

Codly 1.3.1 manual

Your code blocks on steroids

O RLY?

Contents

1.	,		
		Initialization	
		Enabling and disabling codly	
		Short guide to this manual	
2.	A prim	ner on Codly's show-rule like system	. 4
	2.1.	Enabled (enabled)	
	2.2.	Header (header)	
	2.3.	Header Repeat (header-repeat)	
	2.4.	Header Cell Args (header-cell-args)	. 6
	2.5.	Header Transform (header-transform)	. 6
	2.6.	Footer (footer)	. 7
	2.7.	Footer Repeat (footer-repeat)	. 7
	2.8.	Footer Cell Args (footer-cell-args)	. 7
	2.9.	Footer Transform (footer-transform)	
	2.10.	Offset (offset)	. 8
	2.11.	Offset from other code block (offset-from)	. 9
	2.12.	Range (range)	10
	2.13.	Ranges (ranges)	10
		Smart skip (smart-skip)	
		Languages (languages)	
		Default language color (default-color)	
		Radius (radius)	
		Inset (inset)	
		Fill (fill)	
		Zebra fill (zebra-fill)	
		Stroke (stroke)	
		Language box inset (lang-inset)	
		Language box outset (lang-outset)	
		Language box radius (lang-radius)	
		Language box stroke (lang-stroke)	
		Language box stroke (tang-stroke)	
		Language box formatter (lang-format)	
		Display language name (display-name)	
		Display language icon (display-icon)	
		Line number format (number-format)	
		Line number alignment (number-align)	
		Line number placement (number-placement)	
		Smart indentation (smart-indent)	
		Skip last line if empty (skip-last-empty)	
		Breakable (breakable)	
		Skips (skips)	
		Skip line (skip-line)	
		Skip number (skip-number)	
		Annotations (annotations)	
		Annotation formatter (annotation-format)	
		Highlight lines (highlighted-lines)	
		Highlight lines default color (highlighted-default-color)	
	2.43.	Highlights (highlights)	28
		Highlight radius (highlight-radius)	
		Highlight fill (highlight-fill)	
	2.46.	Highlight stroke (highlight-stroke)	30
	2.47.	Highlightinset(highlight-inset)	30
	2.48.	Highlight outset (highlight-outset)	30
	2.49.	Highlight clip (highlight-clip)	31
	2.50.	Reference by (reference-by)	31
		Reference separator (reference-sep)	
		Reference number format (reference-number-format)	
	2.53.	Language name aliases (aliases)	31

3.	Refer	encing code blocks, highlights, and annotations	. 33
		Shorthand line references	
		Highlight references	
4.		ng nice icons	
		Typst language icon (typst-icon)	
5.		r functions	
		Skip (codly-skip)	
		Range (codly-range)	
		Offset (codly-offset)	
		Local (local)	
		No Codly (no-codly)	
		Yes Codly (yes-codly)	
		Enable (codly-enable)	
		Disable (codly-disable)	
		Reset (codly-reset)	
6.		performance	

1. Codly

Codly is a library that enhances the way you write code blocks in Typst. It provides a set of tools to help you manage your code blocks, highlights them, skip parts of them, and more. This manual will guide you through the different features of Codly, how to use them, and how to integrate them into your Typst projects.



Notification

If you find any issues with Codly, please report them on the GitHub repository: https://github.com/Dherse/codly.

1.1. Initialization

To start using Codly, you must first import it into your Typst project.

```
Example code

1 #import "@preview/codly:1.3.1": *

2

3 #show: codly-init
```

As you can see, this does nothing but initialize codly. You can also import it with a specific version, as shown in the example above. For the latest version, always refer to the <u>Typst Universe page</u>.

From this point on, any code block that is included in your Typst project will be enhanced by Codly.



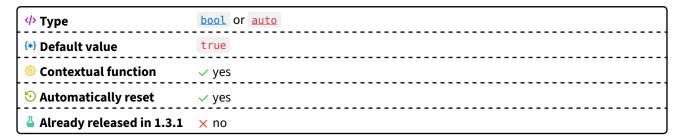
1.2. Enabling and disabling codly

By default Codly will be enabled after initialization. However, disabling codly can be done using the <u>codly-disable</u> function, the <u>enabled</u> argument of the <u>codly</u> function, or the <u>no-codly</u> and <u>yes-codly</u> functions. To enable Codly again, use the <u>codly-enable</u> function or by setting the <u>enabled</u> parameter again.

1.3. Short guide to this manual

Codly can take a lot of different argument to configure your code blocks. Some of these arguments can have pretty complex behaviour, so this manual is here to help you understand how to use them. Each argument is acompagnied by a card that gives you important information about its usage. The card contains the following information:

- the type(s) of values accepted;
- the default value that is pre-set;
- whether the argument can take a contextual function: if it can, you can pass it a function that will be called within a
 typst context block to customize the value of the argument yourself¹;
- whether the argument is automatically reset after the code block is rendered;
- whether the argument is already released in the current version of Codly or will be released in the next version.



Additionally to the card, most arguments are accompanied by an example that shows how to use the argument in a code block. The example is followed by a rendered output of the code block, which shows how the argument affects the code block.

¹This is mostly intended for libraries that wish to extend Codly, or for users that wish to have dynamic control over argument values using their own logic.

2. A primer on Codly's show-rule like system

Codly uses a function called codly to create a kind of show-rule which you can use to configure how your code blocks are displayed. The codly function takes a set of arguments that define how the code block should be displayed. Here is the equivalent definition of the codly function:

```
Typst code
1 let codly(
2
     enabled: true,
3
     offset: 0,
4
     offset-from: none,
5
     range: none,
6
     ranges: (),
7
      languages: (:),
8
     display-name: true,
9
     display-icon: true,
10
     default-color: rgb("#283593"),
11
     radius: 0.32em,
12
     inset: 0.32em,
13
     fill: none,
14
     zebra-fill: luma(240),
15
     stroke: 1pt + luma(240),
16
      lang-inset: 0.32em,
17
     lang-outset: (x: 0.32em, y: 0pt),
18
      lang-radius: 0.32em,
19
     lang-stroke: (lang) => lang.color + 0.5pt,
20
     lang-fill: (lang) => lang.color.lighten(80%),
21
     lang-format: codly.default-language-block,
22
     number-format: (number) => [ #number ],
23
     number-align: left + horizon,
24
     number-placement: "outside",
25
      smart-indent: false,
26
     annotations: none,
27
      annotation-format: numbering.with("(1)"),
28
     highlights: none,
29
     highlight-radius: 0.32em,
30
     highlight-fill: (color) => color.lighten(80%),
     highlight-stroke: (color) => 0.5pt + color,
31
32
     highlight-inset: 0.32em,
33
     highlight-outset: Opt,
34
     highlight-clip: true,
35
     reference-by: line,
36
      reference-sep: "-",
37
     reference-number-format: numbering.with("1"),
38
     header: none,
39
     header-repeat: false,
40
     header-transform: (x) \Rightarrow x,
41
     header-cell-args: (),
42
     footer: none,
43
     footer-repeat: false,
44
     footer-transform: (x) \Rightarrow x,
45
     footer-cell-args: (),
46
      breakable: false,
47 ) = {}
```

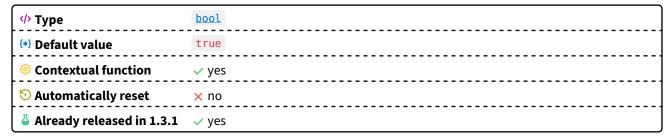
The codly functions acts like a set-rule, this means that calling it will set the configuration for all code blocks that follow it, with the exception of a few arguments that are explicitly set for each code block. To perform changes locally, you can use the local function, or set the arguments before the code block and reset them after to their previous values.

Warning

Unlike regular set-rules in native Typst, there are two considerations:

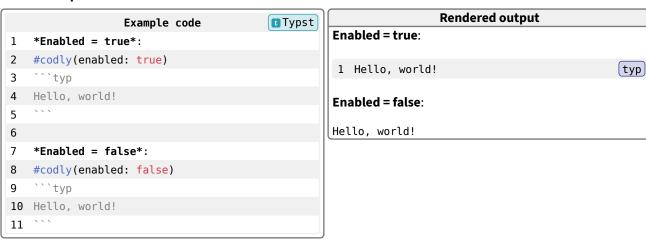
- The codly function uses states to store the configuration, this means that it is dependent on layout order for the order in which settings are applied.
- The codly function is not local, it sets the configuration for all code blocks that follow it in layout order, unless overriden by another codly call. This means that you cannot use it to set the configuration for a specific code block. To perform this, use the Local function to set the configuration for a specific "section".

2.1. Enabled (enabled)

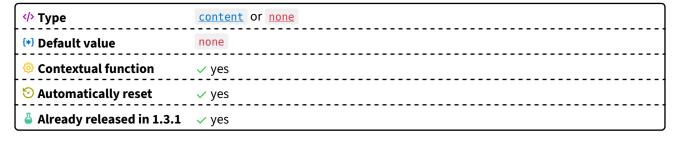


Whether codly is enabled or not. If it is disabled, the code block will be displayed as a normal code block, without any additional codly-specific formatting. This is useful if you want to disable codly for a specific block. You can also disable codly locally using the no-codly function, or disable it and enable it again using the codly-disable and codly-enable functions.

2.1.1. Example



2.2. Header (header)



An optional header to display above the code block. It can be optionally repeated on all subsequent pages with the header-repeat argument. And additional customizations are available with the header-transform arguments.

2.2.1. Example

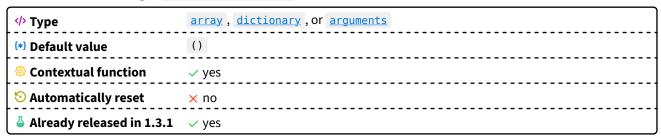


2.3. Header Repeat (header-repeat)

√ Type	bool
(*) Default value	false
© Contextual function	✓ yes
5 Automatically reset	× no
Already released in 1.3.1	✓ yes

Whether to repeat the header on each page. This is only applicable if a header is provided, if the code block is breaksble, and if it actually breaks on more than one page. For more information see grid.header:repeat.

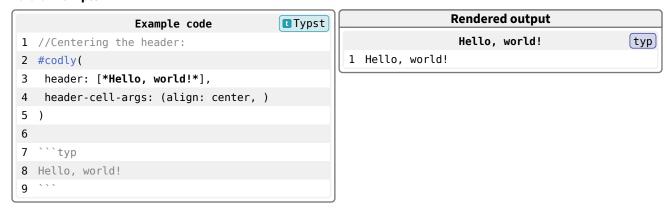
2.4. Header Cell Args (header-cell-args)



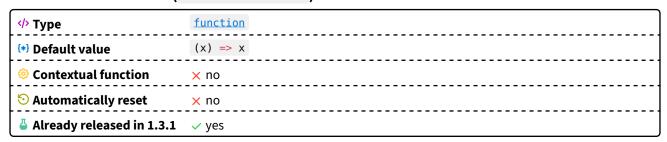
Additional arguments to be provided to the <code>grid.cell</code> containing the header. Lets you customize the header cell further. Internally, codly wraps the content of the <code>header</code> argument in a <code>grid.cell</code> with these arguments. The only argument that is always common is the body argument which is the value of the <code>header</code> argument, and the <code>colspan</code> which is always set to <code>2</code>.

For a full description of the argument, look at the documentation of the grid.cell function.

2.4.1. Example

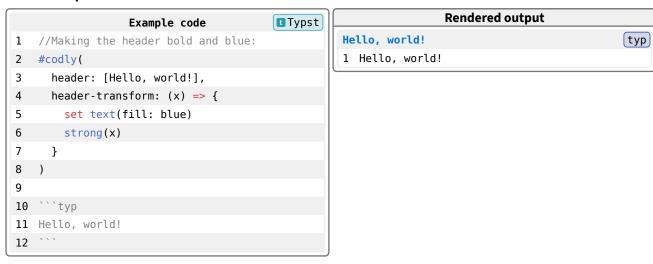


2.5. Header Transform (header-transform)

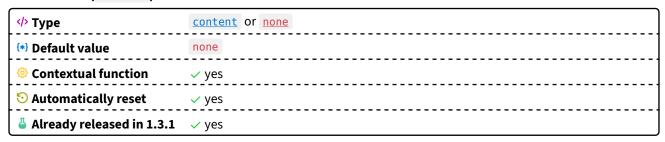


Function that transforms the header into arbitrary content to be stored in the <code>grid.cell</code>. Can be seen as a show-rule for the header. This allows to perform global transformation/show-rule like operations on the header.

2.5.1. Example



2.6. Footer (footer)

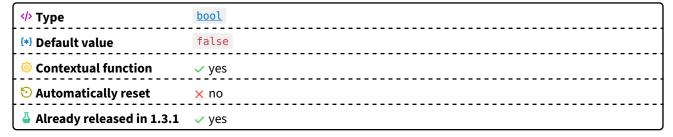


An optional footer to display below the code block. See <u>header</u> for more information.

2.6.1. Example

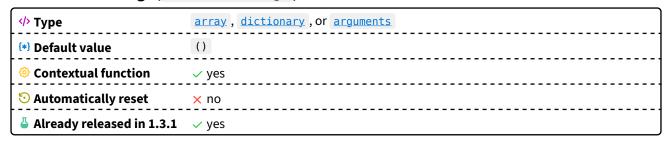


2.7. Footer Repeat (footer-repeat)



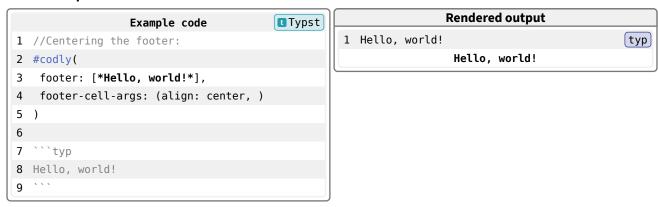
Whether to repeat the footer on each page. See header-repeat for more information.

2.8. Footer Cell Args (footer-cell-args)

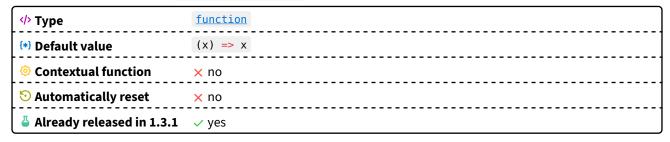


Additional arguments to be provided to the grid.cell containing the footer. See header-cell-args for more information.

2.8.1. Example

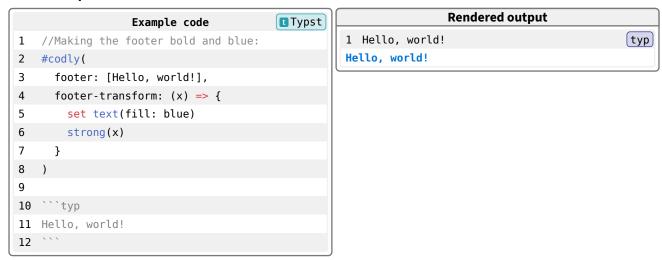


2.9. Footer Transform (footer-transform)

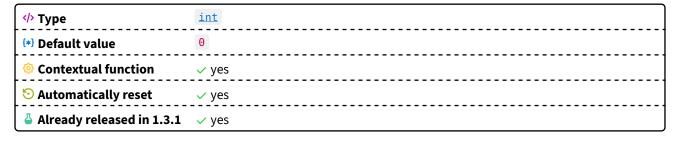


Function that transforms the footer into arbitrary content to be stored in the <code>grid.cell</code> . Can be seen as a show-rule for the footer. See header-transform for more information.

2.9.1. Example



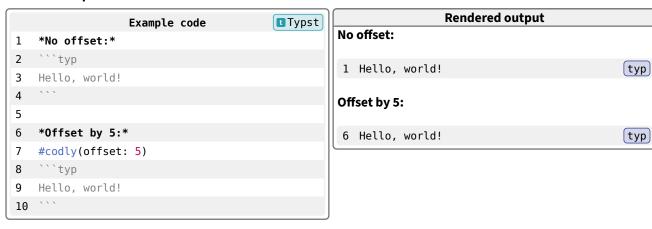
2.10. Offset (offset)



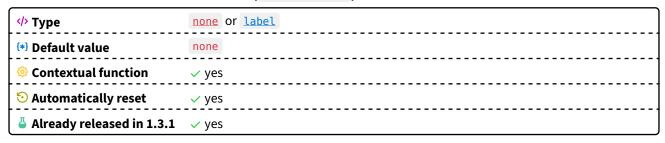
The offset to apply to line numbers.

This is purely cosmetic, only impacting the shown line numbers in the final output.

2.10.1. Example



2.11. Offset from other code block (offset-from)



The offset to apply to line numbers, relative to another code block. This is useful when you want to match line numbers between two code blocks. This code block will continue the line numbers from the other code block, with the specified offset.

This is done by giving a label to the parent raw block, and then setting it as the offset-from on the second code block.

i Info

Note that the offset obtained from the other code block is added to the offset specified in the offset argument.

Warning

Important: this feature works with any offset set on the other code block, including offset-from but may give unexpected results if both code blocks have offset-from set to each other or if the preceding code block has range or skips set.

Experiment

This feature should be considered experimental. Please report any issues you encounter on GitHub: https://github.com/Dherse/codly.

2.11.1. Example

```
Example code

1 ```py

2 def fib(n):

3  if n <= 1:

4  return n

5  return fib(n - 1) + fib(n - 2)

6 ``` <fib-fn>

7

8 *Will continue at line 5*

9 #codly(offset-from: <fib-fn>)

10 ```py

11 fib(25)

12 ```
```

```
Rendered output

1 def fib(n):
2    if n <= 1:
3        return n
4    return fib(n - 1) + fib(n - 2)

Will continue at line 5

5 fib(25)

py
```

2.12. Range (range)

```
✓> Type
none

(*) Default value
none

⑤ Contextual function
✓ yes

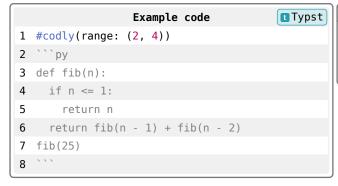
⑤ Automatically reset
✓ yes

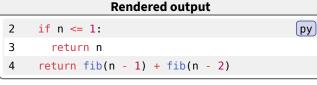
♣ Already released in 1.3.1
✓ yes
```

The range of line numbers to display, one-indexed. If set to none, all lines are displayed. Can also be achieved using the convenience function codly-range. If set to none, all lines are displayed.

i Info Line numbers are one-indexed, meaning that to reference the fourth line, you use the number 4.

2.12.1. Example





2.13. Ranges (ranges)

```
✓ Type
none

(*) Default value
none

⑤ Contextual function
✓ yes

⑤ Automatically reset
✓ yes

☑ Already released in 1.3.1
✓ yes
```

The ranges of line numbers to display, one-indexed. If set to none, all lines are displayed. Can also be achieved using the convenience function codly-range if provided with more than one range. If set to none, all lines are displayed. Note that it override the range argument.

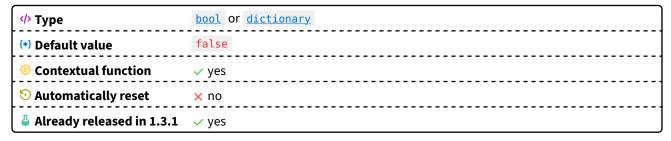
Info

Line numbers are one-indexed, meaning that to reference the fourth line, you use the number 4.

2.13.1. Example

```
Rendered output
                                       Typst
                 Example code
1 #codly(ranges: ((2, 2), (4, 4)))
                                                 2
                                                      if n <= 1:
                                                                                             ру
                                                 4
2 ```py
                                                      return fib(n - 1) + fib(n - 2)
3 def fib(n):
4 if n \le 1:
     return n
6 return fib(n - 1) + fib(n - 2)
7 fib(25)
8 ```
```

2.14. Smart skip (smart-skip)



Whether to automatically insert a skips entry between ranges of displayed values. They must be discontinuous for a skip to be added. The skip will be the size of the discontinuity. It can also be a dictionary with the keys:

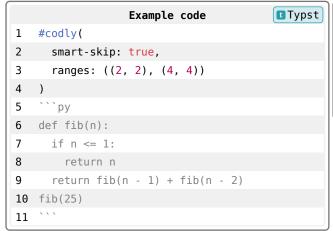
- first: whether to include a skip if the start of the block is outside of the ranges
- last: whether to include a skip if the end of the code block is outside of the ranges
- rest: whether to include a skip for unspecified values and/or in the middle of the code block.

You can specify one or more of these keys, if the rest is not specified, it defaults to none.

Experiment

This feature should be considered experimental. Please report any issues you encounter on GitHub: https://github.com/Dherse/codly.

2.14.1. Example



2.14.2. Example: Using a dictionary

```
Example code
                                     Typst
1 #codly(
2 smart-skip: (
       first: false,
3
     last: false,
5
       rest: true
7
     ranges: ((2, 2), (4, 4))
8 )
9 ```py
10 def fib(n):
   if n <= 1:
11
12 return n
13
   return fib(n - 1) + fib(n - 2)
14 fib(25)
15 ```
```

2.15. Languages (languages)

```
✓> Type
dictionary

(*) Default value
(:)

⑤ Contextual function
✓ yes

⑤ Automatically reset
× no

♣ Already released in 1.3.1
✓ yes
```

The language definitions to use for language block formatting. It is defined as a dictionary where the keys are the language names and each value is another dictionary containing the following keys:

- name: the "pretty" name of the language as a content/showable value
- color: the color of the language, if omitted uses the default color
- icon: the icon of the language, if omitted no icon is shown.

If an entry is missing, and language blocks are enabled, will show the "un-prettified" language name, with the default color.

2.15.1. Example

```
Example code
                                       ■ Typst
   #codly(
1
2
     languages: (
       py: (
3
         name: [Python], color: green, icon:
4
5
       ),
6
   )
7 )
8 ```py
9 def fib(n):
10 if n <= 1:
       return n
11
12
   return fib(n - 1) + fib(n - 2)
13 fib(25)
14
```

```
Rendered output

1 def fib(n):
2   if n <= 1:
3    return n
4   return fib(n - 1) + fib(n - 2)
5   fib(25)</pre>
```

2.15.2. Pre-existing language definitions

i Info

Check out the <u>codly-languages</u> package on Typst universe. It contains pre-definition for many language and is extremely easy to use. You can consider it officially endorsed by the codly author as of the 19th of November 2024.

```
Typst
                Example code
1 #import "@preview/codly-languages:0.1.7": *
   #codly(languages: codly-languages)
3 ```rust
4 fn main() {
5 println!("Hello, world!");
6 }
7
8 ```zig
9 const std = @import("std");
10
11 pub fn main() void {
12 std.debug.print("Hello, world!", .{});
13 }
14
```

```
Rendered output

1 fn main() {
2  println!("Hello, world!");
3 }

1 const std = @import("std");
2
3 pub fn main() void {
4  std.debug.print("Hello, world!", .{});
5 }
```

2.16. Default language color (default-color)

```
✓ Type color, gradient, or tiling
(*) Default value rgb("#283593")
⑤ Contextual function yes
⑤ Automatically reset x no
⑤ Already released in 1.3.1 yes
```

The default color to use for language blocks. Used when a language is not defined in the <u>languages</u> argument. Also note that it is only used when the <u>lang-format</u> is its <u>auto</u> or you are using it in a custom formatter. If you are using a custom formatter, it is passed to the formatter as a named argument color.

2.16.1. Example



```
Rendered output

Default color:

1 print('Hello, world!')
2 print('Hello, world!')

Overriden default color:

1 print('Hello, world!')
2 print('Hello, world!')
```

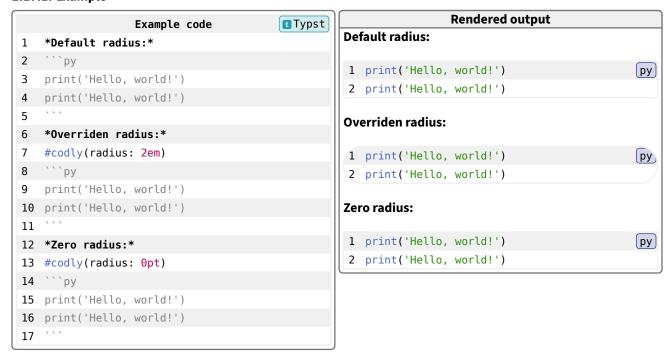
2.17. Radius (radius)

```
⟨→ Type length
(*) Default value 0.32em
© Contextual function ✓ yes
```

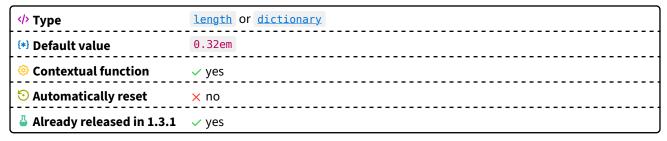
```
S Automatically reset × no
Already released in 1.3.1 ✓ yes
```

The radius of the border of the code block, see <u>block.radius</u> for more information.

2.17.1. Example

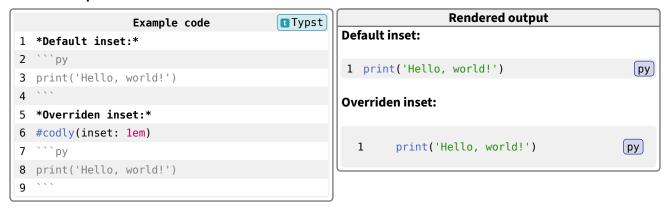


2.18. Inset (inset)

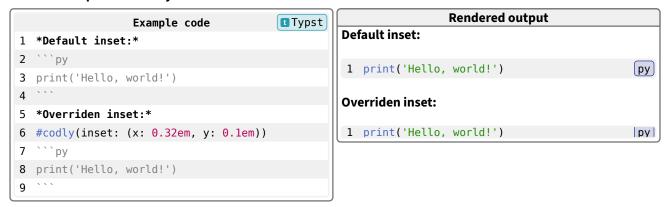


Inset of the code lines, that is the distance between the border and the code lines. It can also be a dictionary with the keys same keys as in the Tyspt built-in block.inset.

2.18.1. Example



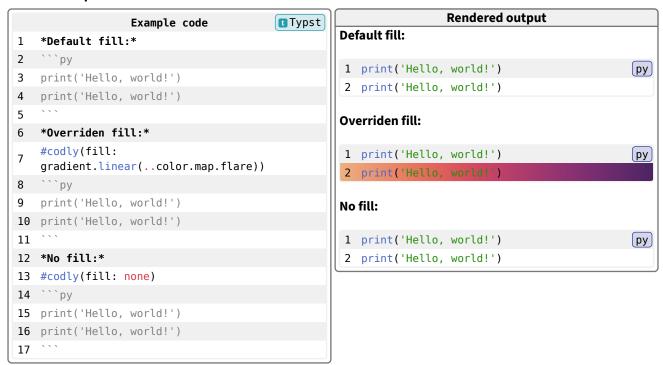
2.18.2. Example: Dictionary inset



2.19. Fill (fill)

The fill of the code block when not zebra-striped.

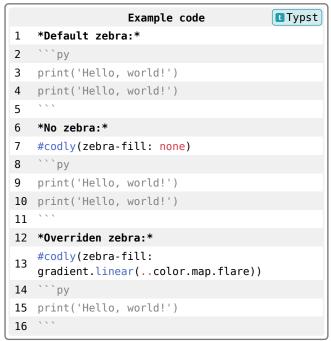
2.19.1. Example

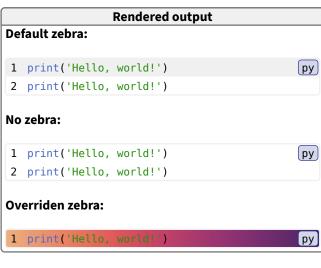


2.20. Zebra fill (zebra-fill)

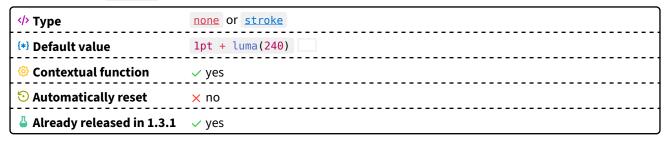
Background color of the code lines when zebra-stripped. If set to none, no zebra-striping is applied.

2.20.1. Example



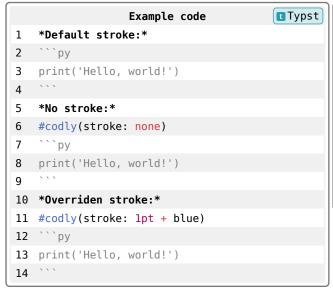


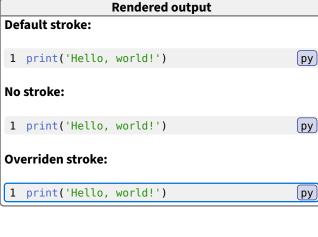
2.21. Stroke (stroke)



The stroke to surround the whole code block with?

2.21.1. Example





2.22. Language box inset (lang-inset)

```
      ✓> Type
      Length or dictionary

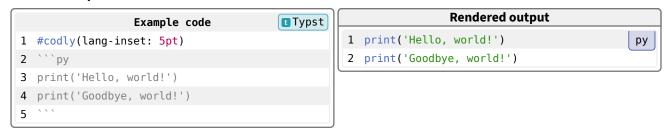
      (*) Default value
      0.32em

      ⑤ Contextual function
      ✓ yes

      ⑤ Automatically reset
      × no
```

The inset of the language block. This only applies if you're using the default language block formatter. It can also be a dictionary with the keys same keys as in the Tyspt built-in block.inset

2.22.1. Example

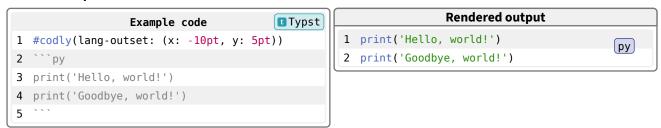


2.23. Language box outset (lang-outset)

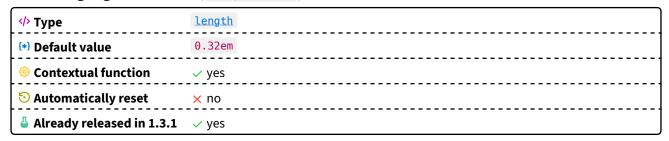
```
✓ Type dictionary
(*) Default value (x: 0.32em, y: 0em)
⑤ Contextual function  yes
⑤ Automatically reset  x no
♣ Already released in 1.3.1 yes
```

The X and Y outset of the language block, applied as a dx and dy during the place operation. This applies in every case, whether or not you're using the default language block formatter. The default value is chosen to get rid of the inset applied to each line.

2.23.1. Example

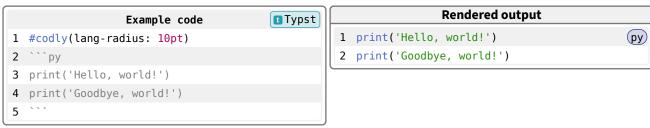


2.24. Language box radius (lang-radius)



The radius of the border of the language block.

2.24.1. Example



2.25. Language box stroke (lang-stroke)

```
(*) Default value (lang) => lang.color + 0.5pt

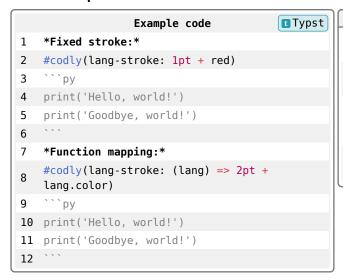
⑤ Contextual function × no

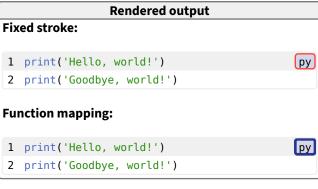
⑤ Automatically reset × no

집 Already released in 1.3.1 ∨ yes
```

The stroke of the language block. Can be a function that takes in the language dictionary or none (see argument languages) and returns a stroke.

2.25.1. Example

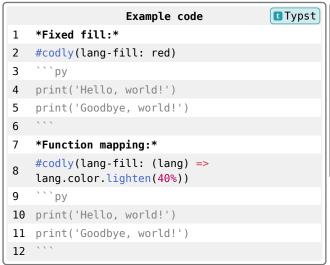


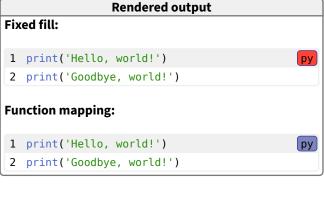


2.26. Language box fill (lang-fill)

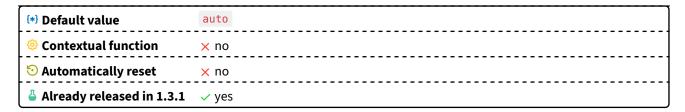
The background color of the language block. Can be a function that takes in the language dictionary or none (see argument <u>languages</u>) and returns a stroke.

2.26.1. Example





2.27. Language box formatter (lang-format)



The formatter for the language block. A value of none will not display the language block. To use the default formatter, set to auto. The function takes three arguments:

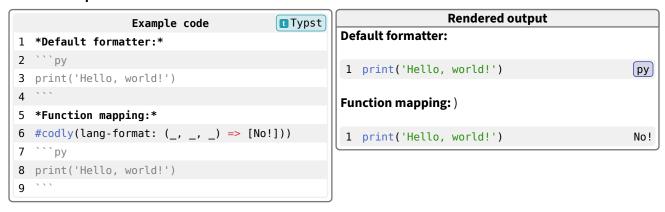
- lang: the language key (e.g. py)
- icon: the language icon, can be none or empty content
- color: the language color

The function should return a content/showable value.

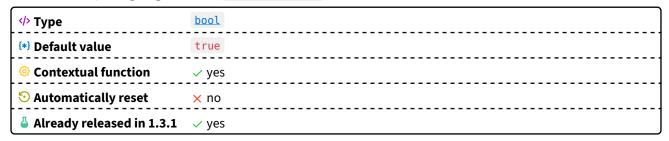
i Info

The language formatter should avoid using state as this can lead to quadratic execution time, see typst/typst #5220 for more information. Internally, when set to auto, codly uses an inlined function to avoid using states.

2.27.1. Example



2.28. Display language name (display-name)



Whether to display the name of the language in the language block. This only applies if you're using the default language block formatter.

2.28.1. Example

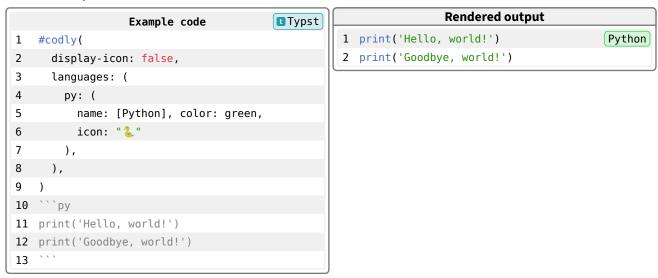
```
Rendered output
                                       Typst
                 Example code
                                                 1 print('Hello, world!')
1 #codly(
                                                 2 print('Goodbye, world!')
2
     display-name: false,
3
     languages: (
       py: (
5
         name: [Python], color: green,
         icon: "%"
7
       ),
8
   ),
9 )
10 ```py
11 print('Hello, world!')
12 print('Goodbye, world!')
13
```

2.29. Display language icon (display-icon)

```
✓ Type bool
(*) Default value true
⑤ Contextual function  yes
⑤ Automatically reset  x no
⑤ Already released in 1.3.1  yes
```

Whether to display the icon of the language in the language block. This only applies if you're using the default language block formatter.

2.29.1. Example



2.30. Line number format (number-format)

The format of the line numbers, a function that takes in number and returns a content. If set to none, disables line numbers.

2.30.1. Example

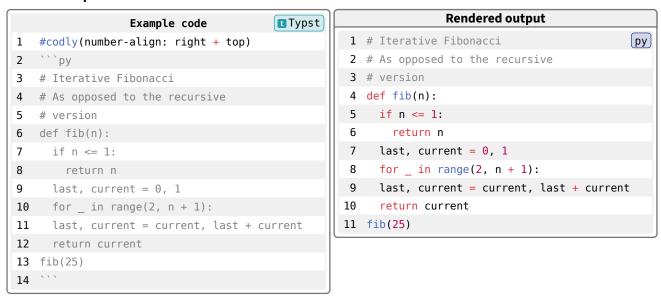
```
Example code

1 #codly(number-format: numbering.with("I."))
2 ```py
3 print('Hello, world!')
4 print('Goodbye, world!')
5 ```
```

2.31. Line number alignment (number-align)

The alignment of the numbers.

2.31.1. Example



2.32. Line number placement (number-placement)

```
✓ Type str
(*) Default value "inside"
⑤ Contextual function × no
⑤ Automatically reset × no
⑤ Already released in 1.3.1 ✓ yes
```

Sets the placement of the numbers, either "inside" (the default) or "outside" of the code block.

2.32.1. Example

```
Example code

1 *Line number outside of the block:*

2 #codly(number-placement: "outside")

3 ```py

4 print('Hello, world!')

5 print('Goodbye, world!')

6 ```

7 *Line number inside of the block:*

8 #codly(number-placement: "inside")

9 ```py

10 print('Hello, world!')

11 print('Goodbye, world!')

12 ```
```

```
Rendered output

Line number outside of the block:

1  print('Hello, world!')
2  print('Goodbye, world!')

Line number inside of the block:

1  print('Hello, world!')
2  print('Goodbye, world!')
```

2.33. Smart indentation (smart-indent)

```
✓ Type bool
(*) Default value true
② Contextual function × no
⑤ Automatically reset × no
⑤ Already released in 1.3.1 ✓ yes
```

Whether to use smart indentation, which will check for indentation on a line and use a bigger left side inset instead of spaces. This allows for linebreaks to continue at the same level of indentation. This is on by default, but disabling it can improve performance.

2.33.1. Example

```
Typst
                 Example code
1 *Enabled (default):*
2 ```py
3 def quicksort(L):
    qsort = lambda L: [] if L==[] else
  qsort([x for x in L[1:] if x < L[0]]) +
   L[0:1] + qsort([x for x in L[1:] if
  x>=L[0])
5
    qsort(L)
7 *Disabled:*
8 #codly(smart-indent: false)
9 ```py
10 def quicksort(L):
    qsort = lambda L: [] if L==[] else
11 qsort([x for x in L[1:] if x< L[0]]) +
   L[0:1] + qsort([x for x in L[1:] if
   x>=L[0])
12 qsort(L))
13 ```
```

```
Rendered output
Enabled (default):
1 def quicksort(L):
                                              (py)
     qsort = lambda L: [] if L==[] else
     qsort([x for x in L[1:] if x < L[0]]) +
     L[0:1] + qsort([x for x in L[1:] if
     x>=L[0])
3 qsort(L)
Disabled:
1 def quicksort(L):
                                              (py)
     qsort = lambda L: [] if L==[] else
gsort([x for x in L[1:] if x< L[0]]) +</pre>
   L[0:1] + qsort([x for x in L[1:] if
   x>=L[0]
3 qsort(L))
```

2.34. Skip last line if empty (skip-last-empty)

```
    ✓> Type
    bool

    (*) Default value
    true

    ⑤ Contextual function
    ✓ yes

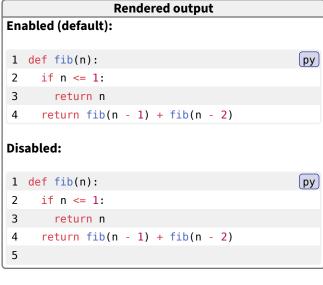
    ⑤ Automatically reset
    × no
```

```
♣ Already released in 1.3.1  yes
```

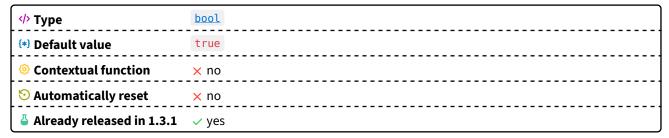
Whether to automatically skip the last line of the code block if it is empty. This avoids having an unnecessary empty line at the end of the code block.

2.34.1. Example

```
Example code
                                      Typst
   *Enabled (default):*
2 ```py
3 def fib(n):
4 if n <= 1:
5
       return n
     return fib(n - 1) + fib(n - 2)
6
7
8
   *Disabled:*
10 #codly(skip-last-empty: false)
11 ```py
12 def fib(n):
    if n <= 1:
13
14
   return n
    return fib(n - 1) + fib(n - 2)
15
16
17
```

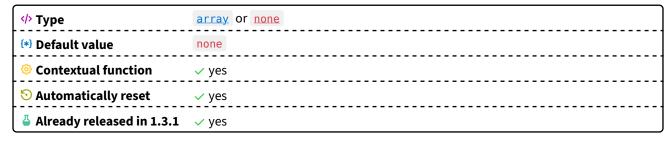


2.35. Breakable (breakable)



Whether the codeblocks are breakable across page and column breaks.

2.36. Skips (skips)



Insert a skip at the specified line numbers, setting its offset to the length of the skip. The skip is formatted using the skip-number argument. Each skip is an array with two values: the line where the skip is inserted (zero indexed) and the number of lines of the skip. The same behavior can be achieved using the codly-skip function.

2.36.1. Example

```
Example code

1 #codly(skips: ((4, 32), ))

2 ```py

3 def fib(n):

4  if n <= 1:

5  return n

6  return fib(n - 1) + fib(n - 2)

7 fib(25)

8 ```
```

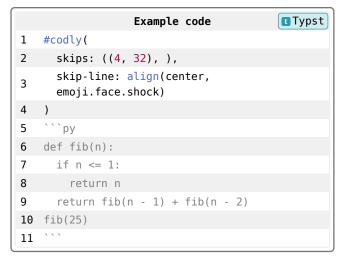
```
Rendered output

1  def fib(n):
2   if n <= 1:
3    return n
...
36   return fib(n - 1) + fib(n - 2)
37  fib(25)
```

2.37. Skip line (skip-line)

Sets the content with which the line code is filled when a skip is encountered.

2.37.1. Example



```
Rendered output

1  def fib(n):
2  if n <= 1:
3   return n
...
36  return fib(n - 1) + fib(n - 2)
37  fib(25)
```

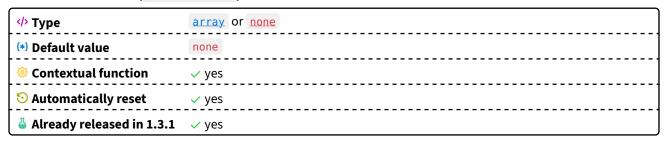
2.38. Skip number (skip-number)

Sets the content with which the line number columns is filled when a skip is encountered. If line numbers are disabled, this has no effect.

2.38.1. Example

```
Typst
                 Example code
1 #codly(
2
     skips: ((4, 32), ),
     skip-number: align(center,
3
     emoji.face.shock)
4
  ```py
5
6 def fib(n):
7
 if n <= 1:
 return n
9
 return fib(n - 1) + fib(n - 2)
10 fib(25)
11
```

## 2.39. Annotations (annotations)



The annotations to display on the code block. A list of annotations that are automatically numbered and displayed on the right side of the code block.

Each entry is a dictionary with the following keys:

- start: the line number to start the annotation
- end: the line number to end the annotation, if missing or none the annotation will only contain the start line
- content: the content of the annotation as a showable value, if missing or none the annotation will only contain the number
- ▶ label: if and only if the code block is in a figure, sets the label by which the annotation can be referenced.

Generally you probably want the content to be contained within a rotate (90deg).

Note: Annotations cannot overlap. Known issues:

- Annotations that spread over a page break will not work correctly
- Annotations on the first line of a code block will not work correctly.
- · Annotations that span lines that overflow (one line of code two lines of text) will not work correctly.

#### i Info

Line numbers are one-indexed, meaning that to reference the fourth line, you use the number 4.

## Experiment

This feature should be considered experimental. Please report any issues you encounter on GitHub: <a href="https://github.com/Dherse/codly">https://github.com/Dherse/codly</a>.

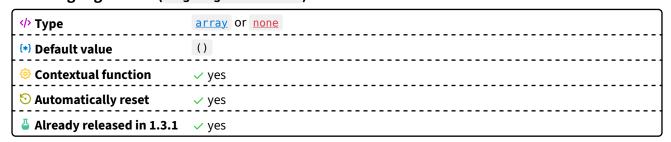
#### 2.39.1. Example

```
Typst
 Example code
 #codly(
1
2
 annotations: (
3
4
 start: 2, end: 5,
5
 content: block(
6
 width: 2em,
7
 rotate(-90deg, reflow: true,
 align(center)[Function body])
8
9
)
10
),
11
),
12)
   ```py
13
14 def fib(n):
15
    if n <= 1:
16
       return n
17
        return fib(n - 1) + fib(n - 2)
18
19 fib(25)
20
```

2.40. Annotation formatter (annotation-format)

The format of the annotation number. Can be none or a function that formats the annotation number.

2.41. Highlight lines (highlighted-lines)



Changes the background fill of certain lines to give them the appearance of being hihglighted. This is done by specifying an array of a combination of:

- line numbers: the line number (one-indexed) as it is shown in the document (including offsets, skips, etc.)
- array of line number & fill: an array of strictly two elements containing the line number (same as before) and a color, gradient, or pattern fill.

The default highlighting color is controlled by the highlighted-default-color argument.

i Info

Line numbers are one-indexed, meaning that to reference the fourth line, you use the number 4.

2.41.1. Example

```
Example code
                                    Typst
1 *Using default highlight color:*
2 #codly(
3 highlighted-lines: (1, 3)
4 )
5 ```py
6 def fib(n):
   if n <= 1:
8 return n
   else:
10
     return fib(n - 1) + fib(n - 2)
11 fib(25)
12 ```
13 *Per-line color selection:*
14 #codly(
   highlighted-lines: ((1,
    red.lighten(60%)), 2, 3)
16 )```py
17 def fib(n):
   if n <= 1:
18
19 return n
20 else:
21 return fib(n - 1) + fib(n - 2)
22 fib(25)
23 ```
```

```
Rendered output
Using default highlight color:
1 def fib(n):
                                               (py)
2 if n <= 1:
       return n
4 else:
5 return fib(n - 1) + fib(n - 2)
6 \text{ fib}(25)
Per-line color selection:
1 def fib(n):
                                               ру
2
   if n <= 1:
       return n
4 else:
       return fib(n - 1) + fib(n - 2)
6 \text{ fib}(25)
```

2.42. Highlight lines default color (highlighted-default-color)

Changes the default color of highlighted lines.

2.42.1. Example

```
■ Typst
                Example code
1 *Using default highlight color:*
2 #codly(
     highlighted-default-color:
3
     green.lighten(60%),
4 highlighted-lines: (1, 3)
5 )
6 ```py
7 def fib(n):
8 if n \le 1:
   return n
10
     else:
   return fib(n - 1) + fib(n - 2)
11
12 fib(25)
13
14 *Per-line color selection still works:*
     highlighted-lines: ((1,
16
     red.lighten(60%)), 2, 3)
17 )```py
18 def fib(n):
19
   if n <= 1:
20 return n
21
   else:
return fib(n - 1) + fib(n - 2)
23 fib(25)
24
```

```
Rendered output
Using default highlight color:
1 def fib(n):
                                               ру
    if n <= 1:
   return n
3
4
   else:
       return fib(n - 1) + fib(n - 2)
6 fib(25)
Per-line color selection still works:
1 def fib(n):
                                               [py]
2
    if n <= 1:
3
    return n
     else:
5
       return fib(n - 1) + fib(n - 2)
6 fib(25)
```

2.43. Highlights (highlights)

You can apply highlights to the code block using the highlights argument. It consists of a list of dictionaries, each with the following keys:

- line: the line number to start highlighting
- start: the character position to start highlighting, zero if omitted or none
- end: the character position to end highlighting, the end of the line if omitted or none
- fill: the fill of the highlight, defaults to the default color
- tag: an optional tag to be displayed alongside the highlight.
- inset : overrides the global highlight-inset.
- baseline : overrides the baseline which is set by default to the bottom component of highlight-inset.
- clip: overrides the global highlight-clip.
- outset: overrides the global highlight-outset.
- radius : overrides the global highlight-radius.
- label: if and only if the code block is in a figure, sets the label by which the highlight can be referenced.

As with other code block settings, annotations are reset after each code block.

i Info

Line numbers are one-indexed, meaning that to reference the fourth line, you use the number 4.

Info

Character positions are zero-indexed, meaning that to reference the fourth character, you use the number 3.

2.43.1. Example

```
Typst
                 Example code
1 #codly(highlights: (
     (line: 4, start: 2, end: none, fill:
2
     red),
     (line: 5, start: 13, end: 19, fill:
3
     green, tag: "(a)"),
     (line: 5, start: 26, fill: blue, tag:
5 ))
6 ```py
7 def fib(n):
8 if n <= 1:
9 return n
10 else:
11 return fib(n - 1) + fib(n - 2)
12 print(fib(25))
13 ```
```

```
Rendered output

1 def fib(n):
2   if n <= 1:
3    return n

4   else:
5   return fib(n - 1)(a) + fib(n - 2)(b)
6   print(fib(25))</pre>
```

2.43.2. Example: Local override of highlight parameters

You can override the global highlight parameters for a specific highlight by using the `inset`, `radius`, `clip`, and `outset` keys in the highlight dictionary.

```
Example code
                                      Typst
1 #codly(
2 highlights: (
       (line: 4, start: 2, end: none, fill:
3
       (line: 5, start: 13, end: 19, fill:
4
       green, tag: "(a)", inset: 0.1em,
       radius: 0.5em),
       (line: 5, start: 26, fill: blue, tag:
5
       "(b)", clip: false)
6 )
7)
8 ```py
9 def fib(n):
10 if n \le 1:
11 return n
12 else:
13
   return fib(n - 1) + fib(n - 2)
14 print(fib(25))
15
```

```
Rendered output

1 def fib(n):
2   if n <= 1:
3    return n

4   else:
5   return fib(n - 1)(a) + fib(n - 2)(b)
6   print(fib(25))</pre>
```

2.44. Highlight radius (highlight-radius)

```
✓ Type length
(*) Default value 0.32em
⑤ Contextual function  yes
⑤ Automatically reset  yes
⑤ Already released in 1.3.1  yes
```

The radius of the highlights.

2.45. Highlight fill (highlight-fill)

✓ Type	function
(*) Default value	(color) => color.lighten(80%)
© Contextual function	× no
5 Automatically reset	× no
Already released in 1.3.1	✓ yes

The fill transformer of the highlights, is a function that takes in the highlight color and returns a fill.

2.46. Highlight stroke (highlight-stroke)

The stroke transformer of the highlights, is a function that takes in the highlight color and returns a stroke.

2.47. Highlight inset (highlight-inset)

√ Type	length or dictionary
(*) Default value	0.32em
© Contextual function	✓ yes
5 Automatically reset	× no
Already released in 1.3.1	✓ yes

The inset of the highlight blocks. It can also be a dictionary with the keys same keys as in the Tyspt built-in block.inset.

2.47.1. Example: Alignment of lines

If alignment between highlighted and non-highlighted lines is critical for your use case, as could be the case in presentation, you can set the horizontal inset to be zero, or close to zero, to maintain alignment between the lines.

```
Example code

1 #codly(
2 highlight-inset: (x: 0pt, y: 0.32em),
3 highlights: ((line: 2, start: 0), )
4 )
5 ```kotlin
6 private val n = 0
7 private val n = 0
8 ```
```

2.48. Highlight outset (highlight-outset)

```
✓> Type
length or dictionary

(*) Default value
0em

⑤ Contextual function
✓ yes

⑤ Automatically reset
× no
```

The outset of the highlight blocks. It can also be a dictionary with the keys same keys as in the Tyspt built-in block.outset.

2.49. Highlight clip (highlight-clip)

⇔ Type	bool
(*) Default value	true
© Contextual function	✓ yes
5 Automatically reset	× no
Already released in 1.3.1	✓ yes

Whether highlight box clips code. See the documentation of the Tyspt built-in block.clip.

2.50. Reference by (reference-by)

√ Type	str
(*) Default value	"line"
© Contextual function	✓ yes
5 Automatically reset	× no
Already released in 1.3.1	✓ yes

The mode by which references are displayed. Two modes are available:

- line: references are displayed as line numbers
- item: references are displayed as items, i.e by the tag for highlights and content for annotations.

2.51. Reference separator (reference-sep)

√ Type	str or content
(*) Default value	п_п
© Contextual function	✓ yes
5 Automatically reset	× no
Already released in 1.3.1	✓ yes

The separator to use when referencing highlights and annotations.

2.52. Reference number format (reference-number-format)

√ Type	function
(*) Default value	numbering.with("1")
© Contextual function	× no
5 Automatically reset	× no
△ Already released in 1.3.1	✓ yes

The format of the reference number line number, only used if reference-by is set to "line".

2.53. Language name aliases (aliases)

⇔ Type	dictionary.
(*) Default value	(:)
© Contextual function	× no
5 Automatically reset	× no

Creates aliases between languages, allowing them to have separate languages but the same syntax highlighting. An example of that could be cuda and c++ having the same highlighting but different icons.

2.53.1. Example

```
Example code
1 #import "@preview/codly-languages:0.1.7": *
2
3 #codly(
4 languages: codly-languages,
5 aliases: ("cuda": "c++")
6 )
7 ```cuda
8 int main() {
9 std::cout << "Hello, World!" << std::endl;
10 return 0;
11 }```</pre>
```

```
Rendered output

1 int main() {
2 std::cout << "Hello, World!" << std::endl;
3 return 0;
4 }
```

3. Referencing code blocks, highlights, and annotations

This section of the documentation will detail how you can use codly to reference: lines, highlights, and annotations in your code blocks. To do this, here are the requirements that must be met **for each code block**:

- □ Numbering of figures must be turned on: set figure(numbering: ...).
- ☐ The code block must be contained within a raw figure: figure(kind: raw).
- ☐ The figure must have a label of its own: figure(...)[...] <my-label>.

3.1. Shorthand line references

You can reference lines directly, if you have set a label correctly, using the shorthand syntax <code>@my-label:2</code> to reference the second line (one-indexed) of the code block with the label <code><my-label></code>. It will always use the <code>reference-number-format</code> argument of the codly function to format the line number.

i Info

Line numbers are one-indexed, meaning that to reference the fourth line, you use the number 4.

Experiment

Due to upcoming changes to v0.13 of Typst, this entire feature has been re-done to be more robust, but that means that the styling might have changed slightly and it might have introduced new issues. If you find any issues, report them on https://github.com/Dherse/codly.

```
Example code

1 #figure(
2 caption: "A code block with a label"
3 )[
... ...
8 ] <my-label>
9 I can reference my code block: @my-label. Or a specific line: @my-label:2.
```

```
Rendered output

1 = Example typ
2 *Hello, world!*
Listing 1: A code block with a label
I can reference my code block: Listing 1. Or a specific line:
Listing 1-2.
```

3.2. Highlight references

You can also highlight by reference, to do this, you need to set a label for your highlight in the highlights argument of the codly function. You can then reference the highlight using the shorthand syntax @my-highlight to reference the highlight with the label my-highlights. There are two supported reference-by modes:

- "line": references the line of the highlight
- "item": references the tag of the highlight, this requires that the tag be set for each tagged highlight.

```
Example code

1 #codly(
highlights: ((line: 2, start: 2, end: 7, label: <hl-1>),

3 ))
...
...

13 Reference a highlight by its label: @hl-1.
```

```
Rendered output

1 = Example typ
2 *Hello, world!*

Listing 2: A code block with a label

Reference a highlight by its label: Listing 2-2.
```

And using <u>"item"</u> mode:

```
Example code

1 #codly(
highlights: ((line: 2, start: 2, end: 7, label: <hl-2>, tag: [ Highlight ]), ),
reference-by: "item",

4 )
...

14 Reference a highlight by its label: @hl-2.
```

```
Rendered output

1 = Example typ
2 * Hello, world! Highlight *

Listing 3: A code block with a label

Reference a highlight by its label: Listing 3-Highlight.
```

4. Getting nice icons

This is a short, non-exhaustive guide on how to get nicer icons for the languages of your code blocks. In the documentation, codly makes use of tabler-icons to display the language icons. But a more general approach is the following:

- 1. Chose a font that contains icons, such as:
 - Tabler Icons
 - Font Awesome
 - Material Icons
 - · Look on Google Fonts for more options
- 2. Download the font and put it in your project (if using the CLI, you need to set the -- font argument)
- 3. Using your font selector, select the icon you wish to use
 - For example, the language icon in Tabler Icons is ebbe (the unicode value of the icon, which you can find in the documentation of the font)
 - Use the <u>text</u> function to display the icon in your document by setting the font, size, and the unicode value of the icon:

```
1 text(font: "tabler-icons" Font name, size: lem, "\u{ebbe UTF-8 icon code}}")
```

4. You can store it the languages argument of the codly function to use it for all code blocks in your document:

```
#let icon = text(font: "tabler-icons",
size: lem, "\u{ebbe}")

#codly(languages: (text: (icon: icon, name:
    "Text")))
3 ```text
4 Hello, world!
5 ```

Rendered output

1 Hello, world!
```

- 5. Congrats, you now have fancy icons!
- 6. ..
- 7. But you can notice that the baseline of the icon is wrong, I find that this is generally the case with tabler, you can set the baseline to 0.1em in the icon to fix it:

```
Example code

| #let icon = text(font: "tabler-icons", size: lem, "\u{ebbe}", baseline: 0.lem)
| #codly(languages: (text: (icon: icon, name: "Text")))
| 3 ```text
| 4 Hello, world!
| 5 ```
```

4.1. Typst language icon (typst-icon)

Additionally, codly ships with language definitions for the Typst language. You can use the typst-icon function to get the Typst icon for your code blocks. This function takes no arguments and returns the proper settings for codly to use the Typst icon.

```
i Info
You can use the ... spread operator to spread it into your own languages dictionary.
```

```
Example code

1 #codly(languages: typst-icon)

2 ```typ

3 = Here's a title

4 Hello, world!

5 ```
```

5. Other functions

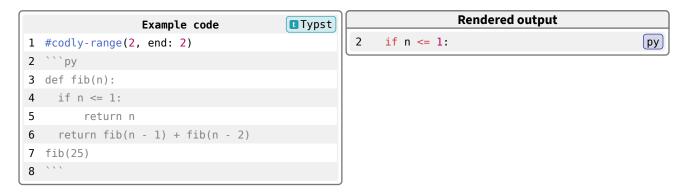
5.1. Skip (codly-skip)

Convenience function for setting the skips, see the skips argument of the codly function.

5.2. Range (codly-range)

Convenience function for setting the range, see the <u>range</u> argument of the <u>codly</u> function. If you provide more than one range, as a list of arguments, it will set the <u>ranges</u> argument instead.

With a single range:



With more than one range:

```
Rendered output
                 Example code
                                      Typst
                                                 2
                                                     if n <= 1:
                                                                                            ру
1 #codly-range(2, end: 2, (4, 5))
2 ```py
                                                 4
                                                     return fib(n - 1) + fib(n - 2)
3 def fib(n):
                                                 5 fib(25)
4 if n \le 1:
       return n
6 return fib(n - 1) + fib(n - 2)
7 fib(25)
8 ```
```

5.3. Offset (codly-offset)

Convenience function for setting the offset, see the offset argument of the codly function.

5.4. Local (local)

Codly provides a convenience function called <code>local</code> that allows you to locally override the global settings for a specific code block. This is useful when you want to apply a specific style to a code block without affecting the rest of the code blocks in your document. It works by overriding the default codly show rule locally with an override of the arguments by those you provide. It does not rely on states (much) and should no longer add layout passes to the rendering which could cause documents to not converge.

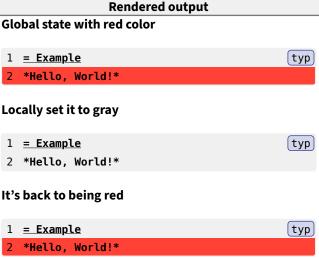
Warning

When using nested: false on your local states, the outermost local state will be overriden by the inner local state(s). This means that the inner local state will be the only one that is applied to the code block. And that any previous local states (in the same hierarchy) will be ignored for subsequent code blocks.

i Info

Once custom elements become available in Typst, and codly moves to using those and set rules, this limitation will be lifted and you will be able to use nested local states without performance impact.





5.4.1. Local state for per-language configuration

Additionally, local settings can be used to set per-language configuration using a show rule on your raw blocks. This can be done in one of two ways: by using a show rule on raw.where(block: true, lang: "<lang>") and calling the local function, or by using the codly function. The main differentiators are that the local function is faster and does not rely on states, while the codly function is more flexible, but slower and will also style all following blocks, you must therefore manually reset the changes.

Experiment

This should work in most cases, but this feature should be considered experimental. Please report any issues you encounter on GitHub: https://github.com/Dherse/codly.

Info

Note that you only want to do show rules on raw blocks where block: true, otherwise this will make your document slow.

Warning

If you use the local function in a show rule, nested local states will not work with the settings you have set! Use the codly method instead. If using the codly method, and you must manually reset the changed settings in the show rule!

```
Example code
                                       Typst
   #show raw.where(block: true, lang: "rust"):
1
   local.with(
   number-format: numbering.with("I")
2
3
  )
4
   #show raw.where(block: true, lang: "py"):
6
    codly(number-format: numbering.with("①"))
7
8
     codly(number-format: numbering.with("1"))
9 }
10
11 *Numbered with Roman numerals*
12 ```rust
13 fn main() {
   println!("Rust code has Roman numbers");
14
15 }
16
17
18 *Numbered with circled numbers*
19
20 print("Python code has circled numbers")
21
22
23 *Override with local state*
24 #local(
25
    fill: blue.lighten(80%),
26
27
     print("Python code is green")
28
29 )
```

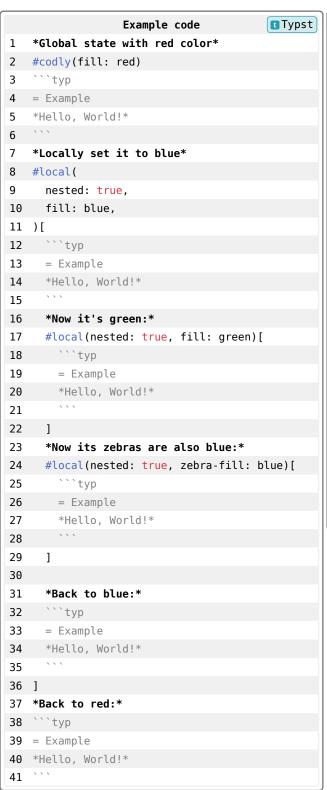
```
Rendered output
Numbered with Roman numerals
                                           rust
    fn main() {
       println!("Rust code has Roman numbers");
II
III }
Numbered with circled numbers
① print("Python code has circled numbers") [py]
Override with local state
1 print("Python code is green")
                                             ру
```

5.4.2. Nested local state

Codly does support nested local state, the innermost local state will override the outermost local state. This allows you to have different styles for different parts of your code block. This function takes the same arguments as the codly function, but only the arguments that are different from the global settings need to be provided.

Warning

Nested local states can slow down documents significantly if over-used (explicitly set nested: true). Use them sparingly and only when necessary. Another solution is to use the normal codly function before and after your code block. You can also use the the argument nested: false on local to prevent nested local states, which significantly reduces the performance impact.





5.5. No Codly (no-codly)

This is a convenience function equivalent to local (enabled: false, body).



```
Rendered output

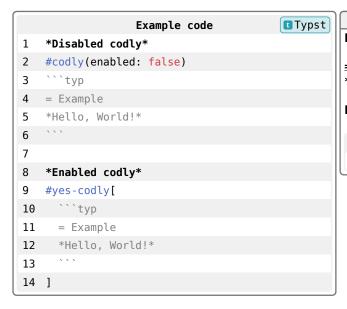
Enabled codly

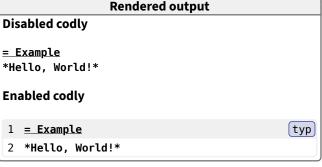
1 = Example
2 *Hello, World!*

Disabled codly
= Example
*Hello, World!*
```

5.6. Yes Codly (yes-codly)

This is a convenience function equivalent to local (enabled: false, body).

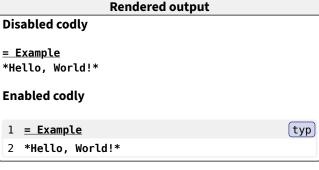




5.7. Enable (codly-enable)

Enables codly globally, equivalent to codly (enabled: true).

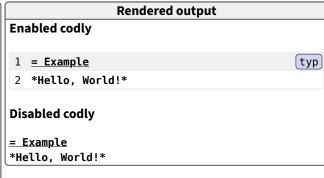




5.8. Disable (codly-disable)

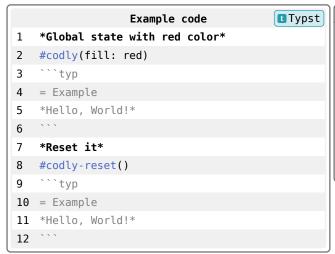
Disables codly globally, equivalent to codly (enabled: false).





5.9. Reset (codly-reset)

Resets all codly settings to their default values. This is useful when you want to reset the settings of a code block to the default values after applying local settings.





6. Codly performance