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 preceeding 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
   the Artos Institute.
15
16 /*
17  Very long line without linebreak
18  with a preceeding block comment
19 */
20 This_report_is_embedded_in_the_ArtosFlow_project._ArtosFlow_is_a_project_of_the_Artos_Institut
21
22 // End example
```

```

/*
 Very long line without linebreak
 with a preceeding block comment
*/
This_report_is_embedded_in_the_ArtosFlow_project._ArtosFlow_is_a_project_of_the_Artos_Institut

```

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*.

```toml
[package]
name = "codelst"
version = "2.0.0"
entrypoint = "codelst.typ"
authors = ["Jonas Neugebauer"]
license = "MIT"
description = "A typst package to render sourcecode."
repository = "https://github.com/jneug/typst-codelst"
exclude = ["example.typ", "example.pdf", "manual.pdf", "manual.typ", "tbump.toml"]
compiler = "0.0.9"
keywords = ["sourcecode", "code", "syntax-highlighting", "raw", "line numbers"]
```

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.

```
  #"hello world!" \
  #"\"hello\n  world\"!" \
  #"1 2 3".split() \
  #"1,2;3".split(regex("[,;]")) \
  #(regex("\d+") in "ten euros") \
  #(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 "./codelst.typ": sourcecode, sourcefile, lineref, code-frame
#let codelst = text(fill: rgb(254,48,147), smallcaps("codelst"))
#let cmd( name ) = text(fill: rgb(99, 170, 234), raw(block:false, sym.hash +
name.text + sym.paren.l + sym.paren.r))

#let code-block = block.with(
  stroke: 1pt,
  inset: 0.65em,
  radius: 4pt
)
#let code-example = ```typ
/*
 * 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 preceeding block comment
*/
This_report_is_embedded_in_the_ArtosFlow_project._ArtosFlow_is_a_project_of_the_Artos_Institute

// End example
```

Normal #cmd[raw] works as expected:

#code-block(code-example)

Using #cmd[sourcecode] will add line numbers and a frame.

#code-block(sourcecode(code-example))

#pagebreak()
#code-block(sourcecode(
  showrange: (15, 21), showlines: true,
code-example))

Sourcecode can be loaded from a file and passed to #cmd[sourcefile]. Any
#codelst sourcecode can be wrapped inside #cmd[figure] as expected.

#codelst line numbers can be formatted via a bunch of `numbers-` options:

#code-block[
  #let filename = "typst.toml"
  #let number-format(n) = text(fill: blue, emph(n))

  #show figure.where(kind: raw): (fig) => grid(
    columns: (1fr, 2fr),
````
````

```
    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].
    ],
    fig
  )

  #figure(
    caption: filename,
    sourcefile(
      numbers-side: right,
      numbers-style: number-format,
      file: filename,
      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:

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

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

#cmd[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.
#code-block[
  #sourcecode(
    numbers-start: 9,
    highlighted: (14,),
    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)
  )[```typ
  #"hello world!" \
  #"\"hello\n  world\"!" \
  #"1 2 3".split() \ <split-example>
  #"1,2;3".split(regex("[,;]")) \
  #(regex("\d+") in "ten euros") \
  #(regex("\d+") in "10 euros")
  ```]
]

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:

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

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

  #show figure.where(kind: raw): (fig) => [
    #v(1em)
    #set align(center)
    #strong([#fig.supplement #fig.counter.display()]): #emph(fig.caption.body)
    #fig.body
  ]

  #figure(
    srcfile("example.typ", highlighted: range(121, 136), numbers-step: 5, numbers-first: 5),
    caption: "Code of this example file."
  )
]

#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"))):

#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
  fizzbuzz_to(100);
}
  ```
]

#sourcecode(
  numbering: "I",
  numbers-style: (lno) => align(right, [#text(eastern, emph(lno)) |]),
  gutter: 1em,
  tab-size: 8,
  gobble: 1,
  showlines: true,
)[
  ```rust

    // Function that returns a boolean value
    fn is_divisible_by(lhs: u32, rhs: u32) -> bool {
        // Corner case, early return
        if rhs == 0 {
            return false;
        }

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

  ```
]
```

````
#block(width:100%)[
  #sourcecode(
    numbers-width: -6mm,
    frame: block.with(width: 75%, fill:rgb("#b7d4cf"), inset:5mm)
  )[```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");
      } else if is_divisible_by(n, 3) {
          println!("fizz");
      } else if is_divisible_by(n, 5) {
          println!("buzz");
      } else {
          println!("{}", n);
      }
  }
  ```]
  #place(top+right, block(width:23%)[
    #set par(justify:true)
    #lorem(40)
  ])
]

#sourcecode(
  numbering: "(1)",
  numbers-side: right,
  numbers-style: (lno) => text(1.5em, rgb(143, 254, 9), [#sym.arrow.l #lno]),
  frame: (code) => {
    set text(luma(245))
    code-frame(
      fill: luma(24),
      stroke: 4pt + rgb(143, 254, 9),
      radius: 0pt,
      inset: .65em,
      code
    )
  })[```rust
// 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);
    }
}
```]
````

## 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`):

```
// 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 |
```

```
// Functions that "don't" return a value, actually return
the unit type `()`
fn fizzbuzz(n: u32) -> () {
    if is_divisible_by(n, 15) {
        println!("fizzbuzz");
    } else if is_divisible_by(n, 3) {
        println!("fizz");
    } else if is_divisible_by(n, 5) {
        println!("buzz");
    } else {
        println!("{}", n);
    }
}
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque 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)
```