Codly

codly

Configures codly. Is used in a similar way as set rules. You can imagine the following:

```
Typst code
1 // This is a representation of the actual code.
   // The actual code behave like a set rule that uses `state`.
3 let codly(
4
      enabled: true,
5
      offset: 0,
6
      range: none,
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,
      zebra-fill: luma(240),
14
15
      stroke: 1pt + luma(240),
16
      lang-inset: 0.32em,
17
      lang-outset: (x: 0.32em, y: 0pt),
18
      lang-radius: 0.32em,
      lang-stroke: (lang) => lang.color + 0.5pt,
19
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
      breakable: false,
25
      smart-indent: false,
26
      annotations: none,
27
      annotation-format: numbering.with("(1)"),
28 ) = {}
```

Each argument is explained below.

Display style

Codly displays your code in three sections:

- The line number, if number-format is not none
- The language block, with a fill and a stroke, only appears on the first line
- The code itself with optional zebra striping

The block as a whole is surrounded by a stroke.

Note about arguments:

Some arguments can be a function that takes no arguments and returns the value. They are called within a **context** that provides the current location. They can be used to have more dynamic control over the value, without the need for sometimes slow state updates.

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.

Default: trueType: bool

• Can be a contextual function: no



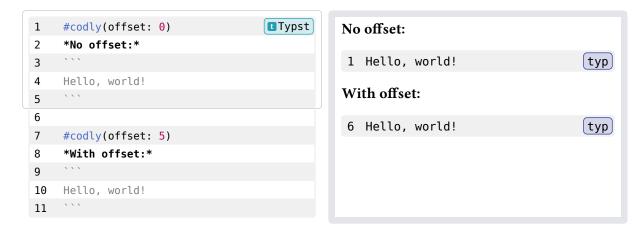
Offset (offset)

The offset to apply to line numbers.

Note that the offset gets reset automatically after every code block.

Default: 0Type: int

• Can be a contextual function: yes



Range (range)

The range of line numbers to display.

Note that the range gets reset automatically after every code block.

The same behavior can be achieved using the <code>codly-range()</code> function.

- Default: none
- Type: (int, int) or none
- Can be a contextual function: yes

```
1 #codly(range: (2, 4))
                                  Typst
                                              2
                                                     if n <= 1:
                                                                                  (py)
  ```py
2
 3
 return n
3 def fib(n):
 return fib(n - 1) + fib(n - 2)
4
 if n <= 1:
 return n
 return fib(n - 1) + fib(n - 2)
6
7 fib(25)
8
```

### Skips (skips)

Insert a skip at the specified line numbers, setting its offset to the length of the skip. The skip is formatted using skip-number and skip-line.

Each skip is an array with two values: the position and length of the skip.

Note that the skips gets reset automatically after every code block.

The same behavior can be achieved using the codly-skip() function, which appends one or more skips to the list of skips.

- Default: ()
- Type: array or none
- Can be a contextual function: yes

```
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 ```</pre>
```

## 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.

## Skip line ( skip-line )

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

## Languages (languages)

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

Alternatively, the value can be a string, in which case it is used as the name of the language. And no icon is shown and the default color is used.

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

- Default: (:)Type: dict
- Can be a contextual function: no

```
1 #codly(
2 languages: (
3 py: (name: "Python", color: red,
icon: "2")
4)
5)
6 ```py
7 print('Hello, world!')
8 print('Goodbye, world!')
9 ```
```



# Display 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.

• Default: true

- Type: bool
- Can be a contextual function: yes

```
1 #codly(display-name: false)
2 ```py
3 print('Hello, world!')
4 print('Goodbye, world!')
5 ```
1 print('Hello, world!')
2 print('Goodbye, world!')
```

## Display icon (display-icon)

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

Default: trueType: bool

• Can be a contextual function: yes

```
Typst
1
 #codly(
 1 print('Hello, world!')
 Python
2
 display-icon: false,
 2 print('Goodbye, world!')
 languages: (
 py: (name: "Python", color: red,
4
 icon: "% ")
5
)
6
  ```py
7
    print('Hello, world!')
8
9
    print('Goodbye, world!')
10
```

Default color (default-color)

The default color to use for language blocks.

This only applies if you're using the default language block formatter. Also note that it is also passed as a named argument to the language block formatter if you've defined your own.

• **Default**: rgb("#283593") (a shade of blue)

• Type: color

• Can be a contextual function: yes

```
1 #codly(default-color: orange)
2 ```py
3 print('Hello, world!')
4 print('Goodbye, world!')
5 ```
1 print('Hello, world!')
2 print('Goodbye, world!')
```

Radius (radius)

The radius of the border of the code block.

Default: 0.32emType: length

• Can be a contextual function: yes

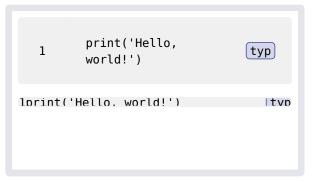
Inset (inset)

The inset of the code block.

Default: 0.32emType: length

• Can be a contextual function: yes

```
1 #codly(inset: 10pt)
2 ```
3 print('Hello, world!')
4 ```
5 #codly(inset: 0pt)
6 ```
7 print('Hello, world!')
8 ```
```



Fill (fill)

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

• Default: none

• Type: none , color , gradient , or pattern

• Can be a contextual function: yes



Zebra color (zebra-fill)

The fill of the code block when zebra-striped, none to disable zebra-striping.

```
1 #codly(
    zebra-fill:
    gradient.linear(..color.map.flare),
3 )
4 ```
5 print('Hello, world!')
6 print('Goodbye, world!')
7 ```
```

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

• Default: none

• Type: none , color , gradient , or pattern

• Can be a contextual function: yes

Stroke (stroke)

The stroke of the code block.

Default: 1pt + luma(240)Type: none or stroke

• Can be a contextual function: yes

```
1 #codly(
2 stroke: 1pt +
gradient.linear(..color.map.flare),
3 )
4 ```
5 print('Hello, world!')
6 print('Goodbye, world!')
7 ```
```

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

Language block inset (lang-inset)

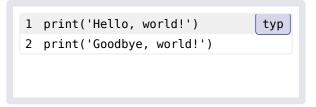
The inset of the language block.

This only applies if you're using the default language block formatter.

Default: 0.32emType: length

• Can be a contextual function: yes

```
1 #codly(lang-inset: 5pt)
2 ```
3 print('Hello, world!')
4 print('Goodbye, world!')
5 ```
```



Language block 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.

• **Default**: (x: 0.32em, y: 0pt)

- Type: dict
- Can be a contextual function: yes

Language block radius (lang-radius)

The radius of the language block.

Default: 0.32emType: length

• Can be a contextual function: yes

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

Language block stroke (lang-stroke)

The stroke of the language block.

- Default: none
- Type: none, stroke, or a function that takes in the language dict or none

(see argument languages) and returns a stroke.

• Can be a contextual function: no

```
#codly(lang-stroke: 1pt +
                                   Typst
                                               1 print('Hello, world!')
                                                                                   typ
1
   red)
                                               2 print('Goodbye, world!')
2
    print('Hello, world!')
3
                                               1 print('Hello, world!')
                                                                                   typ
4
    print('Goodbye, world!')
                                               2 print('Goodbye, world!')
    #codly(lang-stroke: (lang) => 2pt +
   lang.color)
7
    print('Hello, world!')
9
    print('Goodbye, world!')
10
```

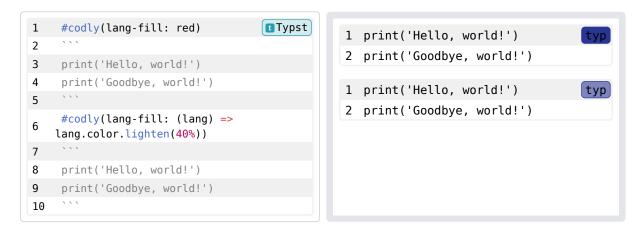
Language block fill (lang-fill)

The fill of the language block.

- Default: none
- Type: none, color, gradient, pattern, or a function that takes in

the language dict or none (see argument languages) and returns a fill.

• Can be a contextual function: no



Language block formatter (lang-format)

The formatter for the language block.

A value of none will not display the language block. To use the default formatter, use auto.

Default: auto Type: function

• Can be a contextual function: no

```
1 #codly(lang-format: (..) =>
1 [No!])
2 ```
3 print('Hello, world!')
4 print('Goodbye, world!')
5 ```
1 print('Hello, world!')
2 print('Goodbye, world!')
```

Line number formatter (number-format)

The formatter for line numbers.

- **Default**: numbering.with("1")
- Type: function
- Can be a contextual function: false

```
#codly(number-format: (n) =>
numbering("I.", n))

indexing the state of the st
```

Line number alignment (number-align)

The alignment of the line numbers.

- **Default**: left + horizon
- Type: top, horizon, or bottom
- Can be a contextual function: yes

```
#codly(number-align: right +
                                    Typst
1
   horizon)
    ```py
2
3
 # Iterative Fibonacci
 # As opposed to the recursive
 # version
6
 def fib(n):
7
 if n <= 1:
8
 return n
9
 last, current = 0, 1
10
 for in range(2, n + 1):
 last, current = current, last +
11
 current
12
 return current
 print(fib(25))
13
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 print(fib(25))</pre>
```

## 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 off by default since it is slower.

Default: falseType: bool

• Can be a contextual function: no

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

### **Bêta: 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

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

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

**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.

This should be considered a Bêta feature.

• Default: none

• Type: array or none

• Can be a contextual function: yes

```
#codly(smart-indent: true,
 Typst
 1 def fib(n):
 (py
 annotation-format: none)
 if n <= 1:
 Function body
 #codly(annotations: ((start: 1, end:
 4, content: block(width: 2em,
 return n
 rotate(-90deg, align(center, box(width:
 else:
 100pt)[Function body]))),))
 return (fib(n-1)+fib(n-2))
   ```py
                                                  6 print(fib(25))
    def fib(n):
     if n <= 1:
6
      return n
7
     else:
      return (fib(n-1)+fib(n-2))
8
9
    print(fib(25))
```

Bêta: Annotation format (annotation-format)

The format of the annotation number.

Can be **none** or a function that formats the annotation number.

• **Default**: numbering.with("(1)")

• Type: none or function

• Can be a contextual function: no

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.

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

Note: This feature performs what I loosely call "globbing", this means that instead of highlighting individual characters, it highlights the whole word or sequence of characters that the start and end positions are part of. This is done to avoid having to deal with the complexity of highlighting individual characters, and needing to re-style them manually. While also making the API a tad less error prone at the cost of sometimes goofy looking highlights if they overlap.

Limitation: If there is a tag and the line overflows, it can look kind of goofy. This is a trade-off, as it would need to either use a grid which will prevent overflowing all together, or use a more complex multi-box based system (which is what is used) that can sometimes look goofy (see example below).

• Default: none

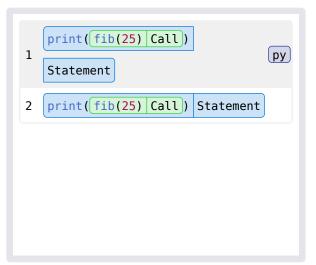
• Type: array or none

• Can be a contextual function: yes

```
#codly(highlights: (
                                     Typst
     (line: 3, start: 4, end: none, fill:
2
   red).
     (line: 4, start: 14, end: 22, fill:
3
   green, tag: "(a)"),
     (line: 4, start: 29, end: 38, fill:
   blue, tag: "(b)"),
5
    ))
    ```py
7
 def fib(n):
8
 if n \le 1:
9
 return n
10
 else:
 return (fib(n - 1) + fib(n - 2))
11
12
 print(fib(25))
13
```

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

```
Typst
 #codly(highlights: (
 (line: 0, start: 7, end: 13, fill:
2
 green, tag: "Call"),
 (line: 0, fill: blue, tag:
3
 "Statement"),
 (line: 1, start: 7, end: 13, fill:
4
 green, tag: "Call"),
 (line: 1, fill: blue, tag:
5
 "Statement"),
6
   ```py
8
    print(fib(25))
9
    print(fib(25))
10
```



Highlight radius (highlight-radius)

The radius of the border of the highlights.

Default: 0.32emType: length

• Can be a contextual function: yes

Highlight fill (highlight-fill)

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

```
• Default: (color) => color.lighten(80%)
```

• Type: function

• Can be a contextual function: no

Highlight stroke (highlight-stroke)

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

• **Default**: (color) => 0.5pt + color

• Type: function

• Can be a contextual function: no

Highlight inset (highlight-inset)

The inset of the highlights.

Default: 0.32emType: length

• Can be a contextual function: yes

Breakable (breakable)

Whether the code block is breakable.

Default: falseType: bool

• Can be a contextual function: no

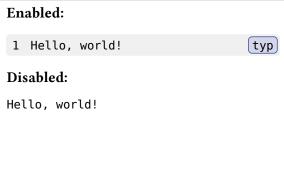
Parameters

```
codly(..args)
```

codly-disable

Disables codly.





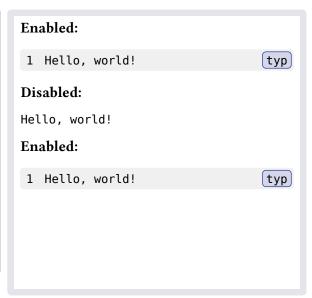
Parameters

```
codly-disable()
```

codly-enable

Enables codly.

```
*Enabled:*
                                   Typst
1
2
3
    Hello, world!
4
5
    #codly-disable()
    *Disabled:*
6
7
    Hello, world!
8
9
10
11
    #codly-enable()
   *Enabled:*
12
13
14 Hello, world!
15
```



Parameters

codly-enable()

codly-init

Initializes the codly show rule.

```
1 #show: codly-init

Parameters
```

codly-init(body)

codly-offset

Lets you set a line number offset.

Parameters

```
codly-offset(offset)
```

codly-range

Lets you set a range of line numbers to highlight. Similar to codly(range: (start, end)).

```
1 #codly-range(start: 2)
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    ```</pre>
```

Parameters

```
codly-range(
   start,
   end
)
```

codly-skip

Appends a skip to the list of skips.

```
1 #codly-skip(4, 32)
2 ```
3 Hello, world!
4 Goodbye, world!
5 ```
```



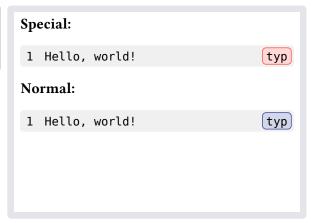
Parameters

```
codly-skip(
  position,
  length
)
```

local

Allows setting codly setting locally. Anything that happens inside the block will have the settings applied only to it. The pre-existing settings will be restored after the block. This is useful if you want to apply settings to a specific block only.

```
*Special:*
1
                                 ■ Typst
2
    #local(default-color: red)[
3
4
     Hello, world!
5
6
   ]
7
8
    *Normal:*
9
10 Hello, world!
11
```

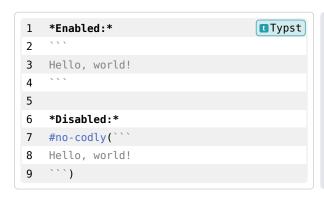


Parameters

```
local(
  body,
    ..args
)
```

no-codly

Disabled codly locally.





Parameters

```
no-codly(body)
```