

Zebraw

Zebraw is a lightweight and fast package for displaying code blocks with line numbers in Typst, supporting code line highlighting. The term **zebra** is a combination of **zebra** and **raw**, as the highlighted lines display in the code block with a zebra-striped pattern.

Quick Start

Import the zebraw package with `#import "@preview/zebra:0.5.0": *` then add `#show: zebraw` to start using zebraw in the simplest way.

```
#import "@preview/zebra:0.5.0": *
#show: zebraw

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

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

To manually render specific code blocks with zebraw, use the `#zebra()` function:

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

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

Features

Line Numbering

Line numbers appear on the left side of the code block. Change the starting line number by passing an integer to the `numbering-offset` parameter. The default value is `0`.

```
#zebra(
  // The first line number will be 2.
  numbering-offset: 1,
  ```typ
 #grid(
 columns: (1fr, 1fr),
 [Hello], [world!],
)
  ```
)
```

```
2 #grid(
3   columns: (1fr, 1fr),
4   [Hello], [world!],
5 )
```

To disable line numbering, pass `false` to the `numbering` parameter:

```
#zebra(
  numbering: false,
  ```typ
 #grid(
 columns: (1fr, 1fr),

```

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

```
[Hello], [world!],
)
...
)
```

## Numbering Separator

You can add a separator line between line numbers and code content by setting the numbering-separator parameter to true:

```
#zebraw(
 numbering-separator: true,
 ``typ
 #grid(
 columns: (1fr, 1fr),
 [Hello], [world!],
)
 ...
)
```

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

## Line Slicing

Slice code blocks by passing the line-range parameter to the zebraw function. The line-range parameter can be either:

- An array of 2 integers representing the range  $[a, b)$  ( $b$  can be none as this feature is based on Typst array slicing)
- A dictionary with range and keep-offset keys

When keep-offset is set to true, line numbers maintain their original values. Otherwise, they reset to start from 1. By default, keep-offset is set to true.

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

#zebraw(code)

#zebraw(line-range: (2, 4), code)

#zebraw(
 line-range: (range: (2, 4), keep-
offset: false),
 code
)

#zebraw(
 numbering-offset: 30,
 line-range: (range: (2, 4), keep-
offset: false),
 code
)

#zebraw(
 numbering-offset: 30,
```

```
1 #grid(
2 columns: (1fr, 1fr),
3 [Hello],
4 [world!],
5)
```

```
2 columns: (1fr, 1fr),
3 [Hello],
```

```
1 columns: (1fr, 1fr),
2 [Hello],
```

```
31 columns: (1fr, 1fr),
32 [Hello],
```

```
32 columns: (1fr, 1fr),
33 [Hello],
```

```
 line-range: (range: (2, 4), keep-
offset: true),
 code
)
```

## Line Highlighting

Highlight specific lines in the code block by passing the `highlight-lines` parameter to the `zebraw` function. The `highlight-lines` parameter accepts either 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),
 ``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))),
))
 ...
)
```

```
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:
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))
```

## Comments

Add explanatory comments to 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 = \text{round}(1 / \text{sqrt}(5)
\phi.\text{alt}^n), \text{quad}
\phi.\text{alt} = (1 + \text{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) }
)

#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  $F_n = F_{n-1} + F_{n-2}$ . It can also be expressed in closed form:

$$F_n = \left\lfloor \frac{1}{\sqrt{5}} \phi^n \right\rfloor, \quad \phi = \frac{1 + \sqrt{5}}{2}$$

```
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))
> The first #count numbers of the
sequence.
```

Comments begin with a flag character, which is ">" by default. Change this flag by setting the `comment-flag` parameter:

```
#zebraw(
 highlight-lines: (
 // Comments can only be passed when
highlight-lines is an array, so a comma
is needed at the end of a single-element
array
 (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) }
)
)
```

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

```

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

```

To disable the flag feature entirely, pass an empty string "" to the comment-flag parameter (this also disables comment indentation):

```

#zebrab(
 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(
 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) }
7)
8
9 #align(center, table(
10 columns: count,
11 .. nums.map(n => $F_#n$),
12 .. nums.map(n => str(fib(n))),
13))

```

The Fibonacci sequence is defined through the recurrence relation

$$F_n = F_{n-1} + F_{n-2}$$

## Headers and Footers

You can add headers and footers to code blocks. One approach is to use special keys in the highlight-lines parameter:

```

#zebrab(
 highlight-lines: (
 (header: [*Fibonacci sequence*]),
 .. range(8, 13),
 // Numbers can be passed as strings
 in the dictionary, though this approach
 is less elegant
 ("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) }
)

```

### 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.

```

)

#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  $F_n = F_{n-1} + F_{n-2}$

Alternatively, use the dedicated header and footer parameters for cleaner code:

```

#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 => $F_#n$),
 ..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

```

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  $F_n = F_{n-1} + F_{n-2}$

## Language Tab

Display a floating language identifier tab in the top-right corner of the code block by setting `lang` to `true`:

```

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

```

```

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

```

typst

Customize the language display by passing a string or content to the `lang` parameter:

```

#zebraw(
 lang: strong[Typst],
 ``typst

```

```

1 #grid(
2 columns: (1fr, 1fr),

```

Typst

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

```
3 [Hello], [world!],
4)
```

## Indentation Lines, Hanging Indentation and Fast Preview

Display indentation guides by passing a positive integer to the indentation parameter, representing the number of spaces per indentation level:

```
#zebraw(
 indentation: 2,
 ``typ
#let forecast(day) = block[
 #box(square(
 width: 2cm,
 inset: 8pt,
 fill: if day.weather == "sunny" {
 yellow
 } else {
 aqua
 },
 align(
 bottom + right,
 strong(day.weather),
),
))
#h(6pt)
#set text(22pt, baseline: -8pt)
#day.temperature °#day.unit
]
...
)
```

```
1 #let forecast(day) = block[
2 #box(square(
3 width: 2cm,
4 inset: 8pt,
5 fill: if day.weather ==
6 "sunny" {
7 yellow
8 } else {
9 aqua
10 },
11 align(
12 bottom + right,
13 strong(day.weather),
14),
15))
16 #h(6pt)
17 #set text(22pt, baseline: -8pt)
18 #day.temperature °#day.unit
19]
```

Enable hanging indentation by setting hanging-indent to true:

```
#zebraw(
 hanging-indent: true,
 ``typ
#let forecast(day) = block[
 #box(square(
 width: 2cm,
 inset: 8pt,
 fill: if day.weather == "sunny" {
 yellow
 } else {
 aqua
 },
 align(
 bottom + right,
 strong(day.weather),
),
))
#h(6pt)
#set text(22pt, baseline: -8pt)
#day.temperature °#day.unit
]
```

```
1 #let forecast(day) = block[
2 #box(square(
3 width: 2cm,
4 inset: 8pt,
5 fill: if day.weather ==
6 "sunny" {
7 yellow
8 } else {
9 aqua
10 },
11 align(
12 bottom + right,
13 strong(day.weather),
14),
15))
16 #h(6pt)
17 #set text(22pt, baseline: -8pt)
18 #day.temperature °#day.unit
19]
```

```
 ...
)
```

Indentation lines can slow down preview performance. For faster previews, enable fast preview mode by passing `true` to the `fast-preview` parameter in `zebraw-init` or by using `zebraw-fast-preview` in the CLI. This renders indentation lines as simple `|` characters:

```
#zebraw(
 hanging-indent: true,
 ``typ
#let forecast(day) = block[
 #box(square(
 width: 2cm,
 inset: 8pt,
 fill: if day.weather == "sunny" {
 yellow
 } else {
 aqua
 },
 align(
 bottom + right,
 strong(day.weather),
),
))
#h(6pt)
#set text(22pt, baseline: -8pt)
#day.temperature °#day.unit
]
...
)
```

```
1 #let forecast(day) = block[
2 | #box(square(
3 | | width: 2cm,
4 | | inset: 8pt,
5 | | fill: if day.weather ==
6 | | | yellow
7 | | } else {
8 | | | aqua
9 | | },
10 | | align(
11 | | | bottom + right,
12 | | | strong(day.weather),
13 | |),
14 |))
15 #h(6pt)
16 #set text(22pt, baseline: -8pt)
17 #day.temperature °#day.unit
18]
```

## Themes

Zebraw includes built-in themes. PRs for additional themes are welcome!

```
#show: zebraw.with(..zebraw-
themes.zebra)

``rust
pub fn fibonacci_reccursive(n: i32) →
u64 {
 if n < 0 {
 panic!("{}", 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!("{}", 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 }
```

```
#show: zebraw.with(..zebraw-
themes.zebra-reverse)
```

```
1 pub fn fibonacci_reccursive(n:
i32) → u64 {
```



```

rust
pub fn fibonacci_reccursive(n: i32) →
u64 {
 if n < 0 {
 panic!("{}", 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),
 }
}

```

```

2 | if n < 0 {
3 | panic!("{}", 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 Export

See [example-html.type](#) or [GitHub Pages](#) for more information.

## Customization

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

1. **Per-block customization:** Manually style specific blocks using the `#zebraw()` function with parameters.
2. **Local customization:** Apply styling to all subsequent raw blocks with `#show: zebraw.with()`. This affects all raw blocks after the `#show` rule, **except** those created manually with `#zebraw()`.
3. **Global customization:** Use `#show: zebraw-init.with()` to affect **all** raw blocks after the rule, **including** those created manually with `#zebraw()`. Reset to defaults by using `zebraw-init` without parameters.

## Inset

Customize the padding around each code line (numberings are not affected) 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 with a single color or an array of alternating colors:

```

#zebraw(
 background-color: luma(250),
 ``typ
 #grid(
 columns: (1fr, 1fr),
 [Hello], [world!],
)
)

```

```

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

```

```

)
 ...
)

#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)

```

Set the highlight color for marked lines with 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

```

Change the comment background color with 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
3 | | | | | > auto indent!
4 | | | | | -- Hamilton

```

Set the language tab background color with the `lang-color` parameter:

```

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

```

```

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

```

typst

## Font

Customize font properties for comments, language tabs, and line numbers by passing a dictionary to the `comment-font-args`, `lang-font-args`, or `numbering-font-args` parameters respectively.

If no custom lang-font-args are provided, language tabs inherit the comment font styling:

```
#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!],
)
 ...
)
```

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

Example with custom language tab styling:

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

```
1 #grid(
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 |)
```

## Example

rust

Calculate Fibonacci number using recursive function

```
1 pub fn fibonacci_reccursive(n: i32) → u64 {
2 if n < 0 {
3 panic!("{}", n);
4 > to avoid negative numbers
5 }
6 match n {
7 0 ⇒ panic!("zero is not a right argument to fibonacci_reccursive()!"),
8 1 | 2 ⇒ 1,
9 3 ⇒ 2,
10 _ ⇒ fibonacci_reccursive(n - 1) + fibonacci_reccursive(n - 2),
11 > 50 ⇒ 12586269025
12 }
13 }
```