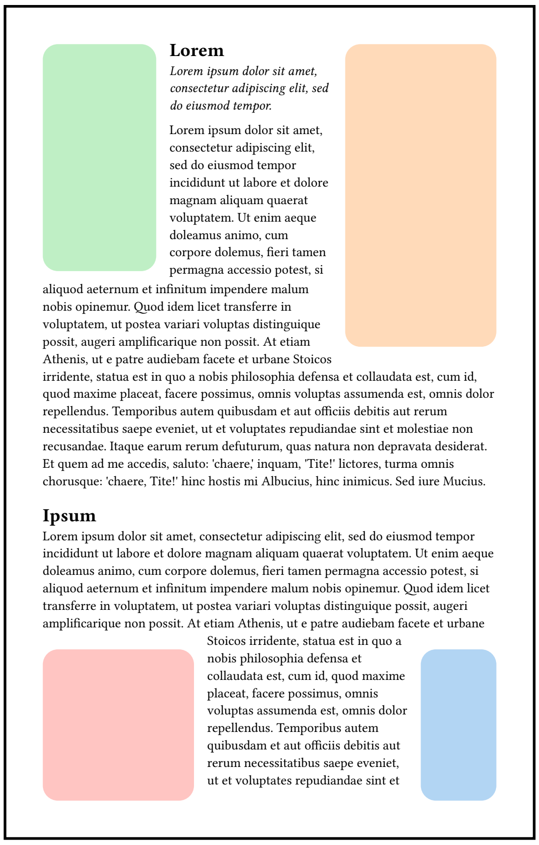
Meander
User guide

Abstract
Meander implements a content layout algorithm to provide text threading (when text from one box spills into a different box if it overflows), uneven columns, and image wrap-around.

Feature requests
For as long as the feature doesn't exist natively in Typst (see issue: `github:typst/typst #5181`), feel free to submit test cases of layouts you would like to see supported by opening a new issue.

Versions
- `dev`
- `0.1.0`



Lorem

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor.

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 impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedis, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius.

Ipsum

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 impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et

Contents

A - Quick start

The main function provided is `meander.reflow`, which takes as input some content, and auto-splits it into "containers", "obstacles", and "flowing text". Obstacles are `content` that are placed on the page with a fixed layout. Containers are created by the function `meander.container`, and everything else is flowing text.

After excluding the zones forbidden by obstacles and segmenting the containers appropriately, the threading algorithm will split the flowing content across containers to wrap around the forbidden regions.

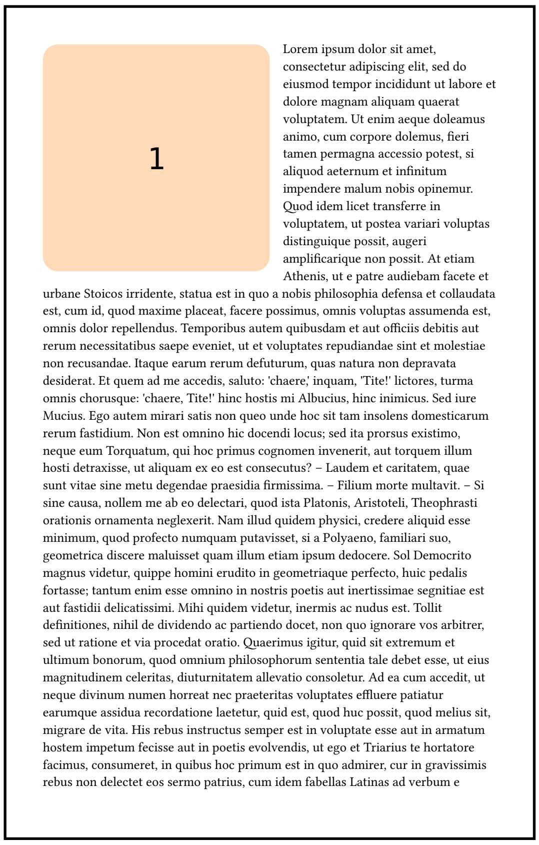
A.1 - A simple example

`meander.reflow` is contextual, so the invocation needs to be wrapped in a `context { ... }` block. Currently multi-page setups are not supported, but this is definitely a desired feature.

```
#context meander.reflow[
  // Obstacle
  #place(top + left, my-image-1)

  // Full-page container
  #meander.container()

  // Flowing text
  #lorem(500)
]
```



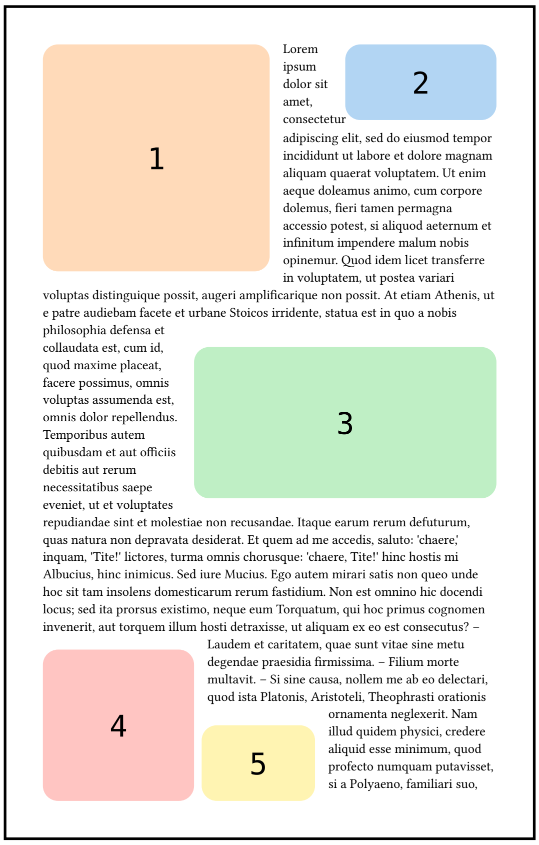
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedis, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius. Ego autem mirari satis non queo unde hoc sit tam insolens domesticarum rerum fastidium. Non est omnino hic docendi locus; sed ita prorsus existimo, neque eum Torquatum, qui hoc primus cognomen invenerit, aut torquem illum hosti detraxisse, ut aliquam ex eo est consecutus? – Laudem et caritatem, quae sunt vitae sine metu degendae praesidia firmissima. – Filium morte multavit. – Si sine causa, nollem me ab eo delectari, quod ista Platonis, Aristoteli, Theophrasti orationis ornamenta neglexerit. Nam illud quidem physici, credere aliquid esse minimum, quod profecto numquam putavisset, si a Polyaeno, familiari suo, geometrica discere maluisset quam illum etiam ipsum dedocere. Sol Democrito magnus videtur, quippe homini erudito in geometriaque perfecto, huic pedalis fortasse; tantum enim esse omnino in nostris poetis aut inertissimae segnitiae est aut fastidii delicatissimi. Mihi quidem videtur, inermis ac nudus est. Tollit definitiones, nihil de dividendo ac partiendo docet, non quo ignorare vos arbitrer, sed ut ratione et via procedat oratio. Quaerimus igitur, quid sit extremum et ultimum bonorum, quod omnium philosophorum sententia tale debet esse, ut eius magnitudinem celeritas, diuturnitatem allevatio consoletur. Ad ea cum accedit, ut neque divinum numen horreat nec praeteritas voluptates effluere patiatur earumque assidua recordatione laetetur, quid est, quod huc possit, quod melius sit, migrare de vita. His rebus instructus semper est in voluptate esse aut in armatum hostem impetum fecisse aut in poetis evolvendis, ut ego et Triarius te hortatore facimus, consumeret, in quibus hoc primum est in quo admirer, cur in gravissimis rebus non delectet eos sermo patrius, cum idem fabellas Latinas ad verbum e

A.2 - Multiple obstacles

`meander.reflow` can handle as many obstacles as you provide (at the cost of potentially performance issues if there are too many, but experiments have shown that up to ~100 obstacles is no problem).

```
#context meander.reflow[
  // Multiple obstacles
  #place(top + left, my-image-1)
  #place(top + right, my-image-2)
  #place(right, my-image-3)
  #place(bottom + left, my-image-4)
  #place(bottom + left, my-image-5,
    dx: 2cm)

  #meander.container()
  #lorem(500)
]
```



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 impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedis, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius. Ego autem mirari satis non queo unde hoc sit tam insolens domesticarum rerum fastidium. Non est omnino hic docendi locus; sed ita prorsus existimo, neque eum Torquatum, qui hoc primus cognomen invenerit, aut torquem illum hosti detraxisse, ut aliquam ex eo est consecutus? – Laudem et caritatem, quae sunt vitae sine metu degendae praesidia firmissima. – Filium morte multavit. – Si sine causa, nollem me ab eo delectari, quod ista Platonis, Aristoteli, Theophrasti orationis ornamenta neglexerit. Nam illud quidem physici, credere aliquid esse minimum, quod profecto numquam putavisset, si a Polyaeno, familiari suo,

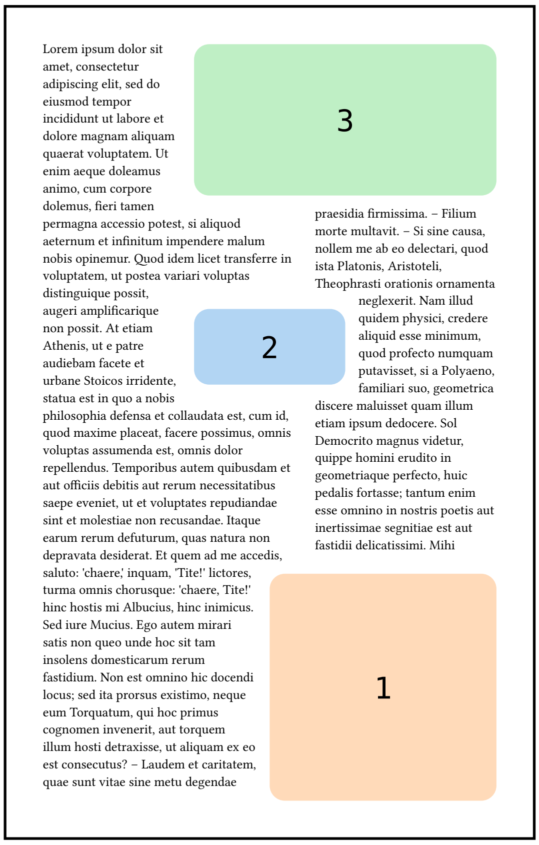
A.3 - Columns

In order to simulate a multi-column layout, you can provide several `container` invocations. They will be filled in the order provided.

```
#context meander.reflow[
  #place(bottom + right, my-image-1)
  #place(center + horizon, my-image-2,
    dy: -1cm)
  #place(top + right, my-image-3)

  // Multiple containers produce
  // multiple columns.
  #meander.container(width: 55%)
  #meander.container(right, width: 40%)

  #lorem(600)
]
```



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 impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura non depravata desiderat. Et quem ad me accedis, saluto: 'chaere,' inquam, 'Tite!' lictores, turma omnis chorusque: 'chaere, Tite!' hinc hostis mi Albucius, hinc inimicus. Sed iure Mucius. Ego autem mirari satis non queo unde hoc sit tam insolens domesticarum rerum fastidium. Non est omnino hic docendi locus; sed ita prorsus existimo, neque eum Torquatum, qui hoc primus cognomen invenerit, aut torquem illum hosti detraxisse, ut aliquam ex eo est consecutus? – Laudem et caritatem, quae sunt vitae sine metu degendae

praesidia firmissima. – Filium morte multavit. – Si sine causa, nollem me ab eo delectari, quod ista Platonis, Aristoteli, Theophrasti orationis ornamenta neglexerit. Nam illud quidem physici, credere aliquid esse minimum, quod profecto numquam putavisset, si a Polyaeno, familiari suo, geometrica discere maluisset quam illum etiam ipsum dedocere. Sol Democrito magnus videtur, quippe homini erudito in geometriaque perfecto, huic pedalis fortasse; tantum enim esse omnino in nostris poetis aut inertissimae segnitiae est aut fastidii delicatissimi. Mihi

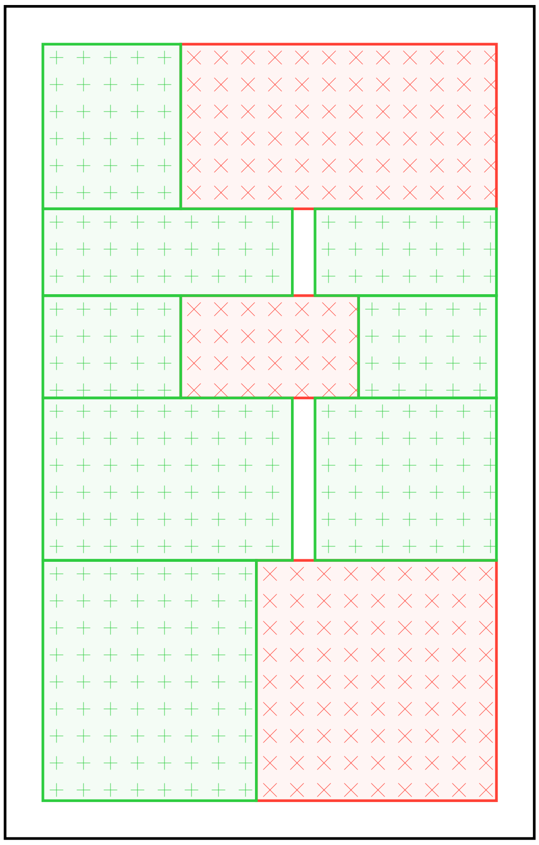
B - Understanding the algorithm

The same page setup as the previous example will internally be separated into

- obstacles `my-image-1`, `my-image-2`, and `my-image-3`. They are shown on the right in red.
- containers `(x: 0%, y: 0%, width: 55%, height: 100%)` and `(x: 60%, y: 0%, width: 40%, height: 100%)`
- flowing text `lorem(600)`, not shown here.

Respecting the horizontal separations of the obstacles, and staying within the bounds of the containers, the page is split into the subcontainers shown to the right in green. These boxes will be filled in order, including heuristics to properly provide vertical spacing between these boxes.

This debug view is visible by simply replacing `reflow` with `debug-reflow`.

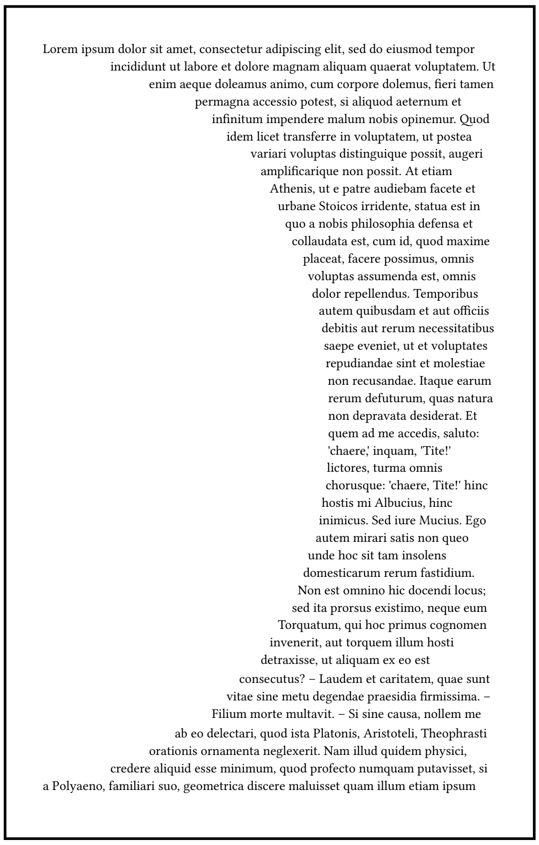


C - Advanced techniques

Here is a way to achieve text that follows a special shape.

```
#context meander.reflow[
  // Draw a half circle of empty boxes
  // that will count as obstacles
  #let vradius = 45%
  #let vcount = 50
  #let hradius = 60%
  #for i in range(vcount) {
    let frac = 2 * (i+0.5) / vcount - 1
    let width = hradius *
      calc.sqrt(1 - frac * frac)
    place(left + horizon,
      dy: (i - vcount / 2) *
        (2 * vradius / vcount)
    )[#box(width: width,
      height: 2 * vradius / vcount
    )]
  }

  // Then do the usual
  #meander.container()
  #lorem(600)
]
```



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 impendere malum nobis opinemur. Quod
idem licet transferre in voluptatem, ut postea
variari voluptas distinguique possit, augeri
amplificarique non possit. At etiam
Athenis, ut e patre audiebam facete et
urbane Stoicos irridente, statua est in
quo a nobis philosophia defensa et
collaudata est, cum id, quod maxime
placeat, facere possimus, omnis
voluptas assumenda est, omnis
dolor repellendus. Temporibus
autem quibusdam et aut officiis
debitis aut rerum necessitatibus
saepe eveniet, ut et voluptates
repudiandae sint et molestiae
non recusandae. Itaque earum
rerum defuturum, quas natura
non depravata desiderat. Et
quem ad me accedis, saluto:
'chaere,' inquam, 'Tite!'
lictores, turma omnis
chorusque: 'chaere, Tite!' hinc
hostis mi Albucius, hinc
inimicus. Sed iure Mucius. Ego
autem mirari satis non queo
unde hoc sit tam insolens
domesticarum rerum fastidium.
Non est omnino hic docendi locus;
sed ita prorsus existimo, neque eum
Torquatum, qui hoc primus cognomen
invenerit, aut torquem illum hosti
detraxisse, ut aliquam ex eo est
consecutus? – Laudem et caritatem, quae sunt
vitae sine metu degendae praesidia firmissima. –
Filium morte multavit. – Si sine causa, nollem me
ab eo delectari, quod ista Platonis, Aristoteli, Theophrasti
orationis ornamenta neglexerit. Nam illud quidem physici,
credere aliquid esse minimum, quod profecto numquam putavisset, si
a Polyaeno, familiari suo, geometrica discere maluisset quam illum etiam ipsum

There are limits to this technique, and in particular increasing the number of obstacles will in turn increase the number of boxes that the layout is segmented into. This means
- performance issues if you get too wild (though notice that having 50 obstacles in the previous example went fine)
- text that doesn't fit in the boxes at all, in particular if you don't give them any vertical space to grow because they are bounded on both sides.

In short, stay reasonable with this and don't try to add hundreds of obstacles of 1mm height each.

D - Modularity (WIP)

Because meander is cleanly split into three algorithms (content segmentation, page segmentation, text threading), there are plans to provide
- configuration options for each of those steps
- the ability to replace entirely an algorithm by either a variant, or a user-provided alternative that follows the same signature.

E - Module details

E.1 - Geometry (`geometry.typ`)

Generalist functions for 1D and 2D geometry.

- clamp()
- between()
- intersects()
- resolve()
- align()

clamp

Bound a value between `min` and `max`. No constraints on types as long as they support inequality testing.

Parameters

```
clamp(
  val: any,
  min: any none,
  max: any none
) -> any
```

val `any`

Base value.

min `any` or `none`

Lower bound.

Default: `none`

max `any` or `none`

Upper bound.

Default: `none`

between

Testing `a <= b <= c`, helps only computing `b` once.

Parameters

```
between(
  a: length,
  b: length,
  c: length
) -> bool
```

a `length`

Lower bound.

b `length`

Tested value.

c `length`

Upper bound. Asserted to be `>= c`.

intersects

Tests if two intervals intersect.

Parameters

```
intersects(
  i1: (length, length),
  i2: (length, length),
  tolerance: length
)
```

i1 `(length, length)`

First interval as a tuple of `(low, high)` in absolute lengths.

i2 `(length, length)`

Second interval.

tolerance `length`

Set to nonzero to ignore small intersections.

Default: `0pt`

resolve

Converts relative and contextual lengths to absolute. The return value will contain each of the arguments once converted, with arguments that contain `'x'` or start with `'w'` being interpreted as horizontal, and arguments that contain `'y'` or start with `'h'` being interpreted as vertical.

```
#context resolve(
    (width: 100pt, height: 200pt),
    x: 10%, y: 50% + 1pt,
    width: 50%, height: 5pt,
)
```

```
(x: 10pt, y: 101pt, width: 50pt, height: 5pt)
```

Parameters

```
resolve(
  size: (width: length, height: length) ,
  ..args: dictionary
) -> dictionary
```

size `(width: length, height: length)`

Size of the container as given by the `layout` function.

align

Compute the position of the upper left corner, taking into account the alignment and displacement.

Parameters

```
align(
  alignment: alignment ,
  dx: relative ,
  dy: relative ,
  width: relative ,
  height: relative
) -> (x: relative, y: relative)
```

alignment `alignment`

Absolute alignment.

dx `relative`

Horizontal displacement.

Default: `0pt`

dy `relative`

Vertical displacement.

Default: `0pt`

width `relative`

Object width.

Default: `0pt`

height `relative`

Object height.

Default: `0pt`

E.2 - Tiling (`tiling.typ`)

Page splitting algorithm.

- separate()
- container()
- pat-forbidden()
- pat-allowed()
- forbidden-rectangles()
- tolerable-rectangles()
- debug-reflow()

separate

Splits content into obstacles, containers, and flowing text.

An “obstacle” is any content inside a `place` at the toplevel. It will be appended in order to the `placed` field as `content`.

A “container” is a `box(place({}))`. Both `box` and `place` are allowed to have `width`, `height`, etc. parameters, but no inner contents. It will be appended in order to the `free` field as a block, i.e. a dictionary with the fields `x`, `y`, `width`, `height` describing the upper left corner and the dimensions of the container. See the helper function `container` that constructs a container directly.

Everything that is neither obstacle nor container is flowing text, and will end in the field `flow`.

```
#separate[
  // This is an obstacle
  #place(top + left, box(width: 50pt, height: 50pt))
  // This is a container
  #box(height: 50%, place({}))
  // This is flowing text
  #lorem(50)
]
```

Parameters

```
separate(ct: content) -> (containers: (..block,), obstacles: (..content,), flow: content)
```

container

Creates a standard container. This is not obscure, it's simply a `box(place({}))`, which is by convention recognized by `separate` as a container.

Parameters

```
container(..args: args) -> content
```

..args `args`

Accepts the parameters:

- `alignment` (positional, default top + left), passed to `place`
- `dx: relative` (named, default `0%`), passed to `place`
- `dy: relative` (named, default `0%`), passed to `place`
- `width: relative` (named, default `100%`), passed to `box`
- `height: relative` (named, default `100%`), passed to `box`

pat-forbidden

Pattern with red crosses to display forbidden zones.

Parameters

```
pat-forbidden(sz) -> pattern
```

sz

Size of the tiling

pat-allowed

Pattern with green pluses to display allowed zones.

Parameters

```
pat-allowed(sz) -> pattern
```

sz

Size of the tiling

forbidden-rectangles

From a set of obstacles (see `separate`: an obstacle is any `placed` content at the toplevel, so excluding `place`s that are inside `box`, `rect`, etc.), construct the blocks `(x: length, y: length, width: length, height: length)` that surround the obstacles.

The return value is as follows:

- `rects`, a list of `block`s `(x: length, y: length, width: length, height: length)`
- `display`, show this to include the placed content in the final output
- `debug`, show this to include helper boxes to visualize the layout

Parameters

```
forbidden-rectangles(
  obstacles: (..content,) ,
  margin: length ,
  size: (width: length, height: length)
) -> (rects: (..block,), display: content, debug: content)
```

obstacles `(..content,)`

Array of all the obstacles that are placed on this document.

margin `length`

Add padding around the obstacles.

Default: `0pt`

size `(width: length, height: length)`

Dimensions of the parent container, as provided by `layout`.

Default: `none`

tolerable-rectangles

Partition the complement of `avoid` into `containers` as a series of rectangles.

The algorithm is roughly as follows:

```
for container in containers {
  horizontal-cuts = sorted(top and bottom of zone for zone in avoid)
  for (top, bottom) in horizontal-cuts.windows(2) {
    vertical-cuts = sorted(
      left and right of zone for zone in avoid
      if zone intersects (top, bottom)
    )
    new zone (top, bottom, left, right)
  }
}
```

The main difficulty is in bookkeeping and handling edge cases (weird intersections, margins of error, containers that overflow the page, etc.) There are no heuristics to exclude zones that are too small, and no worries about zones that intersect vertically. That would be the threading algorithm's job.

Blocks are given an additional field `bounds` that dictate the upper limit of how much this block is allowed to stretch vertically, set to the dimensions of the container that produced this block.

Parameters

```
tolerable-rectangles(
  containers,
  avoid,
  size
) -> (rects: (..block,), debug: content)
```

debug-reflow

Debug version of the toplevel `reflow`, that only displays the partitioned layout.

Parameters

```
debug-reflow(ct: content) -> content
```

ct `content`

Content to be segmented and have its layout displayed.

E.3 - Bisection (`bisect.typ`)

Content splitting algorithm.

- fits-inside()
- default-rebuild()
- take-it-or-leave-it()
- has-text()
- has-child()
- has-children()
- is-list-item()
- is-enum-item()
- has-body()
- dispatch()
- fill-box()

fits-inside

Tests if content fits inside a box.

WARNING: horizontal fit is not strictly checked

The closure of this function constitutes the basis of the entire content splitting algorithm: iteratively add content until it no longer `fits-inside`, with what "iteratively add content" means being defined by the content structure. Essentially all remaining functions in this file are about defining content that can be split and the correct way to invoke `fits-inside` on them.

```
#let dims = (width: 100%, height: 50%)
#box(width: 7cm, height: 3cm)[#layout(size
=> context {
  let words = [#lorem(12)]
  [#fits-inside(dims, words, size: size)]
  linebreak()
  box(..dims, stroke: 0.1pt, words)
})]
```

```
true
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor.
```

```
#let dims = (width: 100%, height: 50%)
#box(width: 7cm, height: 3cm)[#layout(size
=> context {
  let words = [#lorem(15)]
  [#fits-inside(dims, words, size: size)]
  linebreak()
  box(..dims, stroke: 0.1pt, words)
})]
```

```
false
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor
incididunt ut labore.
```

Parameters

```
fits-inside(
  dims: (width: relative, height: relative),
  ct: content,
  size: (width: length, height: length)
) -> bool
```

dims `(width: relative, height: relative)`

Maximum container dimensions. Relative lengths are allowed.

ct `content`

Content to fit in.

size `(width: length, height: length)`

Dimensions of the parent container to resolve relative sizes. These must be absolute sizes.

Default: `none`

default-rebuild

Destructure and rebuild content, separating the outer content builder from the rest to allow substituting the inner contents. In practice what we will usually do is recursively split the inner contents and rebuild the left and right halves separately.

Inspired by `wrap-it`'s implementation (see: `_rewrap` in `github:ntjess/wrap-it`)

```
#let content = box(stroke: red)[Initial]
#let (inner, rebuild) = default-rebuild(
  content, "body",
)

Content: #content \
Inner: #inner \
Rebuild: #rebuild("foo")
```

```
Content: Initial
Inner: Initial
Rebuild: foo
```

```
#let content = [*_Initial_*]
#let (inner, rebuild) = default-rebuild(
  content, "body",
)

Content: #content \
Inner: #inner \
Rebuild: #rebuild("foo")
```

```
Content: Initial
Inner: Initial
Rebuild: foo
```

```
#let content = [a:b]
#let (inner, rebuild) = default-rebuild(
  content, "children",
)

Content: #content \
Inner: #inner \
Rebuild: #rebuild(([x], [y]))
```

```
Content: a:b
Inner: ([a], [:], [b])
Rebuild: xy
```

Parameters

```
default-rebuild(
  ct: content,
  inner-field: string
) -> (dictionnary, function)
```

inner-field `string`

What "inner" field to fetch (e.g. `"body"`, `"text"`, `"children"`, etc.)

take-it-or-leave-it

"Split" opaque content.

Parameters

```
take-it-or-leave-it(
  ct: content,
  fits-inside: function
) -> (content?, content?)
```

ct `content`

This content cannot be split. If it fits take it, otherwise keep it for later.

fits-inside `function`

Closure to determine if the content fits (see `fits-inside` above).

has-text

Split content with a `"text"` main field. Strategy: split by `" "` and take all words that fit.

Parameters

```
has-text(
  ct: content,
  split-dispatch: function,
  fits-inside: function,
  cfg: dictionary
)
```

ct `content`

Content to split.

split-dispatch `function`

Recursively passed around (see `split-dispatch` below).

fits-inside `function`

Closure to determine if the content fits (see `fits-inside` above).

cfg `dictionary`

Extra configuration options.

has-child

Split content with a `"child"` main field. Strategy: recursively split the child.

Parameters

```
has-child(
  ct: content,
  split-dispatch: function,
  fits-inside: function,
  cfg: dictionary
)
```

ct `content`

Content to split.

split-dispatch `function`

Recursively passed around (see `split-dispatch` below).

fits-inside `function`

Closure to determine if the content fits (see `fits-inside` above).

cfg `dictionary`

Extra configuration options.

has-children

Split content with a `"children"` main field. Strategy: take all children that fit.

Parameters

```
has-children(
  ct: content,
  split-dispatch: function,
  fits-inside: function,
  cfg: dictionary
)
```

ct `content`

Content to split.

split-dispatch `function`

Recursively passed around (see `split-dispatch` below).

fits-inside `function`

Closure to determine if the content fits (see `fits-inside` above).

cfg `dictionary`

Extra configuration options.

is-list-item

Split a `list.item`. Strategy: recursively split the `body`, and do some magic to simulate a bullet point indent.

Parameters

```
is-list-item(
  ct: content,
  split-dispatch: function,
  fits-inside: function,
  cfg: dictionary
)
```

ct `content`

Content to split.

split-dispatch `function`

Recursively passed around (see `split-dispatch` below).

fits-inside `function`

Closure to determine if the content fits (see `fits-inside` above).

cfg `dictionary`

Extra configuration options.

is-enum-item

Split an `enum.item`. Strategy: recursively split the `body`, and do some magic to simulate a numbering indent.

Parameters

```
is-enum-item(
  ct: content,
  split-dispatch: function,
  fits-inside: function,
  cfg: dictionary
)
```

ct `content`

Content to split.

split-dispatch `function`

Recursively passed around (see `split-dispatch` below).

fits-inside `function`

Closure to determine if the content fits (see `fits-inside` above).

cfg `dictionary`

Extra configuration options.

has-body

Split content with a `"body"` main field. There is a special strategy for `list.item` and `enum.item` which are handled separately. Elements `strong`, `emph`, `underline`, `stroke`, `overline`, `highlight` are splittable, the rest are treated as non-splittable.

Parameters

```
has-body(
  ct: content,
  split-dispatch: function,
  fits-inside: function,
  cfg: dictionary
)
```

ct `content`

Content to split.

split-dispatch `function`

Recursively passed around (see `split-dispatch` below).

fits-inside `function`

Closure to determine if the content fits (see `fits-inside` above).

cfg `dictionary`

Extra configuration options.

dispatch

Based on the fields on the content, call the appropriate splitting function. This function is involved in a mutual recursion loop, which is why all other splitting functions take this one as a parameter.

Parameters

```
dispatch(
  ct: content,
  fits-inside: function,
  cfg: dictionary
)
```

ct `content`

Content to split.

fits-inside `function`

Closure to determine if the content fits (see `fits-inside` above).

cfg `dictionary`

Extra configuration options.

fill-box

Initialize default configuration options and take as much content as fits in a box of given size.

Parameters

```
fill-box(
  dims: (width: length, height: length),
  ct: content,
  size: (width: length, height: length),
  cfg: dictionary
)
```

dims `(width: length, height: length)`

Container size.

ct `content`

Content to split.

size `(width: length, height: length)`

Parent container size.

Default: `(:)`

cfg `dictionary`

Configuration options.

- `list-markers: (..content,)`, default value `([•], [‣], [–], [•], [‣], [–])`. If you change the markers of `list`, put the new value in the parameters so that `list`s are correctly split.
- `enum-numbering: (..str,)`, default value `("1.", "1.", "1.", "1.", "1.", "1.")`. If you change the numbering style of `enum`, put the new style in the parameters so that `enum`s are correctly split.

Default: `(:)`

E.4 - Threading (`threading.typ`)

Filling and stretches boxes iteratively.

- smart-fill-boxes()
- reflow()

smart-fill-boxes

Thread text through a list of boxes in order, allowing the boxes to stretch vertically to accomodate for uneven tiling.

Parameters

```
smart-fill-boxes(
  body: content,
  avoid: (..block,),
  boxes: (..block,),
  extend: length,
  size: (width: length, height: length)
) -> (..content,)
```

body `content`

Flowing text.

avoid `(..block,)`

Obstacles to avoid. A list of `(x: length, y: length, width: length, height: length)`.

Default: `()`

boxes `(..block,)`

Boxes to fill. A list of `(x: length, y: length, width: length, height: length, bound: block)`.

`bound` is the upper limit of how much to stretch the container, i.e. also `(x: length, y: length, width: length, height: length)`.

Default: `()`

extend `length`

How much the baseline can extend downwards (within the limits of `bounds`).

Default: `1em`

size `(width: length, height: length)`

Dimensions of the container as given by `layout`.

Default: `none`

reflow

Segment the input content according to the tiling algorithm, then thread the flowing text through it.

Parameters

```
reflow(ct: content) -> content
```

ct `content`

See module `tiling` for how to format this content.