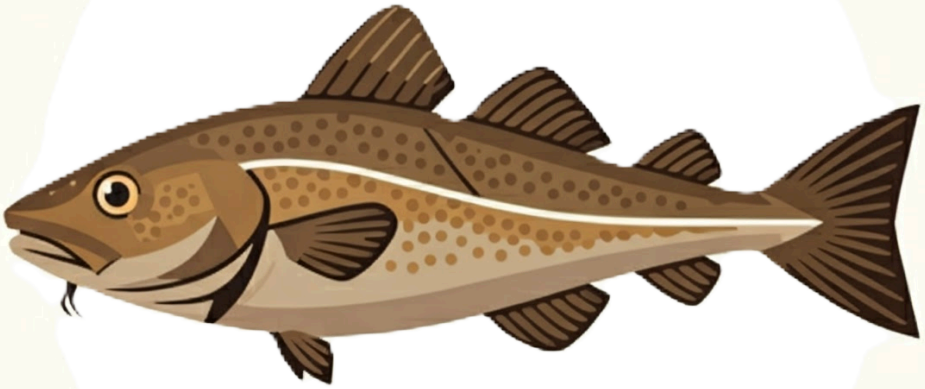
Always start a new Typst project by importing codly



Codly v1.1.0 manual

Your code blocks on steroids

O RLY?

Dherse

Contents

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

```
#import "@preview/codly:1.0.0": *

#show: codly-init
```

Rendered output

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 Typst Universe page.

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

Example code Typst

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

Rendered output

```
Hello, world!
```

1.2. Enabling and disabling codly

By defauly Codly will be enabled after initialization. However, disabling codly can be done using the `codly-disable` function, the `enabled` argument of the `codly` function, or the `no-codly` function. To enable Codly again, use 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

```
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"),
  header: none,
  header-repeat: false,
  header-transform: (x) => x,
  header-cell-args: (),
  footer: none,
  footer-repeat: false,
  footer-transform: (x) => x,
  footer-cell-args: (),
  breakable: false,
) = {}
```

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.

2.1. Enabled (`enabled`)

Type	`bool`
Default value	`true`
Contextual function	✓ yes
Automatically reset	✗ no

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

Example code (Typst)

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

Enabled = false:
#codly(enabled: false)
```typ
Hello, world!
```
```

Rendered output

Enabled = true:

```typ
Hello, world!
```

Enabled = false:

Hello, world!

2.2. Header (`header`)

Type	`content` or `none`
Default value	`none`
Contextual function	✓ yes
Automatically reset	✓ yes

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-cell-args` and `header-transform` arguments.

2.2.1. Example

Example code (Typst)

```typ
#codly(header: [*Hello, world!*])
```typ
Hello, world!
```
```

Rendered output

Hello, world! (typ)

```typ
Hello, world!
```

2.3. Header Repeat (`header-repeat`)

Type	`bool`
Default value	`false`
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 `breakable`, 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`)

Type	`array` , `dictionary` , or `arguments`
Default value	`()`
Contextual function	✓ yes
Automatically reset	no

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

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

2.4.1. Example

Example code Typst

```
//Centering the header:
#codly(
  header: [*Hello, world!*],
  header-cell-args: (align: center, )
)

```typ
Hello, world!
```
```

Rendered output

Hello, world! typ

```
1 Hello, world!
```

2.5. Header Transform (`header-transform`)

Type	`function`
Default value	`(x) => x`
Contextual function	✗ no
Automatically reset	no

Function that transforms the header into arbitrary content to be stored in the `grid.cell` . 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

Example code Typst

```
//Making the header bold and blue:
#codly(
   header: [Hello, world!],
   header-transform: (x) => {
     set text(fill: blue)
     strong(x)
   }
)

```typ
Hello, world!
```
```

Rendered output

Hello, world! typ

```
1 Hello, world!
```

2.6. Footer (`footer`)

Type	`content` or `none`
Default value	`none`
Contextual function	✓ yes
Automatically reset	✓ yes

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

2.6.1. Example

Example code Typst

```
#codly(footer: [*Hello, world!*])
```typ
Hello, world!
```
```

Rendered output

```
Hello, world!
```typ
Hello, world!
```

2.7. Footer Repeat (`footer-repeat`)

Type	`bool`
Default value	`false`
Contextual function	✓ yes
Automatically reset	no

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

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

Type	`array` , `dictionary` , or `arguments`
Default value	`()`
Contextual function	✓ yes
Automatically reset	no

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

2.8.1. Example

Example code Typst

```
//Centering the footer:
#codly(
 footer: [*Hello, world!*],
 footer-cell-args: (align: center, )
)

```typ
Hello, world!
```
```

**Rendered output**

```
Hello, world!
```typ
Hello, world!
```

2.9. Footer Transform (`footer-transform`)

Type	function
Default value	`(x) => x`
Contextual function	✗ no
Automatically reset	no

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

2.9.1. Example

Example code (Typst)

```
1  //Making the footer bold and blue:
2  #codly(
3    footer: [Hello, world!],
4    footer-transform: (x) => {
5      set text(fill: blue)
6      strong(x)
7    }
8  )
9
10 ```typ
11 Hello, world!
12 ```
```

Rendered output

```
1 Hello, world!                                    typ
Hello, world!
```

2.10. Offset (`offset`)

Type	int
Default value	0
Contextual function	✓ yes
Automatically reset	✓ yes

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

Example code (Typst)

```
1  *No offset:*
2  ```typ
3  Hello, world!
4  ```
5
6  *Offset by 5:*
7  #codly(offset: 5)
8  ```typ
9  Hello, world!
10 ```
```

Rendered output

No offset:

```
1 Hello, world!                                    typ
```

Offset by 5:

```
6 Hello, world!                                    typ
```

2.11. Range (`range`)

Type	`none` or `array`
Default value	`none`
Contextual function	✓ yes
Automatically reset	✓ yes

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.

2.11.1. Example

Example code (Typst)

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

Rendered output

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

2.12. Languages (`languages`)

Type	`dictionary`
Default value	`(:)`
Contextual function	✓ yes
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.

2.12.1. Example

Example code (Typst)

```
#codly(
  languages: (
    py: (
      name: [Python], color: green, icon:
      "🐍"
    ),
  )
)
```py
def fib(n):
  if n <= 1:
    return n
  return fib(n - 1) + fib(n - 2)
fib(25)
```
```

Rendered output

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

2.13. Default language color (`default-color`)

Type	`color` , `gradient` , or `pattern`
Default value	`rgb("#283593")`
Contextual function	✓ yes
Automatically reset	no

The default color to use for language blocks. Used when a language is not defined in the `languages` argument. Also note that it is only used when the `lang-format` is its `auto` 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` .

2.13.1. Example

Example code (Typst)

```
*Default color:*
```py
print('Hello, world!')
print('Hello, world!')
```
Overriden default color:
#codly(default-color: orange)
```py
print('Hello, world!')
print('Hello, world!')
```
```

Rendered output

Default color:

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

Overriden default color:

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

2.14. Radius (`radius`)

Type	`length`
Default value	`0.32em`
Contextual function	✓ yes
Automatically reset	no

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

2.14.1. Example

Example code (Typst)

```
*Default radius:*
```py
print('Hello, world!')
print('Hello, world!')
```
Overriden radius:
#codly(radius: 2em)
```py
print('Hello, world!')
print('Hello, world!')
```
Zero radius:
#codly(radius: 0pt)
```py
print('Hello, world!')
print('Hello, world!')
```
```

Rendered output

Default radius:

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

Overriden radius:

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

Zero radius:

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

2.15. Inset (`inset`)

Type	`length`
Default value	`0.32em`
Contextual function	✓ yes
Automatically reset	no

Inset of the code lines, that is the distance between the border and the code lines.

2.15.1. Example

Example code Typst

```
*Default inset:*
```py
print('Hello, world!')
```
Overriden inset:
#codly(inset: 1em)
```py
print('Hello, world!')
```
```

Rendered output

Default inset:

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

Overriden inset:

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

2.16. Fill (`fill`)

Type	`none` , `color` , `gradient` , or `pattern`
Default value	`none`
Contextual function	✓ yes
Automatically reset	no

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

2.16.1. Example

Example code Typst

```
*Default fill:*
```py
print('Hello, world!')
print('Hello, world!')
```
Overriden fill:
#codly(fill:
gradient.linear(..color.map.flare))
```py
print('Hello, world!')
print('Hello, world!')
```
No fill:
#codly(fill: none)
```py
print('Hello, world!')
print('Hello, world!')
```
```

Rendered output

Default fill:

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

Overriden fill:

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

No fill:

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

2.17. Zebra fill (`zebra-fill`)

Type	`none` , `color` , `gradient` , or `pattern`
Default value	`luma(240)`
Contextual function	✓ yes
Automatically reset	no

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

2.17.1. Example

Example code Typst

```
*Default zebra:*
```py
print('Hello, world!')
print('Hello, world!')
```
No zebra:
#codly(zebra-fill: none)
```py
print('Hello, world!')
print('Hello, world!')
```
Overriden zebra:
#codly(zebra-fill:
gradient.linear(..color.map.flare))
```py
print('Hello, world!')
```
```

Rendered output

Default zebra:

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

No zebra:

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

Overriden zebra:

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

2.18. Stroke (`stroke`)

Type	`none` or `stroke`
Default value	`1pt + luma(240)`
Contextual function	✓ yes
Automatically reset	no

The stroke to surround the whole code block with?

2.18.1. Example

Example code Typst

```
*Default stroke:*
```py
print('Hello, world!')
```
No stroke:
#codly(stroke: none)
```py
print('Hello, world!')
```
Overriden stroke:
#codly(stroke: 1pt + blue)
```py
print('Hello, world!')
```
```

Rendered output

Default stroke:

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

No stroke:

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

Overriden stroke:

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

2.19. Language box inset (`lang-inset`)

Type	length or dictionary
Default value	0.32em
Contextual function	✓ yes
Automatically reset	no

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

2.19.1. Example

Example code Typst

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

Rendered output

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

2.20. Language box outset (`lang-outset`)

Type	dictionary
Default value	(x: 0.32em, y: 0em)
Contextual function	✓ yes
Automatically reset	no

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.

2.20.1. Example

Example code Typst

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

Rendered output

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

2.21. Language box radius (`lang-radius`)

Type	length
Default value	0.32em
Contextual function	✓ yes
Automatically reset	no

The radius of the border of the language block.

2.21.1. Example

Example code Typst

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

Rendered output

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

2.22. Language box stroke (`lang-stroke`)

Type	`none` , `stroke` , or `function`
Default value	`(lang) => lang.color + 0.5pt`
Contextual function	× no
Automatically reset	no

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.

2.22.1. Example

Example code (Typst)

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

Rendered output

Fixed stroke:

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

Function mapping:

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

2.23. Language box fill (`lang-fill`)

Type	`none` , `color` , `gradient` , `pattern` , or `function`
Default value	`(lang) => lang.color.lighten(80%)`
Contextual function	× no
Automatically reset	no

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

2.23.1. Example

Example code (Typst)

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

Rendered output

Fixed fill:

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

Function mapping:

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

2.24. Language box formatter (`lang-format`)

Type	`type(auto)` , `none` , or `function`
Default value	`auto`
Contextual function	✕ no
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.

2.24.1. Example

Example code Typst

```
*Default formatter:*
```py
print('Hello, world!')
```
Function mapping:
#codly(lang-format: (_, _, _) => [No!]))
```py
print('Hello, world!')
```
```

Rendered output

Default formatter:

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

Function mapping:)

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

2.25. Display language name (`display-name`)

Type	`bool`
Default value	`true`
Contextual function	✓ yes
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.

2.25.1. Example

Example code Typst

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

Rendered output

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

2.26. Display language icon (`display-icon`)

Type	bool
Default value	true
Contextual function	✓ yes
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.

2.26.1. Example

Example code (Typst)

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

Rendered output

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

2.27. Line number format (`number-format`)

Type	function or none
Default value	numbering.with("1")
Contextual function	✗ no
Automatically reset	no

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

2.27.1. Example

Example code (Typst)

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

Rendered output

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

2.28. Line number alignment (`number-align`)

Type	alignment
Default value	`left + horizon`
Contextual function	✓ yes
Automatically reset	no

The alignment of the numbers.

2.28.1. Example

Example code (Typst)

```typst
#codly(number-align: right + top)
```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
fib(25)
```
```

Rendered output

```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
fib(25)
```

2.29. Smart indentation (`smart-indent`)

Type	bool
Default value	true
Contextual function	✕ no
Automatically reset	no

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.

2.29.1. Example

Example code Typst

````
*Enabled (default):*
```py
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]])
  qsort(L)
```
*Disabled:*
#codly(smart-indent: false)
```py
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]])
  qsort(L))
```
````

Rendered output

Enabled (default):

```py
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]])
  qsort(L)
```

Disabled:

```py
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]])
  qsort(L))
```

2.30. Breakable (`breakable`)

Type	bool
Default value	true
Contextual function	✕ no
Automatically reset	no

Whether the codeblocks are breakable across page and column breaks.

2.31. Skips (`skips`)

Type	`array` or `none`
Default value	`none`
Contextual function	✓ yes
Automatically reset	✓ yes

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.

2.31.1. Example

Example code (Typst)

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

Rendered output

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

2.32. Skip line (`skip-line`)

Type	`content` or `none`
Default value	`align(center)[ ... ]`
Contextual function	✓ yes
Automatically reset	no

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

2.32.1. Example

Example code (Typst)

```
#codly(
  skips: ((4, 32), ),
  skip-line: align(center,
  emoji.face.shock)
)
```py
def fib(n):
  if n <= 1:
    return n
  return fib(n - 1) + fib(n - 2)
fib(25)
```
```

Rendered output

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

2.33. Skip number (`skip-number`)

Type	`content` or `none`
Default value	`[ ... ]`
Contextual function	✓ yes
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.

2.33.1. Example

Example code Typst

```
#codly(
  skips: ((4, 32), ),
  skip-number: align(center,
  emoji.face.shock)
)
```py
def fib(n):
  if n <= 1:
    return n
  return fib(n - 1) + fib(n - 2)
fib(25)
```
```

Rendered output

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

2.34. Annotations (`annotations`)

Type	`array` or `none`
Default value	`none`
Contextual function	✓ yes
Automatically reset	✓ yes

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: https://github.com/Dherse/codly.

2.34.1. Example

Example code (Typst)

```
#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)
fib(25)
```
```

Rendered output

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

(1) Function body

2.35. Annotation formatter (`annotation-format`)

Type	`none` or `function`
Default value	`numbering.with("(1)")`
Contextual function	✗ no
Automatically reset	no

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

2.36. Highlights (`highlights`)

Type	`array` or `none`
Default value	`none`
Contextual function	✓ yes
Automatically reset	no

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.

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.

2.36.1. Example

Example code (Typst)

```
#codly(highlights: (
  (line: 3, start: 3, end: none, fill: red),
  (line: 4, start: 12, end: 21, fill: green, tag: "(a)"),
  (line: 4, start: 28, 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))
```

Rendered output

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

2.37. Highlight radius (`highlight-radius`)

Type	`length`
Default value	`0.32em`
Contextual function	✓ yes
Automatically reset	✓ yes

The radius of the highlights.

2.38. Highlight fill (`highlight-fill`)

Type	`function`
Default value	`(color) => color.lighten(80%)`
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.

2.39. 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.

2.40. Highlight inset (`highlight-inset`)

Type	`length`
Default value	`0.32em`
Contextual function	✓ yes
Automatically reset	no

The inset of the highlights.

2.41. Reference by (`reference-by`)

Type	`str`
Default value	`"line"`
Contextual function	✓ 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.

2.42. Reference separator (`reference-sep`)

Type	`str` or `content`
Default value	`"-"`
Contextual function	✓ yes
Automatically reset	no

The separator to use when referencing highlights and annotations.

2.43. Reference number format (`reference-number-format`)

Type	`function`
Default value	`numbering.with("1")`
Contextual function	✗ no
Automatically reset	no

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

3. Getting nice icons

3.1. Typst language icon (`typst-icon`)

4. Other functions

4.1. Reset (`codly-reset`)

4.2. Skip (`codly-skip`)

4.3. Range (`codly-range`)

4.4. Offset (`codly-offset`)

4.5. Local (`local`)

4.6. No codly (`no-codly`)

4.7. Enable (`codly-enable`)

4.8. Disable (`codly-disable`)