🦓 Zebraw

Zebraw is a lightweight and fast package for displaying code blocks with line numbers in typst, supporting code line highlighting. The term ***zebraw*** is a combination of ***zebra*** and ***raw***, for the highlighted lines will be displayed in the code block like a zebra lines.

Starting

Import `zebraw` package by `#import "@preview/zebraw:0.4.3": *` then follow with `#show: zebraw` to start using zebraw in the simplest way. To manually display some specific code blocks in zebraw, you can use `#zebraw()` function:

````
```typ
#grid(
  columns: (1fr, 1fr),
  [Hello], [world!],
)
```

#zebraw(
  ```typ
  #grid(
    columns: (1fr, 1fr),
    [Hello], [world!],
  )
  ```
)

#show: zebraw

```typ
#grid(
  columns: (1fr, 1fr),
  [Hello], [world!],
)
```
````

```typ
#grid(
  columns: (1fr, 1fr),
  [Hello], [world!],
)
```

```
1 #grid(
2   columns: (1fr, 1fr),
3   [Hello], [world!],
4 )
```


```
1 #grid(
2   columns: (1fr, 1fr),
3   [Hello], [world!],
4 )
```


Features

Line Highlighting

You can highlight specific lines in the code block by passing the `highlight-lines` parameter to the `zebraw` function. The `highlight-lines` parameter can be a single line number or an array of line numbers.

````
#zebraw(
  // Single line number:
  highlight-lines: 2,
  ```typ
  #grid(
    columns: (1fr, 1fr),
    [Hello], [world!],
  )
  ```
)

#zebraw(
  // Array of line numbers:
  highlight-lines: (6, 7) + range(9,
15),
````

```
1 #grid(
2   columns: (1fr, 1fr),
3   [Hello], [world!],
4 )
```

```
1 = Fibonacci sequence
2 The Fibonacci sequence is defined
  through the
3 recurrence relation $F_n = F_(n-1)
  + F_(n-2)$.
4 It can also be expressed in
  _closed form:_
```

````
  ```typ
  = Fibonacci sequence
  The Fibonacci sequence is defined
through the
  recurrence relation $F_n = F_(n-1) +
F_(n-2)$.
  It can also be expressed in _closed
form:_

  $ F_n = round(1 / sqrt(5) phi.alt^n),
quad
    phi.alt = (1 + sqrt(5)) / 2 $

  #let count = 8
  #let nums = range(1, count + 1)
  #let fib(n) = (
    if n ⩽ 2 { 1 }
    else { fib(n - 1) + fib(n - 2) }
  )

  The first #count numbers of the
sequence are:

  #align(center, table(
    columns: count,
    ..nums.map(n ⇒ $F_#n$),
    ..nums.map(n ⇒ str(fib(n))),
  ))
  ```
)
````

```
 5
 6 $ F_n = round(1 / sqrt(5)
   phi.alt^n), quad
 7   phi.alt = (1 + sqrt(5)) / 2 $
 8
 9 #let count = 8
10 #let nums = range(1, count + 1)
11 #let fib(n) = (
12   if n ⩽ 2 { 1 }
13   else { fib(n - 1) + fib(n - 2) }
14 )
15
16 The first #count numbers of the
   sequence are:
17
18 #align(center, table(
19   columns: count,
20   ..nums.map(n ⇒ $F_#n$),
21   ..nums.map(n ⇒ str(fib(n))),
22 ))
```

Comment

You can add comments to the highlighted lines by passing an array of line numbers and comments to the `highlight-lines` parameter.

````
#zebraw(
  highlight-lines: (
    (1, [The Fibonacci sequence is
defined through the recurrence relation
$F_n = F_(n-1) + F_(n-2)$\
    It can also be expressed in _closed
form:_ $ F_n = round(1 / sqrt(5)
phi.alt^n), quad
    phi.alt = (1 + sqrt(5)) / 2 $]),
    // Passing a range of line numbers
in the array should begin with `..`
    ..range(9, 14),
    (13, [The first \#count numbers of
the sequence.]),
  ),
  ```typ
  = Fibonacci sequence
  #let count = 8
  #let nums = range(1, count + 1)
  #let fib(n) = (
    if n ⩽ 2 { 1 }
    else { fib(n - 1) + fib(n - 2) }
  )
````

```
 1 = Fibonacci sequence
   > The Fibonacci sequence is
   defined through the recurrence
   relation F_n = F_{n-1} + F_{n-2}
   It can also be expressed in closed
   form:

        F_n = ⌊ 1/√5 φ^n ⌉,   φ = (1+√5)/2

 2 #let count = 8
 3 #let nums = range(1, count + 1)
 4 #let fib(n) = (
 5   if n ⩽ 2 { 1 }
 6   else { fib(n - 1) + fib(n - 2) }
 7 )
 8
 9 #align(center, table(
10   columns: count,
11   ..nums.map(n ⇒ $F_#n$),
12   ..nums.map(n ⇒ str(fib(n))),
13 ))
```
```

```
  #align(center, table(
    columns: count,
    ..nums.map(n ⇒ $F_#n$),
    ..nums.map(n ⇒ str(fib(n))),
  ))
  ```
)
```

> The first #count numbers of the sequence.

Comments can begin with a flag, which is `">"` by default. You can change the flag by passing the `comment-flag` parameter to the `zebraw` function:

```
#zebraw(
  highlight-lines: (
    // Comments can only be passed when highlight-lines is an array, so at the end of the single element array, a comma is needed.
    (6, [The Fibonacci sequence is defined through the recurrence relation $F_n = F_(n-1) + F_(n-2)$]),
  ),
  comment-flag: "⇝",
  ```typ
  = Fibonacci sequence
  #let count = 8
  #let nums = range(1, count + 1)
  #let fib(n) = (
    if n ⩽ 2 { 1 }
    else { fib(n - 1) + fib(n - 2) }
  )

  #align(center, table(
    columns: count,
    ..nums.map(n ⇒ $F_#n$),
    ..nums.map(n ⇒ str(fib(n))),
  ))
  ```
)
```

```
 1 = Fibonacci sequence
 2 #let count = 8
 3 #let nums = range(1, count + 1)
 4 #let fib(n) = (
 5   if n ⩽ 2 { 1 }
 6   else { fib(n - 1) + fib(n - 2) }
     ⇝ The Fibonacci sequence is defined through the recurrence relation $F_n = F_{n-1} + F_{n-2}$
 7 )
 8
 9 #align(center, table(
10   columns: count,
11   ..nums.map(n ⇒ $F_#n$),
12   ..nums.map(n ⇒ str(fib(n))),
13 ))
```

To disable the flag feature, pass `""` to the `comment-flag` parameter (the indentation of the comment will be disabled as well):

```
#zebraw(
  highlight-lines: (
    (6, [The Fibonacci sequence is defined through the recurrence relation $F_n = F_(n-1) + F_(n-2)$]),
  ),
  comment-flag: "",
  ```typ
  = Fibonacci sequence
  #let count = 8
  #let nums = range(1, count + 1)
  #let fib(n) = (
    if n ⩽ 2 { 1 }
    else { fib(n - 1) + fib(n - 2) }
  )

  #align(center, table(
```

```
 1 = Fibonacci sequence
 2 #let count = 8
 3 #let nums = range(1, count + 1)
 4 #let fib(n) = (
 5   if n ⩽ 2 { 1 }
 6   else { fib(n - 1) + fib(n - 2) }
   The Fibonacci sequence is defined through the recurrence relation $F_n = F_{n-1} + F_{n-2}$
 7 )
 8
 9 #align(center, table(
10   columns: count,
11   ..nums.map(n ⇒ $F_#n$),
12   ..nums.map(n ⇒ str(fib(n))),
```

````
    columns: count,
    ..nums.map(n => $F_#n$),
    ..nums.map(n => str(fib(n))),
  ))
  ```
)
````

```
13 ))
```

Header and Footer

Usually, the comments passing by a dictionary of line numbers and comments are used to add a header or footer to the code block:

````
#zebraw(
  highlight-lines: (
    (header: [*Fibonacci sequence*]),
    ..range(8, 13),
    // Numbers can be passed as a string
in the dictionary, but it's too ugly.
    ("12": [The first \#count numbers of
the sequence.]),
    (footer: [The fibonacci sequence is
defined through the recurrence relation
$F_n = F_(n-1) + F_(n-2)$]),
  ),
  ```typ
  #let count = 8
  #let nums = range(1, count + 1)
  #let fib(n) = (
    if n <= 2 { 1 }
    else { fib(n - 1) + fib(n - 2) }
  )

  #align(center, table(
    columns: count,
    ..nums.map(n => $F_#n$),
    ..nums.map(n => str(fib(n))),
  ))
  ```
)
````

Fibonacci sequence

```
1 #let count = 8
2 #let nums = range(1, count + 1)
3 #let fib(n) = (
4   if n <= 2 { 1 }
5   else { fib(n - 1) + fib(n - 2) }
6 )
7
8 #align(center, table(
9   columns: count,
10  ..nums.map(n => $F_#n$),
11  ..nums.map(n => str(fib(n))),
12 ))
```

> The first #count numbers of the sequence.

The fibonacci sequence is defined through the recurrence relation $F_n = F_{n-1} + F_{n-2}$

Or you can use `header` and `footer` parameters to add a header or footer to the code block:

````
#zebraw(
  highlight-lines: (
    ..range(8, 13),
    (12, [The first \#count numbers of
the sequence.]),
  ),
  header: [*Fibonacci sequence*],
  ```typ
  #let count = 8
  #let nums = range(1, count + 1)
  #let fib(n) = (
    if n <= 2 { 1 }
    else { fib(n - 1) + fib(n - 2) }
  )

  #align(center, table(
    columns: count,
    ..nums.map(n => $F_#n$),
````

Fibonacci sequence

```
1 #let count = 8
2 #let nums = range(1, count + 1)
3 #let fib(n) = (
4   if n <= 2 { 1 }
5   else { fib(n - 1) + fib(n - 2) }
6 )
7
8 #align(center, table(
9   columns: count,
10  ..nums.map(n => $F_#n$),
11  ..nums.map(n => str(fib(n))),
12 ))
```

> The first #count numbers of the sequence.

````
    ..nums.map(n => str(fib(n))),
  ))
  ```
    ,
  footer: [The fibonacci sequence is
defined through the recurrence relation
$F_n = F_(n-1) + F_(n-2)$],
)
````

The fibonacci sequence is defined through the recurrence relation $F_n = F_{n-1} + F_{n-2}$

Language Tab

If `lang` is set to `true`, then there will be a language tab on the top right corner of the code block:

````
#zebraw(
  lang: true,
  ```typst
  #grid(
    columns: (1fr, 1fr),
    [Hello], [world!],
  )
  ```
)
````

```typst
1 #grid(
2   columns: (1fr, 1fr),
3   [Hello], [world!],
4 )
```

Customize the language to display by pass a string or content to the `lang` parameter.

````
#zebraw(
  lang: strong[Typst],
  ```typst
  #grid(
    columns: (1fr, 1fr),
    [Hello], [world!],
  )
  ```
)
````

```typst
1 #grid(
2   columns: (1fr, 1fr),
3   [Hello], [world!],
4 )
```

Copyable

Line numbers will not be selected when selecting exported code in one page.

`copyable: false`

```
1 #grid(
2   columns: (1fr, 1fr),
3   [Hello], [world!],
4 )
```

`copyable: true`

```
1 #grid(
2   columns: (1fr, 1fr),
3   [Hello], [world!],
4 )
```

Theme

PRs are welcome!

````
#show: zebraw-init.with(..zebraw-
themes.zebra, lang: false)
#show: zebraw

```rust
pub fn fibonacci_reccursive(n: i32) ->
u64 {
    if n < 0 {
        panic!("{} is negative!", n);
    }
    match n {
````

```rust
1 pub fn fibonacci_reccursive(n:
  i32) -> u64 {
2     if n < 0 {
3         panic!("{} is negative!",
  n);
4     }
5     match n {
```

````
        0 => panic!("zero is not a right
argument to fibonacci_reccursive()!"),
        1 | 2 => 1,
        3 => 2,
        _ => fibonacci_reccursive(n - 1)
+ fibonacci_reccursive(n - 2),
    }
}
```
````

```
 6          0 => panic!("zero is not a
   right argument to
   fibonacci_reccursive()!"),
 7          1 | 2 => 1,
 8          3 => 2,
 9          _ =>
   fibonacci_reccursive(n - 1) +
   fibonacci_reccursive(n - 2),
10      }
11  }
```

````
#show: zebraw-init.with(..zebraw-
themes.zebra-reverse, lang: false)
#show: zebraw

```rust
pub fn fibonacci_reccursive(n: i32) ->
u64 {
    if n < 0 {
        panic!("{} is negative!", n);
    }
    match n {
        0 => panic!("zero is not a right
argument to fibonacci_reccursive()!"),
        1 | 2 => 1,
        3 => 2,
        _ => fibonacci_reccursive(n - 1)
+ fibonacci_reccursive(n - 2),
    }
}
```
````

```
 1  pub fn fibonacci_reccursive(n:
    i32) -> u64 {
 2      if n < 0 {
 3          panic!("{} is negative!",
    n);
 4      }
 5      match n {
 6          0 => panic!("zero is not a
    right argument to
    fibonacci_reccursive()!"),
 7          1 | 2 => 1,
 8          3 => 2,
 9          _ =>
    fibonacci_reccursive(n - 1) +
    fibonacci_reccursive(n - 2),
10      }
11  }
```

(Experimental) HTML Variant

See example-html.typ or GitHub Pages for more information.


```

Customization

There are 3 ways to customize code blocks in your document:

- Manually render some specific blocks by `#zebraw()` function and passing parameters to it.
- By passing parameters to `#show: zebraw.with()` will affect every raw block after the `#show` rule, **except** blocks created manually by `#zebraw()` function.
- By passing parameters to `#show: zebraw-init.with()` will affect every raw block after the `#show` rule, **including** blocks created manually by `#zebraw()` function. By using `zebraw-init` without any parameters, the values will be reset to default.

Inset

Customize the inset of each line by passing a `dictionary` to the `inset` parameter:

````
#zebraw(
  inset: (top: 6pt, bottom: 6pt),
  ```typ
  #grid(
    columns: (1fr, 1fr),
    [Hello], [world!],
  )
  ```
)
````

```
1 #grid(
2   columns: (1fr, 1fr),
3   [Hello], [world!],
4 )
```

Colors

Customize the background color by passing a `color` or an `array` of `colors` to the `background-color` parameter.

````
#zebraw(
  background-color: luma(235),
  ```typ
  #grid(
    columns: (1fr, 1fr),
    [Hello], [world!],
  )
  ```,
)

#zebraw(
  background-color: (luma(235),
luma(245), luma(255), luma(245)),
  ```typ
  #grid(
    columns: (1fr, 1fr),
    [Hello], [world!],
  )
  ```,
)
````

```
1 #grid(
2   columns: (1fr, 1fr),
3   [Hello], [world!],
4 )
```

```
1 #grid(
2   columns: (1fr, 1fr),
3   [Hello], [world!],
4 )
```

Customize the highlight color by passing a `color` to the `highlight-color` parameter:

````
#zebraw(
  highlight-lines: 1,
  highlight-color: blue.lighten(90%),
  ```text
  I'm so blue!
               -- George III
````

```
1 I'm so blue!
2              -- George III
```
```

````
  ```
    ,
)
````

Customize the comments' background color by passing a `color` to the `comment-color` parameter:

````
#zebraw(
  highlight-lines: (
    (2, "auto indent!"),
  ),
  comment-color: yellow.lighten(90%),
  ```text
  I'm so blue!
               -- George III
  I'm not.
               -- Hamilton
  ```
    ,
)
````

```
1 I'm so blue!
2              -- George III
               > auto indent!
3 I'm not.
4              -- Hamilton
```

Customize the language tab's background color by passing a `color` to the `lang-color` parameter.

````
#zebraw(
  lang: true,
  lang-color: teal,
  ```typst
  #grid(
    columns: (1fr, 1fr),
    [Hello], [world!],
  )
  ```
)
````

typst

```
1 #grid(
2   columns: (1fr, 1fr),
3   [Hello], [world!],
4 )
```

Font

To customize the arguments of comments' font and the language tab's font, pass a dictionary to `comment-font-args` parameter and `lang-font-args` parameter.

Language tab will be rendered as comments if nothing is passed.

````
#zebraw(
  highlight-lines: (
    (2, "columns ... "),
  ),
  lang: true,
  comment-color: white,
  comment-font-args: (
    font: "IBM Plex Sans",
    style: "italic"
  ),
  ```typst
  #grid(
    columns: (1fr, 1fr),
    [Hello], [world!],
  )
  ```
)
````

typst

```
1 #grid(
2   columns: (1fr, 1fr),
    > columns...
3   [Hello], [world!],
4 )
```

````
#zebraw(
  highlight-lines: (
    (2, "columns ... "),
````

typst

```
1 #grid(
```
```

```
  ),
  lang: true,
  lang-color: eastern,
  lang-font-args: (
    font: "libertinus serif",
    weight: "bold",
    fill: white,
  ),
  comment-font-args: (
    font: "IBM Plex Sans",
    style: "italic"
  ),
  ```typst
  #grid(
    columns: (1fr, 1fr),
    [Hello], [world!],
  )
  ```
)
```

```
2   columns: (1fr, 1fr),
    > columns...
3   [Hello], [world!],
4 )
```

Extend

Extend at vertical is enabled at default. When there's header or footer it will be automatically disabled.

```
#zebraw(
  extend: false,
  ```typst
  #grid(
    columns: (1fr, 1fr),
    [Hello], [world!],
  )
  ```
)
```

```
1 #grid(
2   columns: (1fr, 1fr),
3   [Hello], [world!],
4 )
```

Documentation

The default value of most parameters are `none` for it will use the default value in `zebraw-init`.

- zebraw-init()
- zebraw()

zebraw-init

Initialize the `zebraw` block in global.

Parameters

```
zebraw-init(
  inset: dictionary,
  background-color: color array,
  highlight-color: color,
  comment-color: color,
  lang-color: color,
  comment-flag: string content,
  lang: boolean string content,
  comment-font-args: dictionary,
  lang-font-args: dictionary,
  extend: boolean,
  body: content
) -> content
```

inset `dictionary`

The inset of each line.

Default: `(top: 0.34em, right: 0.34em, bottom: 0.34em, left: 0.34em)`

background-color `color` or `array`

The background color of the block and normal lines.

Default: `luma(245)`

highlight-color `color`

The background color of the highlighted lines.

Default: `rgb("#94e2d5").lighten(70%)`

comment-color `color`

The background color of the comments. When it's set to `none`, it will be rendered in a lightened `highlight-color`.

Default: `none`

lang-color `color`

The background color of the language tab. The color is set to `none` at default and it will be rendered in comments' color.

Default: `none`

comment-flag `string` or `content`

The flag at the beginning of comments.

Default: `">"`

lang `boolean` or `string` or `content`

Whether to show the language tab, or a string or content of custom language name to display.

Default: `true`

comment-font-args `dictionary`

The arguments passed to comments' font.

Default: `(:)`

lang-font-args `dictionary`

The arguments passed to the language tab's font.

Default: `(:)`

extend `boolean`

Whether to extend the vertical spacing.

Default: `true`

body `content`

The body

zebraw

Block of code with highlighted lines and comments.

Parameters

```
zebraw(
  highlight-lines: array int,
  header: string content,
  footer: string content,
  inset: dictionary,
  background-color: color array,
  highlight-color: color,
  comment-color: color,
  lang-color: color,
  comment-flag: string content,
  lang: boolean string,
  comment-font-args: dictionary,
  lang-font-args: dictionary,
  extend: boolean,
  body: content
) -> content
```

highlight-lines `array` or `int`

Lines to highlight or comments to show.

```
#zebraw(
  highlight-lines: range(3, 7),
  header: [*Fibonacci sequence*],
  ```typst
  #let count = 8
  #let nums = range(1, count + 1)
  #let fib(n) = (
    if n ⩽ 2 { 1 }
    else { fib(n - 1) + fib(n - 2) }
  )

  #align(center, table(
    columns: count,
    ..nums.map(n ⇒ $F_#n$),
    ..nums.map(n ⇒ str(fib(n))),
  ))
  ```,
  footer: [The fibonacci sequence is
defined through the recurrence relation
$F_n = F_(n-1) + F_(n-2)$],
)
```

Fibonacci sequence

```typst
#let count = 8
#let nums = range(1, count + 1)
#let fib(n) = (
  if n ⩽ 2 { 1 }
  else { fib(n - 1) + fib(n - 2) }
)

#align(center, table(
  columns: count,
  ..nums.map(n ⇒ $F_#n$),
  ..nums.map(n ⇒ str(fib(n))),
))
```

The fibonacci sequence is defined through the recurrence relation $F_n = F_{n-1} + F_{n-2}$

Default: ( )

header `string` or `content`

The header of the code block.

Default: none

footer `string` or `content`

The footer of the code block.

Default: none

inset `dictionary`

The inset of each line.

````
#zebraw(
  inset: (top: 6pt, bottom: 6pt),
  ```typst
  #let count = 8
  #let nums = range(1, count + 1)
  #let fib(n) = (
    if n ⩽ 2 { 1 }
    else { fib(n - 1) + fib(n - 2) }
  )

  #align(center, table(
    columns: count,
    ..nums.map(n ⇒ $F_#n$),
    ..nums.map(n ⇒ str(fib(n))),
  ))
  ```,
)
````

```typst
#let count = 8
#let nums = range(1, count + 1)
#let fib(n) = (
  if n ⩽ 2 { 1 }
  else { fib(n - 1) + fib(n - 2) }
)

#align(center, table(
  columns: count,
  ..nums.map(n ⇒ $F_#n$),
  ..nums.map(n ⇒ str(fib(n))),
))
```

Default: none

background-color `color` or `array`

The background color of the block and normal lines.

````
#zebraw(
  background-color: (luma(240), luma(245), luma(250), luma(245)),
  ```typst
  #let count = 8
  #let nums = range(1, count + 1)
  #let fib(n) = (
    if n ⩽ 2 { 1 }
    else { fib(n - 1) + fib(n - 2) }
  )

  #align(center, table(
    columns: count,
    ..nums.map(n ⇒ $F_#n$),
    ..nums.map(n ⇒ str(fib(n))),
  ))
  ```,
)
````

```typst
#let count = 8
#let nums = range(1, count + 1)
#let fib(n) = (
  if n ⩽ 2 { 1 }
  else { fib(n - 1) + fib(n - 2) }
)

#align(center, table(
  columns: count,
  ..nums.map(n ⇒ $F_#n$),
  ..nums.map(n ⇒ str(fib(n))),
))
```

Default: none

highlight-color `color`

The background color of the highlighted lines.

Default: none

comment-color `color`

The background color of the comments. The color is set to none at default and it will be rendered in a lightened highlight-color.

````
#zebraw(
  highlight-color: yellow.lighten(80%),
  comment-color: yellow.lighten(90%),
  highlight-lines: (
    (1, [The Fibonacci sequence is
defined through the recurrence relation
$F_n = F_(n-1) + F_(n-2)$]),
    ..range(9, 14),
    (13, [The first \#count numbers of
the sequence.]),
  ),
  ```typ
  = Fibonacci sequence
  #let count = 8
  #let nums = range(1, count + 1)
  #let fib(n) = (
    if n <= 2 { 1 }
    else { fib(n - 1) + fib(n - 2) }
  )

  #align(center, table(
    columns: count,
    ..nums.map(n => $F_#n$),
    ..nums.map(n => str(fib(n))),
  ))
  ```
)
````

```
1  = Fibonacci sequence
   > The Fibonacci sequence is
   defined through the recurrence
   relation F_n = F_(n-1) + F_(n-2)
2  #let count = 8
3  #let nums = range(1, count + 1)
4  #let fib(n) = (
5    if n <= 2 { 1 }
6    else { fib(n - 1) + fib(n -
   2) }
7  )
8
9  #align(center, table(
10   columns: count,
11   ..nums.map(n => $F_#n$),
12   ..nums.map(n => str(fib(n))),
13 ))
   > The first #count numbers of
   the sequence.
```

Default: none

lang-color `color`

The background color of the language tab. The color is set to none at default and it will be rendered in comments' color.

````
#zebraw(
  lang: true,
  lang-color: eastern,
  lang-font-args: (
    font: "libertinus serif",
    weight: "bold",
    fill: white
  ),
  ```typst
  #grid(
    columns: (1fr, 1fr),
    [Hello], [world!],
  )
  ```
)
````

typst

```
1 #grid(
2   columns: (1fr, 1fr),
3   [Hello], [world!],
4 )
```

Default: none

comment-flag `string` or `content`

The flag at the beginning of comments. The indentation of codes will be rendered before the flag. When the flag is set to "", the indentation before the flag will be disabled as well.

````
#zebraw(
  comment-flag: "",
  highlight-lines: (
    (1, [The Fibonacci sequence is
defined through the recurrence relation
$F_n = F_(n-1) + F_(n-2)$]),
    ..range(9, 14),
    (13, [The first \#count numbers of
the sequence.]),
  ),
  ```typ
  = Fibonacci sequence
  #let count = 8
  #let nums = range(1, count + 1)
  #let fib(n) = (
    if n ⩽ 2 { 1 }
    else { fib(n - 1) + fib(n - 2) }
  )

  #align(center, table(
    columns: count,
    ..nums.map(n ⇒ $F_#n$),
    ..nums.map(n ⇒ str(fib(n))),
  ))
  ```
)
````

```
1 = Fibonacci sequence
   The Fibonacci sequence is
  defined through the recurrence
  relation F_n = F_(n-1) + F_(n-2)
2 #let count = 8
3 #let nums = range(1, count + 1)
4 #let fib(n) = (
5   if n ⩽ 2 { 1 }
6   else { fib(n - 1) + fib(n -
  2) }
7 )
8
9 #align(center, table(
10   columns: count,
11   ..nums.map(n ⇒ $F_#n$),
12   ..nums.map(n ⇒ str(fib(n))),
13 ))
   The first #count numbers of
  the sequence.
```

Default: none

lang `boolean` or `string`

Whether to show the language tab, or a string or content of custom language name to display.

````
#zebraw(
  lang: true,
  ```typ
  #grid(
    columns: (1fr, 1fr),
    [Hello,], [world!],
  )
  ```
)
````

typ

```typ
#grid(
  columns: (1fr, 1fr),
  [Hello,], [world!],
)
```

````
#zebraw(
  lang: strong[Typst],
  ```typ
  #grid(
    columns: (1fr, 1fr),
    [Hello,], [world!],
  )
  ```
)
````

Typst

```typ
#grid(
  columns: (1fr, 1fr),
  [Hello,], [world!],
)
```

Default: `none`

comment-font-args `dictionary`

The arguments passed to comments' font.

Default: `none`

lang-font-args `dictionary`

The arguments passed to the language tab's font.

````
#zebraw(
  lang: true,
  comment-font-args: (font: "IBM Plex
Serif", style: "italic"),
  lang-font-args: (font: "IBM Plex Sans",
weight: "bold"),
  highlight-lines: (
    (1, [The Fibonacci sequence is
defined through the recurrence relation
$F_n = F_(n-1) + F_(n-2)$]),
    ..range(9, 14),
    (13, [The first \#count numbers of
the sequence.]),
  ),
  ```typ
  = Fibonacci sequence
  #let count = 8
  #let nums = range(1, count + 1)
  #let fib(n) = (
    if n ⩽ 2 { 1 }
    else { fib(n - 1) + fib(n - 2) }
  )

  #align(center, table(
    columns: count,
    ..nums.map(n ⇒ $F_#n$),
    ..nums.map(n ⇒ str(fib(n))),
  ))
  ```
)
````

typ

```typ
= Fibonacci sequence
> The Fibonacci sequence is defined through the recurrence relation $F_n = F_{n-1} + F_{n-2}$
#let count = 8
#let nums = range(1, count + 1)
#let fib(n) = (
  if n ⩽ 2 { 1 }
  else { fib(n - 1) + fib(n - 2) }
)

#align(center, table(
  columns: count,
  ..nums.map(n ⇒ $F_#n$),
  ..nums.map(n ⇒ str(fib(n))),
))
> The first #count numbers of the sequence.
```

Default: `none`

extend `boolean`

Whether to extend the vertical spacing.

````
#zebraw(
  extend: false,
  ```typ
  #grid(
    columns: (1fr, 1fr),
    [Hello,], [world!],
  )
  ```
)
````

```typ
#grid(
  columns: (1fr, 1fr),
  [Hello,], [world!],
)
```

Default: `none`

body `content`

The body.

Example

rust

Calculate Fibonacci number using reccursive function

```rust
pub fn fibonacci_reccursive(n: i32) -> u64 {
    if n < 0 {
        panic!("{} is negative!", n);
        > to avoid negative numbers
    }
    match n {
        0 => panic!("zero is not a right argument to fibonacci_reccursive()!"),
        1 | 2 => 1,
        3 => 2,
        _ => fibonacci_reccursive(n - 1) + fibonacci_reccursive(n - 2),
        > 50 => 12586269025
    }
}
```