will be set to the color. If its thickness is set to auto, it defaults to `.048em`.

Default: `.048em`

outset `none` or `length` or `dictionary`

How much to expand the marking box size without affecting the layout. This can be specified as a single length value which applies to all sides, or as a dictionary of length with keys `left`, `right`, `top`, `bottom`, `x`, `y`, or `rest`.

Default: (top: `.1em`, bottom: `.144em`)

core-annot

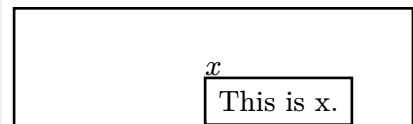
Places a custom annotation on content (or contents) within a math block that was previously marked.

This function generates a custom annotation by applying an overlay based on metadata associated with content marked with a specific tag using a marking function.

Use this function as a foundation for defining custom annotation functions.

Example

```
1 #let myannot(tag, annotation) = {  
2   let a = rect(annotation)  
3   let overlay(markers) = {  
4     let m = markers.first()  
5     place(dx: m.x, dy: m.y + m.height, a)  
6   }  
7   return core-annot(tag, overlay)  
8 }  
9  
10 $  
11 mark(x, tag: #<e>)  
12 #myannot(<e>)[This is x.]  
13 $
```



Parameters

```
core-annot(  
  tag: label array ,  
  overlay: function  
) -> content
```

tag `label` or `array`

The tag (or an array of tags) associated with the content to annotate. This tag must match the tag previously used in a marking function.

overlay function

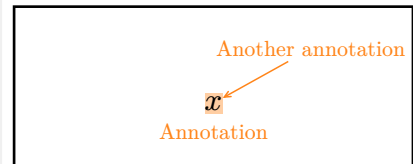
The function to create the custom annotation overlay. This function receives a list of metadata for the marked contents, and should return content to be placed relative to the top-left corner of the page.

annot

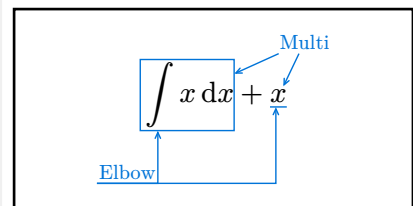
Places an annotation on content (or contents) within a math block that was previously marked.

Example

```
1 #v(2em)
2 $
3 markhl(x, tag: #<e>)
4 #annot(<e>)[Annotation]
5 #annot(<e>, pos: top + right, dy: -1em)[Another
  annotation]
6 $
7 #v(1em)
```



```
1 $
2 markrect(integral x dif x, tag: #<0>, color: #blue)
3 + markul(x, tag: #<1>, color: #blue)
4
5 #annot((<0>, <1>), pos: top, dx: 4em)[Multi]
6 #annot((<0>, <1>), pos: bottom + left, dx: -1em, dy: 1em,
  leader-connect: "elbow")[Elbow]
7 $
8 #v(1em)
```



Parameters

```
annot(
  tag: label array ,
  annotation: content ,
  pos: alignment (alignment, alignment) ,
  dx: auto length ,
  dy: auto length ,
  leader: auto bool ,
  leader-stroke,
  leader-tip,
  leader-toe,
  leader-connect,
  annot-inset: length dictionary ,
  annot-alignment: auto alignment ,
  annot-text-props: dictionary ,
  annot-par-props: dictionary
) -> content
```

tag `label` or `array`

The tag (or an array of tags) associated with the content to annotate. This tag must match the tag previously used in a marking function.

annotation `content`

Content of the annotation.

pos `alignment` or `(alignment, alignment)`

Where to place the annotation relative to the marked content. This can be:

- A single alignment for the relative position to the marked content.
- A pair of alignments. The first one describes the anchor of the marked content, and the second is the anchor of the annotation.

Default: `bottom`

dx `auto` or `length`

The horizontal displacement of the annotation.

Default: `auto`

dy `auto` or `length`

The vertical displacement of the annotation.

Default: `auto`

leader `auto` or `bool`

Whether to draw a leader line from the marked content to the annotation. If set to `auto`, it depends on the distance between the marked content and the annotation.

Default: `auto`

leader-stroke

How to stroke the leader line. If its paint is set to `auto`, it will be set to the marking color. If its thickness is set to `auto`, it defaults to `.048em`.

Default: `.048em`

leader-tip

How to end the leader line. See [`tiptoe`](#).

Default: `none`

leader-toe

How to start the leader line. See [tiptoe](#).

Default: `tiptoe.straight.with(length: 600%)`

leader-connect

How to connect the leader line. This can be:

- A pair of alignments describing the start anchor of the marked content and the end anchor of the annotation.
- “elbow” for an elbow-shaped leader line

Default: `(center + horizon, center + horizon)`

annot-inset `length` or `dictionary`

How much to pad the annotation content.

Default: `(x: .08em, y: .16em)`

annot-alignment `auto` or `alignment`

How to align the annotation text.

Default: `auto`

annot-text-props `dictionary`

Properties for the annotation text. If the `fill` property is not specified, it defaults to the marking’s color.

Default: `(size: .7em)`

annot-par-props `dictionary`

Properties for the annotation paragraph.

Default: `(leading: .4em)`

annot-cetz

Places a CeTZ canvas annotation on content (or contents) within a math block that was previously marked.

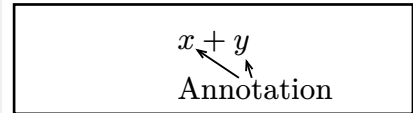
Within the CeTZ canvas code block, you can refer to the position of the marked content using an anchor with the same name as the tag. For multiple tags, you’ll have multiple corresponding anchors.

Example

```

1  #import "@preview/cetz:0.3.4"
2
3  $
4    mark(x, tag: #<0>) + mark(y, tag: #<1>)
5
6    #annot-cetz((<0>, <1>), cetz, {
7      import cetz.draw: *
8      content((0, -.6), [Annotation], anchor: "north-
9        west", name: "a")
10     set-style(stroke: .7pt, mark: (start: "straight",
11       scale: 0.6))
12     line("0", "a")
13     line("1", "a")
14   })
15 $
16 #v(1em)

```



Parameters

```

annot-cetz(
  tag: label array,
  cetz: module,
  drawable: array
) -> content

```

tag label or array

The tag (or an array of tags) associated with the content to annotate. This tag must match the tag previously used in a marking function.

cetz module

A CeTZ module.

drawable array

A code block containing CeTZ drawing commands.