Codly

codly

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

Typst code

```
// This is a representation of the actual code.
// The actual code behave like a set rule that uses `state`.
let codly(
    enabled: true,
    offset: 0,
    range: none,
    languages: (:),
    display-name: true,
    display-icon: true,
    default-color: rgb("#283593"),
    radius: 0.32em,
    inset: 0.32em,
    fill: none,
    zebra-fill: luma(240),
    stroke: 1pt + luma(240),
    lang-inset: 0.32em,
    lang-outset: (x: 0.32em, y: 0pt),
    lang-radius: 0.32em,
    lang-stroke: (lang) => lang.color + 0.5pt,
    lang-fill: (lang) => lang.color.lighten(80%),
    lang-format: codly.default-language-block,
    number-format: (number) => [ #number ],
    number-align: left + horizon,
    smart-indent: false,
    annotations: none,
    annotation-format: numbering.with("(1)"),
    highlights: none,
    highlight-radius: 0.32em,
    highlight-fill: (color) => color.lighten(80%),
    highlight-stroke: (color) => 0.5pt + color,
    highlight-inset: 0.32em,
    reference-by: line,
    reference-sep: "-",
    reference-number-format: numbering.with("1"),
    breakable: false,
) = {}
```

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.

```
|-------------------------------------------------------------|
|1 | [line, zebra]                                | [lang block] |
|2 | [line, non-zebra]                                         |
|3 | [line, zebra]                                             |
|4 | [line, non-zebra]                                         |
|-------------------------------------------------------------|
```

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.

In figure code blocks:

If the code block is in a figure, additional features are available for referencing annotations and highlights.

````
1  #codly(annotations: ((start: 0, end: 0, content: "Function call", label: <func-call>), ))
2  #codly(highlights: ((line: 1, start: 7, end: none, fill: green, tag: "Call", label: <call>), ))
3  #figure(
4    caption: ["Hello, world!" in Python]
5  )[
6  ```py
7    print("Hello, world!")
8    print("Hello, world!")
9  ```
10 ] <fig-hello>
11
12 I can later reference the figure: @fig-hello. But I can also reference the
13 annotation: @func-call. And finally, I can reference the highlight: @call.
````

Typst

```
1  print("Hello, world!")          } (1) Function call py
2  print("Hello, world!") Call
```

Listing 1: “Hello, world!” in Python

I can later reference the figure: Listing 1. But I can also reference the annotation: Listing 1-1. And finally, I can reference the highlight: Listing 1-2.

This behaviour can be completely customized using the `reference-by`, `reference-sep`, and `reference-number-format` fields listed below.

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**: `true`

- **Type**: `bool`
- **Can be a contextual function**: no

```typst
#codly(enabled: true)
*Enabled:*
```
Hello, world!
```

#codly(enabled: false)
*Disabled:*
```
Hello, world!
```
```

Enabled:

```typ
Hello, world!
```

Disabled:

Hello, world!

Offset (`offset`)

The offset to apply to line numbers.

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

- **Default**: `0`
- **Type**: `int`
- **Can be a contextual function**: yes

```typst
#codly(offset: 0)
*No offset:*
```
Hello, world!
```

#codly(offset: 5)
*With offset:*
```
Hello, world!
```
```

No offset:

```typ
Hello, world!
```

With offset:

```typ
Hello, world!
```

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 `codly-range()` function.

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

```typst
#codly(range: (2, 4))
```py
def fib(n):
    if n <= 1:
        return n
    return fib(n - 1) + fib(n - 2)
fib(25)
```
```

```py
    if n <= 1:
        return n
    return fib(n - 1) + fib(n - 2)
```

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

```typst
#codly(skips: ((4, 32), ))
```py
def fib(n):
    if n <= 1:
        return n
    return fib(n - 1) + fib(n - 2)
fib(25)
```
```

```py
def fib(n):
    if n <= 1:
        return n
    return fib(n - 1) + fib(n - 2)
…                   …
fib(25)
```

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

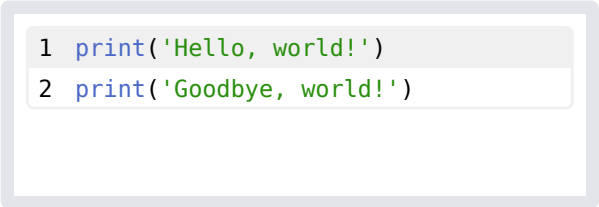
```typst
#codly(
  languages: (
    py: (name: "Python", color: red, icon: "🐍")
  )
)
```py
print('Hello, world!')
print('Goodbye, world!')
```
```

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

```typst
#codly(display-name: false)
```py
print('Hello, world!')
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**: `true`
- **Type**: `bool`
- **Can be a contextual function**: yes

```typst
#codly(
  display-icon: false,
  languages: (
    py: (name: "Python", color: red, icon: "🐍")
  )
)
```py
print('Hello, world!')
print('Goodbye, world!')
```
```

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

```typst
#codly(default-color: orange)
```

````
```py
print('Hello, world!')
print('Goodbye, world!')
```
````

```py
print('Hello, world!')
print('Goodbye, world!')
```

Radius (`radius`)

The radius of the border of the code block.

- **Default**: `0.32em`
- **Type**: `length`
- **Can be a contextual function**: yes

```typst
#codly(radius: 0pt)
```

````
```
print('Hello, world!')
```
#codly(radius: 20pt)
```
print('Hello, world!')
```
````

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

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

Inset (`inset`)

The inset of the code block.

- **Default**: `0.32em`
- **Type**: `length`
- **Can be a contextual function**: yes

```typst
#codly(inset: 10pt)
```

````
```
print('Hello, world!')
```
#codly(inset: 0pt)
```
print('Hello, world!')
```
````

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

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

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

```typst
#codly(
  fill: gradient.linear(..color.map.flare),
)
```

```typ
print('Hello, world!')
print('Goodbye, world!')
```

Zebra color (`zebra-fill`)

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

```typst
#codly(
  zebra-fill: gradient.linear(..color.map.flare),
)
```

```typ
print('Hello, world!')
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

```typst
#codly(
  stroke: 1pt + gradient.linear(..color.map.flare),
)
```

```typ
print('Hello, world!')
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**: `0.32em`
- **Type**: `length`

- **Can be a contextual function**: yes

```typst
#codly(lang-inset: 5pt)
```
````
```
print('Hello, world!')
print('Goodbye, world!')
```
````

```typ
print('Hello, world!')
print('Goodbye, world!')
```

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

```typst
#codly(lang-outset: (x: -10pt, y: 5pt))
```
````
```
print('Hello, world!')
print('Goodbye, world!')
```
````

```typ
print('Hello, world!')
print('Goodbye, world!')
```

Language block radius (`lang-radius`)

The radius of the language block.

- **Default**: `0.32em`
- **Type**: `length`
- **Can be a contextual function**: yes

```typst
#codly(lang-radius: 10pt)
```
````
```
print('Hello, world!')
print('Goodbye, world!')
```
````

```typ
print('Hello, world!')
print('Goodbye, world!')
```

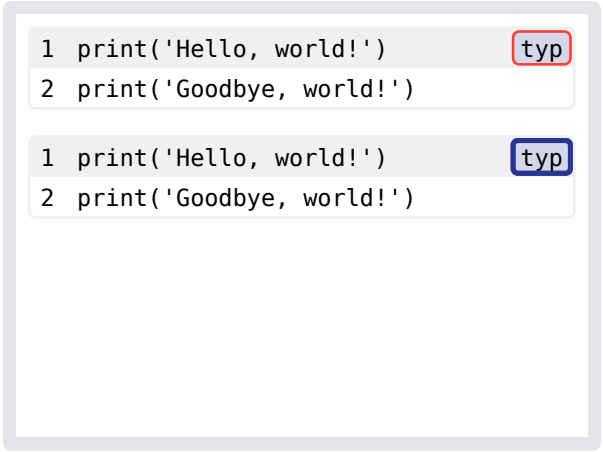
Language block stroke (`lang-stroke`)

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

```typst
#codly(lang-stroke: 1pt + red)
```
```
print('Hello, world!')
print('Goodbye, world!')
```
```typst
#codly(lang-stroke: (lang) => 2pt + lang.color)
```
```
print('Hello, world!')
print('Goodbye, world!')
```



Language block fill (`lang-fill`)

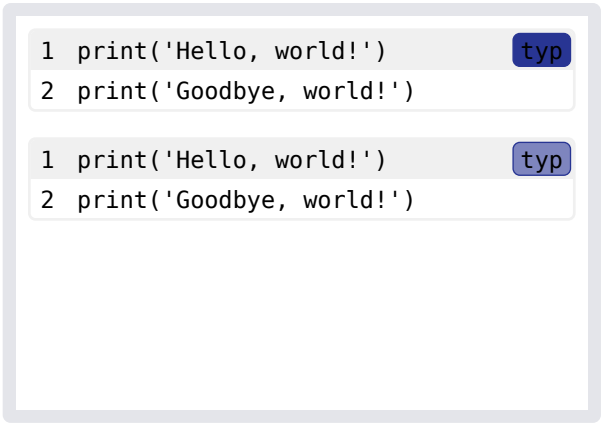
The fill of the language block.

- **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

```typst
#codly(lang-fill: red)
```
```
print('Hello, world!')
print('Goodbye, world!')
```
```typst
#codly(lang-fill: (lang) => lang.color.lighten(40%))
```
```
print('Hello, world!')
print('Goodbye, world!')
```



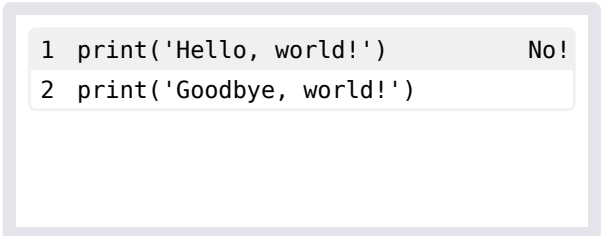
Language block formatter (`lang-format`)

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

```typst
#codly(lang-format: (..) => [ No! ])
```
```
print('Hello, world!')
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))
```
print('Hello, world!')
print('Goodbye, world!')
```
````

```
I.  print('Hello, world!')
II. print('Goodbye, world!')
```

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 + horizon)
```py
# Iterative Fibonacci
# As opposed to the recursive
# version
def fib(n):
  if n <= 1:
    return n
  last, current = 0, 1
  for _ in range(2, n + 1):
    last, current = current, last + current
  return current
print(fib(25))
```
````

```py
# Iterative Fibonacci
# As opposed to the recursive
# version
def fib(n):
  if n <= 1:
    return n
  last, current = 0, 1
  for _ in range(2, n + 1):
    last, current = current, last + current
  return current
print(fib(25))
```

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**: `false`
- **Type**: `bool`
- **Can be a contextual function**: no

````
#codly(smart-indent: true)
```py
def fib(n):
  if n <= 1:
    return n
  else:
    return (fib(n - 1) + fib(n - 2))
print(fib(25))
```
````

```py
def fib(n):
  if n <= 1:
    return n
  else:
    return (fib(n - 1) + fib(n - 2))
print(fib(25))
```

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
- `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)` .

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

Typst

```py
def fib(n):
 if n <= 1:
  return n
 else:
  return (fib(n-1)+fib(n-2))
print(fib(25))
```

Function body

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.
- `label`: **if and only if** the code block is in a `figure`, and it has a `tag`, sets the label by which the highlight can be referenced.

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: (
  (line: 3, start: 4, end: none, fill: red),
  (line: 4, start: 14, end: 22, fill: green, tag: "(a)"),
  (line: 4, start: 29, end: 38, fill: blue, tag: "(b)"),
 ))
```py
def fib(n):
  if n <= 1:
    return n
  else:
    return (fib(n - 1) + fib(n - 2))
print(fib(25))
```
````

```py
def fib(n):
  if n <= 1:
    return n
  else:
    return (fib(n - 1) (a) + fib(n - 2) (b))
print(fib(25))
```

````
#codly(highlights: (
  (line: 0, start: 7, end: 13, fill: green, tag: "Call"),
  (line: 0, fill: blue, tag: "Statement"),
  (line: 1, start: 7, end: 13, fill: green, tag: "Call"),
  (line: 1, fill: blue, tag: "Statement"),
 ))
```py
print(fib(25))
print(fib(25))
```
````

```py
print(fib(25) Call) Statement
print(fib(25) Call) Statement
```

Highlight radius (`highlight-radius`)

The radius of the border of the highlights.

- **Default**: `0.32em`
- **Type**: `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**: `0.32em`
- **Type**: `length`
- **Can be a contextual function**: yes

Reference mode (`reference-by`)
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
- **Default**: `"line"`
- **Type**: `str`
- **Can be a contextual function**: yes

Reference separator (`reference-sep`)
The separator to use between the figure reference and the reference itself.

- **Default**: `"-"`
- **Type**: `str`
- **Can be a contextual function**: yes

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

- **Default**: `numbering.with("1")`
- **Type**: `function`
- **Can be a contextual function**: no

Breakable (`breakable`)
Whether the code block is breakable.

- **Default**: `false`
- **Type**: `bool`
- **Can be a contextual function**: no

Parameters

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

codly-disable

Disables codly.

```typst
*Enabled:*
```
Hello, world!
```

#codly-disable()
*Disabled:*
```
Hello, world!
```
```

Enabled:

```typ
Hello, world!
```

Disabled:

Hello, world!

Parameters

`codly-disable()`

codly-enable

Enables codly.

```typst
*Enabled:*
```
Hello, world!
```

#codly-disable()
*Disabled:*
```
Hello, world!
```

#codly-enable()
*Enabled:*
```
Hello, world!
```
```

Enabled:

```typ
Hello, world!
```

Disabled:

Hello, world!

Enabled:

```typ
Hello, world!
```

Parameters

`codly-enable()`

codly-init

Initializes the codly show rule.

```typ
#show: codly-init
```

Parameters

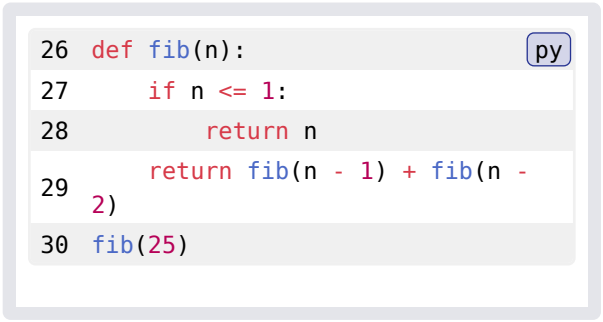
`codly-init(body)`

codly-offset

Lets you set a line number offset.

```typst
#codly-offset(offset: 25)
```

```py
def fib(n):
    if n <= 1:
        return n
    return fib(n - 1) + fib(n - 2)
fib(25)
```



Parameters

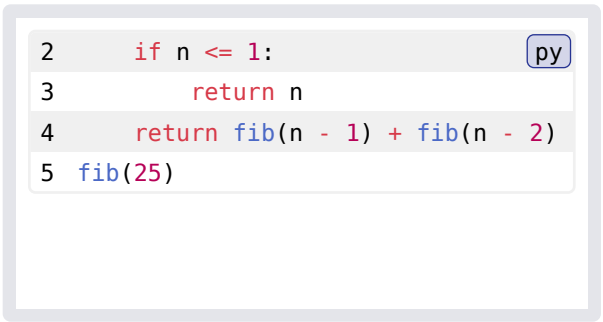
`codly-offset(offset)`

codly-range

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

```typst
#codly-range(start: 2)
```

```py
def fib(n):
    if n <= 1:
        return n
    return fib(n - 1) + fib(n - 2)
fib(25)
```



Parameters

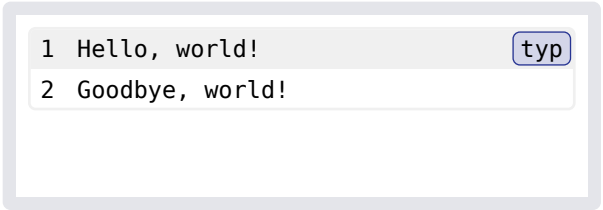
```
codly-range(
  start,
  end
)
```

codly-skip

Appends a skip to the list of skips.

```typst
#codly-skip(4, 32)
```

```
Hello, world!
Goodbye, world!
```



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:*
#local(default-color: red)[
  ```
  Hello, world!
  ```
]

*Normal:*
```
Hello, world!
```
````

Special:

```
Hello, world!
```

Normal:

```
Hello, world!
```

Parameters

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

no-codly

Disabled codly locally.

````
*Enabled:*
```
Hello, world!
```

*Disabled:*
#no-codly(```
Hello, world!
```)
````

Enabled:

```
Hello, world!
```

Disabled:

Hello, world!

Parameters

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