

Normal `#raw()` works as expected:

```
/*
 * Example taken from
 * https://typst.app/docs/tutorial/formatting/
 */
#show "ArtosFlow": name => box[
#box(image(
"logo.svg",
height: 0.7em,
))
#name
]

// Long line that breaks
This report is embedded in the ArtosFlow project. ArtosFlow is a project of the
Artos Institute.

/*
Very long line without linebreak
with a preceding block comment
*/
This_report_is_embedded_in_the_ArtosFlow_project._ArtosFlow_is_a_project_of_the_Artos_Institute.

// End example
```

Using `#sourcecode()` will add line numbers and a frame.

```
1  /*
2   * Example taken from
3   * https://typst.app/docs/tutorial/formatting/
4   */
5  #show "ArtosFlow": name => box[
6  #box(image(
7  "logo.svg",
8  height: 0.7em,
9  ))
10 #name
11 ]
12
13 // Long line that breaks
14 This report is embedded in the ArtosFlow project. ArtosFlow is a project of
15 the Artos Institute.
16
17 /*
18 Very long line without linebreak
19 with a preceding block comment
20 */
21 This_report_is_embedded_in_the_ArtosFlow_project._ArtosFlow_is_a_project_of_the_Artos_Institut
22 // End example
```

```

15
16  /*
17   Very long line without linebreak
18   with a preceding block comment
19  */
20  This_report_is_embedded_in_the_ArtosFlow_project._ArtosFlow_is_a_project_of_the_Artos_Institut
21

```

Sourcecode can be loaded from a file and passed to `#sourcefile()`. Any **CODELST** sourcecode can be wrapped inside `#figure()` as expected.

CODELST line numbers can be formatted via a bunch of numbers - options:

To the right in Listing 1 you can see the `typst.toml` file of this package with some *fancy line numbers*.

```

[package] 1
name = "codelst" 2
version = "2.0.0" 3
entrypoint = "codelst.typ" 4
authors = ["Jonas Neugebauer"] 5
license = "MIT" 6
description = "A typst package to render 7
sourcecode." 7
repository = "https://github.com/jneug/typst- 8
codelst" 8
exclude = ["example.typ", "example.pdf", 9
"manual.pdf", "manual.typ", "tbump.toml"] 9
compiler = "0.0.9" 10
keywords = ["sourcecode", "code", "syntax- 11
highlighting", "raw", "line numbers"] 11

```

Listing 1: `typst.toml`

Since packages can't `#read()` files from outside their own directory, you can alias `#sourcefile()` for a more convenient command:

```
let srcfile( filename, ..args ) = sourcefile(read(filename), file:filename, ..args)
```

Formatting is controlled through options. To use a default style, create an alias for your command:

```
let code = sourcecode.with(
numbers-style: (lno) => text(black, lno),
frame: none
)
```

`#sourcecode()` accepts a number of arguments to affect the output like *highlighting lines*, *restrict the line range* or *place labels* in specific lines to reference them later.

```

9   #"hello world!" \
10  #"\"hello\n world\"!" \
11  #"1 2 3".split() \
12  #"1,2;3".split(regex("[,;]")) \
13  #(regex("\\d+") in "ten euros") \
14  #(regex("\\d+") in "10 euros")

```

To reference a line use `#lineref()`:

- See line 11 for an example of the `split()` function.

Long code breaks to new pages. To have listings in figures break, you need to allow it via a `#show()` rule:

```
#show figure.where(kind: raw): set block(breakable: true)
```

Listing 2: Code of this example file.

```
#import "./code1st.typ": sourcecode, sourcefile, lineref, code-frame
#let code1st = text(fill: rgb(254,48,147), smallcaps("code1st"))
#let cmd( name ) = text(fill: rgb(99, 170, 234), raw(block:false, sym.hash +
name.text + sym.paren.l + sym.paren.r))
5
#let code-block = block.with(
stroke: 1pt,
inset: 0.65em,
radius: 4pt
10
)
#let code-example = ```typ
/*
 * Example taken from
15 * https://typst.app/docs/tutorial/formatting/
 */
#show "ArtosFlow": name => box[
#box(image(
"logo.svg",
20 height: 0.7em,
))
#name
]
25 // Long line that breaks
This report is embedded in the ArtosFlow project. ArtosFlow is a project of the
Artos Institute.
/*
Very long line without linebreak
30 with a preceding block comment
*/
This_report_is_embedded_in_the_ArtosFlow_project._ArtosFlow_is_a_project_of_the_Artos_Institut
// End example
35 ```
Normal #cmd[raw] works as expected:
#code-block(code-example)
40
Using #cmd[sourcecode] will add line numbers and a frame.
#code-block(sourcecode(code-example))
45
#pagebreak()
#code-block(sourcecode(
showrange: (15, 21), showlines: true,
code-example))
50
Sourcecode can be loaded from a file and passed to #cmd[sourcefile]. Any
#code1st sourcecode can be wrapped inside #cmd[figure] as expected.
#code1st line numbers can be formatted via a bunch of `numbers-` options:
#code-block[
55 #let filename = "typst.toml"
#let number-format(n) = text(fill: blue, emph(n))
#show figure.where(kind: raw): (fig) => grid(
columns: (1fr, 2fr),
```

```

60 gutter: .65em,
   [
   #set align(left)
   #set par(justify:true)
   To the right in @lst-sourcefile you can see the #raw(filename) file of this
   package with some #number-format[fancy line numbers].
65 ],
   fig
   )

#figure(
70 caption: filename,
   sourcefile(
   numbers-side: right,
       numbers-style: number-format,
   file: filename,
75 read(filename))
   )<lst-sourcefile>
   ]

Since packages can't #cmd[read] files from outside their own directory, you can
alias #cmd[sourcefile] for a more convenient command:

80 ```typc
let srcfile( filename, ..args ) = sourcefile(read(filename),
file:filename, ..args)
```

#let srcfile(filename, ..args) = sourcefile(read(filename),
file:filename, ..args)

85 Formatting is controlled through options. To use a default style, create an
alias for your command:

```typc
let code = sourcecode.with(
90 numbers-style: (lno) => text(black, lno),
frame: none
)
```

#cmd[sourcecode] accepts a number of arguments to affect the output like
95 _highlighting lines_, _restrict the line range_ or _place labels_ in specific
lines to reference them later.
#code-block[
#sourcecode(
numbers-start: 9,
highlighted: (14,),
100 highlight-labels: true,
highlight-color: rgb(250, 190, 144),
gutter: 2em,
label-regex: regex("<([a-z-]+)>"),
frame: (code) => block(width:100%, fill: rgb(254, 249, 222), inset: 5pt, code)
105)][```typ
 #"hello world!" \
 #"\"hello\n world\"!" \
 #"1 2 3".split() \ <split-example>
 #"1,2;3".split(regex("[,;]")) \
110 #(regex("\\d+") in "ten euros") \
 #(regex("\\d+") in "10 euros")
    ```]
]

115 To reference a line use #cmd[lineref]:
- See #lineref(<split-example>) for an example of the `split()` function.

```

Long code breaks to new pages. To have listings in figures break, you need to allow it via a `#cmd[show]` rule:

```
120 ```typ
    #show figure.where(kind: raw): set block(breakable: true)
    ```
 #[
125 #show figure.where(kind: raw): set block(breakable: true)
 #show figure.where(kind: raw): (fig) => [
 #v(1em)
 #set align(center)
130 #strong([#fig.supplement #fig.counter.display()]): #emph(fig.caption.body)
 #fig.body
]
 #figure(
135 srcfile("example.typ", highlighted: range(121, 136), numbers-step: 5, numbers-
 first: 5),
 caption: "Code of this example file."
)
]
140 #pagebreak()
 == More examples

 And last but not least, some weird examples of stuff you can do with this
 package (example code taken from #link("https://github.com/rust-lang/rust-by-
 example/blob/master/src/fn.md", raw("rust-lang/rust-by-example"))):
145 #sourcecode(frame:none, numbering:none)[
    ```rust
    // Unlike C/C++, there's no restriction on the order of function definitions
    fn main() {
    // We can use this function here, and define it somewhere later
150 fizzbuzz_to(100);
    }
    ```

]
155 #sourcecode(
 numbering: "I",
 numbers-style: (lno) => align(right, [#text(eastern, emph(lno)) |]),
 gutter: 1em,
 tab-size: 8,
160 gobble: 1,
 showlines: true,
)[
    ```rust
165 // Function that returns a boolean value
    fn is_divisible_by(lhs: u32, rhs: u32) -> bool {
    // Corner case, early return
    if rhs == 0 {
170 return false;
    }

    // This is an expression, the `return` keyword is not necessary here
    lhs % rhs == 0
175 }

    ```

]
180 #block(width:100%)[
 #sourcecode(
```

```

 numbers-width: -6mm,
 frame: block.with(width: 75%, fill:rgb("#b7d4cf"), inset:5mm)
185)["``rust
 // Functions that "don't" return a value, actually return the unit type `()`
 fn fizzbuzz(n: u32) -> () {
 if is_divisible_by(n, 15) {
 println!("fizzbuzz");
190 } else if is_divisible_by(n, 3) {
 println!("fizz");
 } else if is_divisible_by(n, 5) {
 println!("buzz");
 } else {
195 println!("{}", n);
 }
 }
}
```
200 #place(top+right, block(width:23%)[
    #set par(justify:true)
    #lorem(40)
])
]
205 #sourcecode(
    numbering: "(1)",
    numbers-side: right,
    numbers-style: (lno) => text(1.5em, rgb(143, 254, 9), [#sym.arrow.l #lno]),
210 frame: (code) => {
    set text(luma(245))
    code-frame(
        fill: luma(24),
        stroke: 4pt + rgb(143, 254, 9),
215 radius: 0pt,
        inset: .65em,
        code
    )
    })["``rust
220 // When a function returns `()``, the return type can be omitted from the
    // signature
    fn fizzbuzz_to(n: u32) {
        for n in 1..=n {
            fizzbuzz(n);
225         }
    }
}
```

```

## More examples

And last but not least, some weird examples of stuff you can do with this package (example code taken from [rust-lang/rust-by-example](https://rust-lang/rust-by-example/)):

```
// Unlike C/C++, there's no restriction on the order of function definitions
fn main() {
 // We can use this function here, and define it somewhere later
 fizzbuzz_to(100);
}
```

```
I |
II |
III | // Function that returns a boolean value
IV | fn is_divisible_by(lhs: u32, rhs: u32) -> bool {
V | // Corner case, early return
VI | if rhs == 0 {
VII | return false;
VIII | }
IX |
X | // This is an expression, the `return` keyword is not necessary here
XI | lhs % rhs == 0
XII | }
XIII |
XIV |
```

```
1 // Functions that "don't" return a value, actually return
2 the unit type `()`
3 fn fizzbuzz(n: u32) -> () {
4 if is_divisible_by(n, 15) {
5 println!("fizzbuzz");
6 } else if is_divisible_by(n, 3) {
7 println!("fizz");
8 } else if is_divisible_by(n, 5) {
9 println!("buzz");
10 } else {
11 println!("{}", n);
12 }
}
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aequale doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum

```
// When a function returns `()`, the return type can be omitted from the ← (1)
// signature ← (2)
fn fizzbuzz_to(n: u32) { ← (3)
 for n in 1..=n { ← (4)
 fizzbuzz(n); ← (5)
 } ← (6)
} ← (7)
```