

Codly v1.1.0 manual

Your code blocks on steroids

O RLY?

Contents

| 1. Codly | 2 |
|---|----|
| 1.1. Initialization | |
| 1.2. Enabling and disabling codly | |
| 2. A primer on Codly's show-rule like system | |
| 2.1. Enabled (enabled) | |
| 2.2. Header (header) | |
| 2.3. Header Repeat (header-repeat) | |
| 2.4. Header Cell Args (header-cell-args) | |
| 2.5. Header Transform (header-transform) | |
| 2.6. Footer (footer) | |
| 2.7. Footer Repeat (footer-repeat) | |
| 2.8. Footer Cell Args (footer-cell-args) | |
| 2.9. Footer Transform (footer-transform) | |
| 2.10. Offset (offset) | |
| 2.11. Offset from other code block (offset-from) | |
| 2.12. Range (range) | |
| 2.13. Ranges (ranges) | 11 |
| 2.14. Languages (languages) | |
| 2.15. Default language color (default-color) | 13 |
| 2.16. Radius (radius) | 13 |
| 2.17. Inset (inset) | 14 |
| 2.18. Fill (fill) | |
| 2.19. Zebra fill (zebra-fill) | 16 |
| 2.20. Stroke (stroke) | |
| 2.21. Language box inset (lang-inset) | |
| 2.22. Language box outset (lang-outset) | 17 |
| 2.23. Language box radius (lang-radius) | |
| 2.24. Language box stroke (lang-stroke) | |
| 2.25. Language box fill (lang-fill) | |
| 2.26. Language box formatter (lang-format) | |
| 2.27. Display language name (display-name) | |
| 2.28. Display language icon (display-icon) | |
| 2.29. Line number format (number - format) | |
| 2.30. Line number alignment (number-align) | |
| 2.31. Smart indentation (smart-indent) | |
| 2.32. Breakable (breakable) | |
| 2.33. Skips (skips) | |
| 2.34. Skip line (skip-line) | |
| 2.35. Skip number (skip-number) | |
| 2.36. Annotations (annotations) | |
| 2.37. Annotation formatter (annotation-format) | |
| 2.38. Highlights (highlights) | |
| 2.39. Highlight radius (highlight - radius) | |
| 2.40. Highlight fill (highlight-fill) | |
| 2.41. Highlight stroke (highlight-stroke) | |
| 2.42. Highlight inset (highlight-inset) | |
| 2.43. Reference by (reference-by) | |
| 2.44. Reference separator (reference-sep) | |
| 3. Referencing code blocks, highlights, and annotations | |
| 3.1. Shorthand line references | |
| 3.1. Snorthand line references | |
| 4. Getting nice icons | |
| 4.1. Typst language icon (typst-icon) | |
| 5. Other functions | |
| 5.1. Skip (codly-skip) | |
| 5.2. Range (codly-range) | |
| 5.3. Offset (codly-range) | |
| | |

| 5.4. Local (local) | 32 |
|------------------------------|----|
| 5.5. No codly (no-codly) | |
| 5.6. Enable (codly-enable) | |
| 5.7. Disable (codly-disable) | 35 |
| 5.8. Reset (codly-reset) | 36 |
| 6. Codly 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

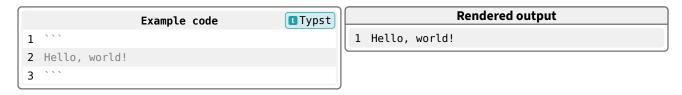
1 #import "@preview/codly:1.1.0": *

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 codly-disable function, the enabled argument of the codly-enable function or by setting the enabled parameter again.

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
      smart-indent: false,
25
     annotations: none,
26
     annotation-format: numbering.with("(1)"),
27
      highlights: none,
28
     highlight-radius: 0.32em,
29
      highlight-fill: (color) => color.lighten(80%),
30
      highlight-stroke: (color) => 0.5pt + color,
31
     highlight-inset: 0.32em,
      reference-by: line,
32
33
     reference-sep: "-",
34
      reference-number-format: numbering.with("1"),
35
     header: none,
36
      header-repeat: false,
37
     header-transform: (x) \Rightarrow x,
38
     header-cell-args: (),
39
     footer: none,
40
     footer-repeat: false,
41
    footer-transform: (x) \Rightarrow x,
42
     footer-cell-args: (),
43
     breakable: false,
44 ) = \{\}
```

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

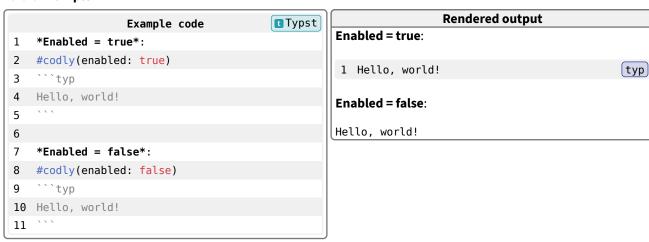
Unlike regular set-rules in native Typst, there are two considerations:

- 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 |
| ② Contextual function | ✓ yes |
| ூ Automatically reset | × no |

Whether to repeat the header on each page. This is only applicable if a header is provided, if the code block is breaksle, and if it actually breaks on more than one page. For more information see <a href="https://grantsleepide.com/grants

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

```
⟨→ Type array, dictionary, or arguments

(*) Default value ()

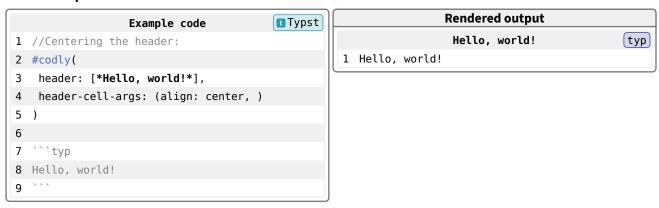
⑤ Contextual function ✓ yes

⑤ Automatically reset × no
```

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.

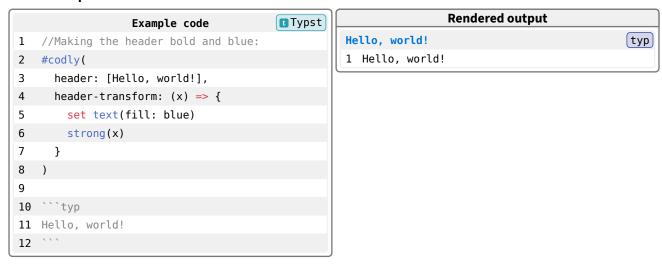
2.4.1. Example



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

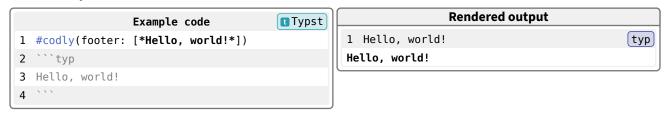


2.6. Footer (footer)

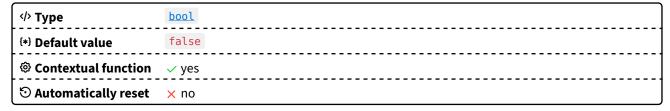
```
⟨→ Type content or none
(*) Default value none
② Contextual function ✓ yes
⊙ Automatically reset ✓ yes
```

An optional footer to display below the code block. See header for more information.

2.6.1. Example



2.7. Footer Repeat (footer-repeat)

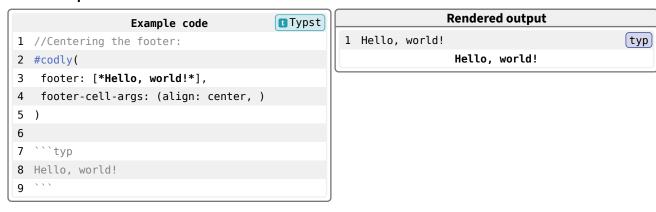


Whether to repeat the footer on each page. See header-repeat for more information.

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

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

2.8.1. Example

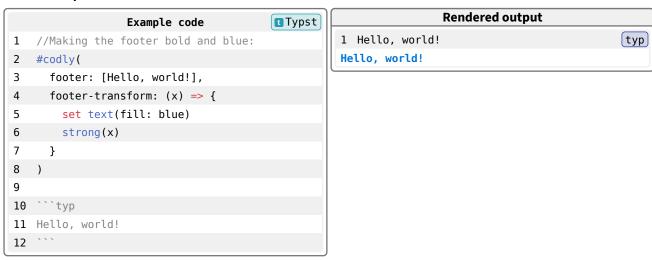


2.9. Footer Transform (footer-transform)

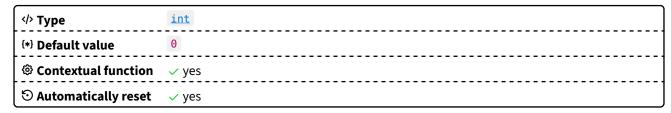
```
⟨→ Type function
(*) Default value (x) ⇒ x
⑤ Contextual function x no
⑤ Automatically reset x no
```

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 <code>header-transform</code> for more information.

2.9.1. Example



2.10. Offset (offset)

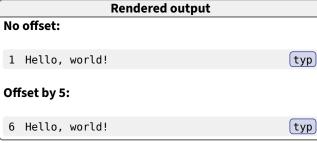


The offset to apply to line numbers.

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.

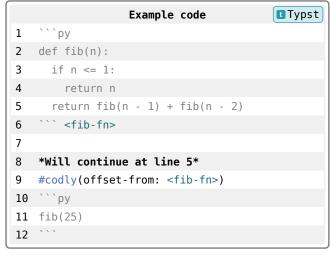
Warning

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



```
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

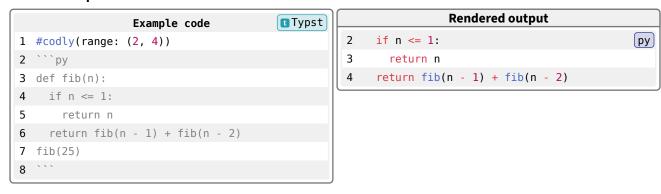
5 fib(25)

py
```

2.12. Range (range)

The range of line numbers to display, zero-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.

2.12.1. Example



2.13. Ranges (ranges)

The ranges of line numbers to display, zero-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.

```
i Info

Overrides the range argument.
```

2.13.1. Example

```
Example code

1 #codly(ranges: ((2, 2), (4, 4)))

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 '``
```

2.14. Languages (languages)

```
⟨→ Type dictionary
(*) Default value (:)
② Contextual function ✓ yes
⑤ Automatically reset × no
```

The language definitions to use for language block formatting. It is defined as a dictionary where the keys are the language names andeach 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.

2.14.1. Example

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

2.14.2. 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.

```
Example code
                                       Typst
1 #import "@preview/codly-languages:0.1.1": *
2 #codly(languages: codly-languages)
3 ```rust
4 fn main() {
5
    println!("Hello, world!");
6 }
7
8
   const std = @import("std");
10
11 pub fn main() void {
    std.debug.print("Hello, world!", .{});
12
13 }
14
```

```
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 }
```

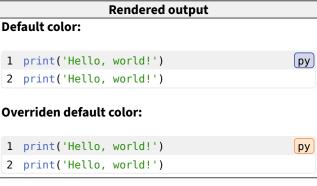
Rendered output

2.15. Default language color (default-color)

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 <u>color</u>.

2.15.1. Example

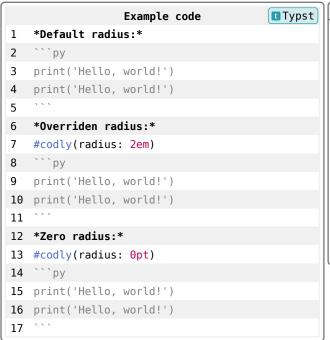


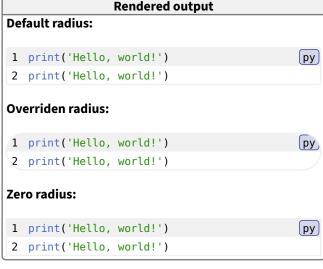


2.16. Radius (radius)

The radius of the border of the code block, see block, radius for more information.

2.16.1. Example

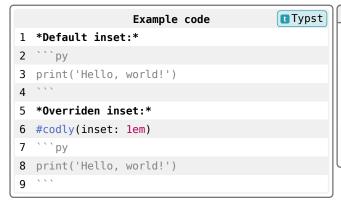


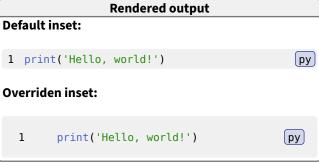


2.17. Inset (inset)

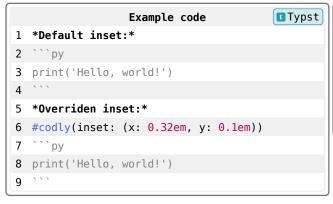
Inset of the code lines, that is the distance between the border and the code lines. It can also be a dictionary with the keys same keys as in the Tyspt built-in block.inset.

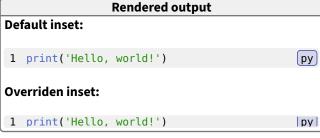
2.17.1. Example





2.17.2. Example: Dictionary inset





2.18. Fill (fill)

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

2.18.1. Example

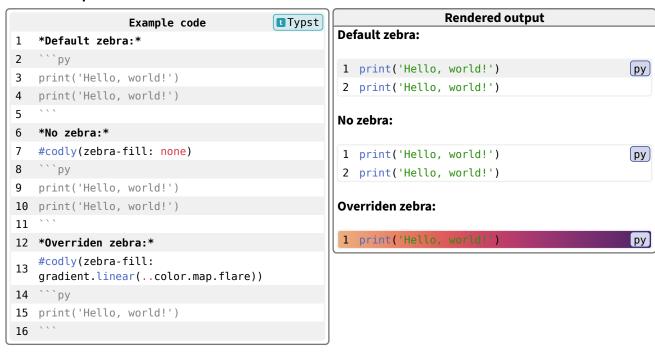




2.19. Zebra fill (zebra-fill)

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

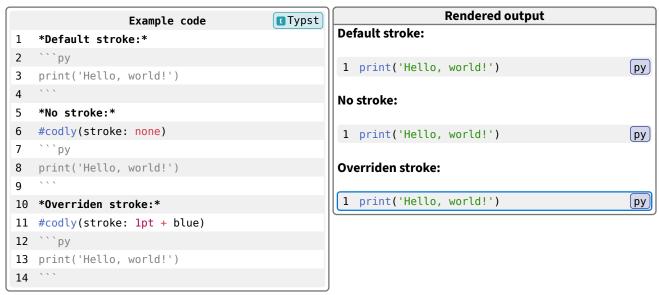
2.19.1. Example



2.20. Stroke (stroke)

The stroke to surround the whole code block with?

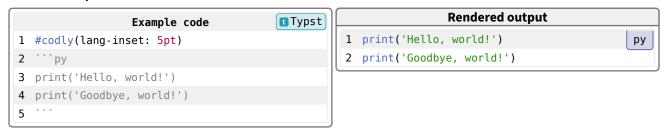
2.20.1. Example



2.21. Language box inset (lang-inset)

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

2.21.1. Example



2.22. Language box outset (lang-outset)

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.

2.22.1. Example

```
Example code

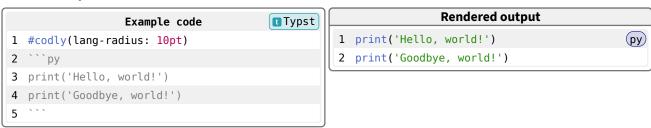
1 #codly(lang-outset: (x: -10pt, y: 5pt))
2 ```py
3 print('Hello, world!')
4 print('Goodbye, world!')
5 ```
```

2.23. Language box radius (lang-radius)

```
⟨→ Type length
(*) Default value 0.32em
② Contextual function ✓ yes
⊙ Automatically reset × no
```

The radius of the border of the language block.

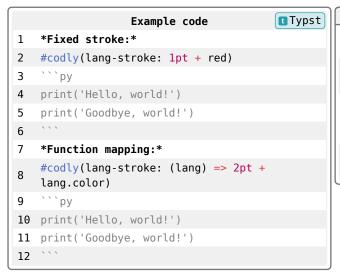
2.23.1. Example

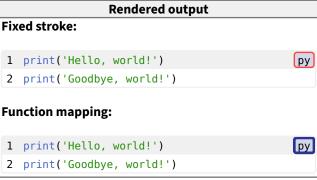


2.24. Language box stroke (lang-stroke)

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.

2.24.1. Example

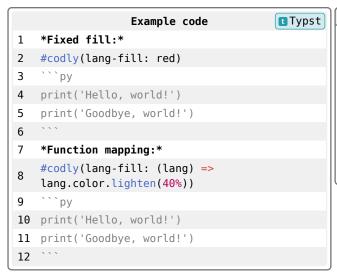




2.25. Language box fill (lang-fill)

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.

2.25.1. Example



```
Rendered output

Fixed fill:

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

Function mapping:

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

2.26. Language box formatter (lang-format)

```
⟨→ Type type(auto), none, or function

(*) Default value auto

⑤ Contextual function × no
⑤ Automatically reset × no
```

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:

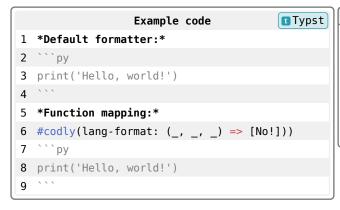
- lang: the language key (e.g. py)
- icon: the language icon, can be none or empty content
- color: the language color

The function should return a content/showable value.

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.

2.26.1. Example



```
Rendered output

Default formatter:

1 print('Hello, world!')

Function mapping:)

1 print('Hello, world!')

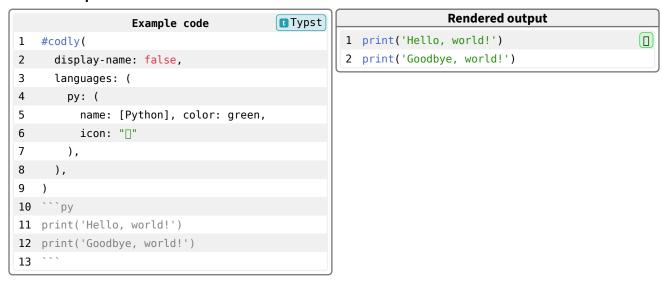
No!
```

2.27. Display language name (display-name)

```
⟨→ Type bool
(*) Default value true
② Contextual function ✓ yes
⑤ Automatically reset × no
```

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

2.27.1. Example



2.28. Display language icon (display-icon)

```
      ⟨→ Type
      bool

      (*) Default value
      true

      ⑤ Contextual function
      ✓ yes

      ⑤ Automatically reset
      × no
```

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

2.28.1. Example

```
Rendered output
                Example code
                                      Typst
                                                1 print('Hello, world!')
                                                                                       Python
1 #codly(
  display-icon: false,
                                                2 print('Goodbye, world!')
2
3
     languages: (
4
       py: (
5
         name: [Python], color: green,
6
        icon: "□"
7
      ),
8),
9)
10 ```py
11 print('Hello, world!')
12 print('Goodbye, world!')
13 ```
```

2.29. 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.

2.29.1. Example

```
Example code

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

2.30. Line number alignment (number-align)

The alignment of the numbers.

2.30.1. Example

```
Example code
                                     Typst
1 #codly(number-align: right + top)
2 ```py
3 # Iterative Fibonacci
4 # As opposed to the recursive
5 # version
6 def fib(n):
7 if n \le 1:
8 return n
9 last, current = 0, 1
10 for \underline{} in range(2, n + 1):
11 last, current = current, last + current
12 return current
13 fib(25)
14
```

```
1 # Iterative Fibonacci
2 # As opposed to the recursive
3 # version
4 def fib(n):
5   if n <= 1:
6    return n
7   last, current = 0, 1
8   for _ in range(2, n + 1):
9   last, current = current, last + current
10   return current
11 fib(25)</pre>
```

Rendered output

2.31. Smart indentation (smart-indent)

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.

2.31.1. Example

```
Rendered output
                 Example code
                                        Typst
                                                  Enabled (default):
   *Enabled (default):*
2 ```py
                                                  1 def quicksort(L):
                                                                                               (py)
3 def quicksort(L):
                                                       qsort = lambda L: [] if L==[] else
     qsort = lambda L: [] if L==[] else
                                                       qsort([x for x in L[1:] if x< L[0]]) +</pre>
                                                  2
  qsort([x for x in L[1:] if x < L[0]]) +
                                                       L[0:1] + qsort([x for x in L[1:] if
   L[0:1] + qsort([x for x in L[1:] if
                                                       X>=L[0]
   x>=L[0]]
                                                  3 qsort(L)
5 qsort(L)
6
                                                  Disabled:
7 *Disabled:*
8 #codly(smart-indent: false)
                                                  1 def quicksort(L):
                                                                                               (py)
9 ```py
                                                       qsort = lambda L: [] if L==[] else
10 def quicksort(L):
                                                     qsort([x for x in L[1:] if x< L[0]]) +</pre>
                                                     L[0:1] + qsort([x for x in L[1:] if
     qsort = lambda L: [] if L==[] else
                                                     X>=L[0])
   qsort([x for x in L[1:] if x < L[0]]) +
                                                  3 qsort(L))
   L[0:1] + qsort([x for x in L[1:] if
   x>=L[0]])
12 qsort(L))
```

2.32. Breakable (breakable)

```
⟨→ Type bool
⟨*) Default value true

⊕ Contextual function × no

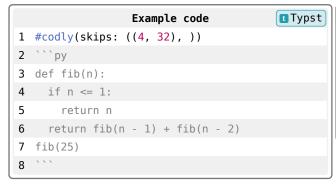
□ Automatically reset × no
```

Whether the codeblocks are breakable across page and column breaks.

2.33. 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.

2.33.1. Example



```
Rendered output

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

2.34. Skip line (skip-line)

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

2.34.1. Example

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

#### 2.35. Skip number (skip-number)

```
 ✓ Type
 content
 or none

 (*) Default value
 [...]

 ② Contextual function
 ✓ yes

 ③ Automatically reset
 × no
```

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.

#### 2.35.1. Example

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

```
Rendered output

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

 ...
37 fib(25)
```

#### 2.36. 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.

#### 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>.

#### 2.36.1. Example

```
Typst
 Example code
 #codly(
1
 annotations:(
2
3
4
 start: 1, end: 4,
 content: block(
5
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
     else:
18
       return fib(n - 1) + fib(n - 2)
19 fib(25)
20
```

2.37. Annotation formatter (annotation-format)

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

2.38. 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.
- 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.

2.38.1. Example

```
Example code
                                       ■ Typst
  #codly(highlights: (
     (line: 3, start: 2, end: none, fill:
2
     (line: 4, start: 13, end: 19, fill:
3
     green, tag: "(a)"),
     (line: 4, start: 26, fill: blue, tag:
4
     "(b)"),
5 ))
6 ```py
7 def fib(n):
8
  if n <= 1:
   return n
10 else:
   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>
```

2.39. Highlight radius (highlight-radius)

The radius of the highlights.

2.40. Highlight fill (highlight-fill)

| √y Type | function |
|------------------------------|---|
| {*} Default value | <pre>(color) => color.lighten(80%)</pre> |
| ② Contextual function | × no |
| ூ Automatically reset | × no |

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

2.41. Highlight stroke (highlight-stroke)

| √> Type | stroke or function |
|-----------------------|--------------------------|
| {*} Default value | (color) => 0.5pt + color |
| © Contextual function | × no |
| Automatically reset | × no |

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

2.42. Highlight inset (highlight-inset)

| √> Type | length |
|-----------------------|--------|
| {*} Default value | 0.32em |
| ② Contextual function | ✓ yes |
| ⊙ Automatically reset | × no |

The inset of the highlights.

2.43. Reference by (reference-by)

| √> Type | str |
|------------------------------|--------|
| {*} Default value | "line" |
| | ✓ yes |
| ூ Automatically reset | × no |

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.

2.44. Reference separator (reference-sep)

| √> Type | str or content |
|------------------------------|----------------|
| {*} Default value | "_" |
| © Contextual function | √ yes |
| ⑤ Automatically reset | × no |

The separator to use when referencing highlights and annotations.

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

| <pre></pre> <pre></pre> | function |
|-------------------------|--------------------------------|
| (*) Default value | <pre>numbering.with("1")</pre> |
| | × no |
| ⊙ Automatically reset | × no |

The format of the reference number line number, only used if $\ensuremath{\mathsf{reference-by}}$ is set to $\ensuremath{\mathsf{"line"}}$.

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 references lines directly, if you have set a label correctly, using the shorthand syntax <code>@my-label:1</code> to reference the second line (zero-based index) 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.

8

Experiment

You might notice that the second reference in the example below is formatted like a <u>link</u>. This is because it internally uses a <u>show ref</u>: .. show-rule which produces a link. This is a limitation of Typst and cannot be easily changed.

```
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.
10 But I can also reference a specific line of the label: @my-label:1.
```



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 - 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 highlight.

```
Example code

1 #codly(

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

3 ))

...

I can also reference a specific highlight by its label: @hl-1.
```

```
Rendered output

1 = Example typ
2 *Hello, world!*

Listing 2: A code block with a label
I can also reference a specific highlight by its label:
Listing 2-2.
```

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

```
Example code

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

4 )

I can also reference a specific highlight by its label: @hl-2.
```

```
Rendered output

1 = Example typ

2 *Hello, world! Highlight *

Listing 3: A code block with a label

I can also reference a specific 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:

```
Rendered output
                                       Typst
                 Example code
    #let icon = text(font: "tabler-icons",
                                                  1 Hello, world!
                                                                                           ₄Text
1
    size: lem, "\u{ebbe}")
    #codly(languages: (text: (icon: icon,
2
    name: "Text")))
    ```text
3
 Hello, world!
4
5
```

- 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:

```
Rendered output
 Example code
 Typst
 1 Hello, world!
 #let icon = text(font: "tabler-icons",

↓ Text
1
 size: 1em, "\u{ebbe}", baseline: 0.1em)
 #codly(languages: (text: (icon: icon,
2
 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.
```



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:
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.

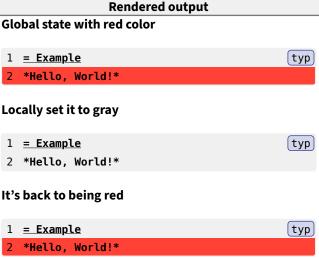
Warning

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.

i 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 ```py
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 }
ΙV
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

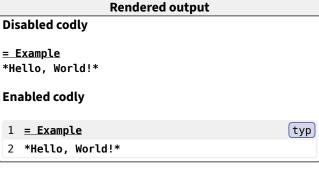
1 = Example
2 *Hello, World!*

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

5.6. Enable (codly-enable)

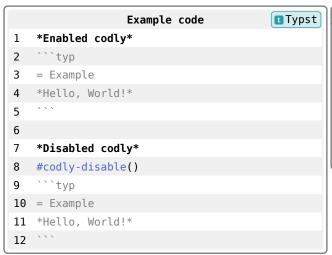
Enables codly globally, equivalent to codly (enabled: true).

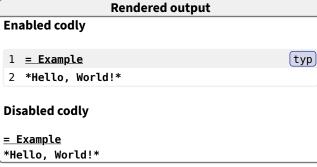




5.7. Disable (codly-disable)

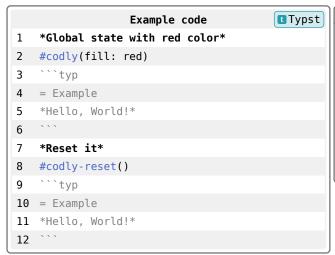
Disables codly globally, equivalent to codly (enabled: false).

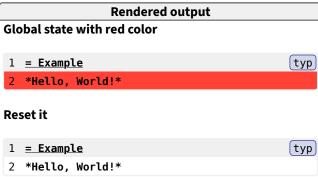




5.8. Reset (codly-reset)

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