Metro

fenjalien and Mc-Zen

https://github.com/fenjalien/metro

Version 0.3.0

Requires Typst 0.11+

Contents

1 Introduction

The Metro package aims to be a port of the Latex package siunitx. It allows easy typesetting of numbers and units with options. This package is very early in development and many features are missing, so any feature requests or bug reports are welcome!

Metro's name comes from Metrology, the study scientific study of measurement.

2 Usage

Typst 0.11.0+ is required. You can import the package using the package manager:

```
#import "@preview/metro:0.3.0": *
```

Or download the `src` folder and import `lib.typ`:

```
#import "/src/lib.typ": *
```

2.1 Options

```
#metro-setup(..options)
```

All provided functions in this package have options that can control how they parse, process and print items. They can normally be given as keyword arguments directly to the function, but this can get tedious if you want the same options to apply throughout the document. You can instead use the `metro-setup` function. Any options given as keyword arguments will then be applied to the relevant subsequent functions in the document.

All options and function arguments will use the following types:

Literal Takes the given value directly. Input type is a string, content and sometimes a number.

Switch On-off switches. Input type is a boolean.

Choice Takes a limited number of choices, which are described separately for each option. Input type is a string.

Number A float or integer.

Integer An integer.

2.2 Numbers

```
#num(number, e: none, pm: none, pwr: none, ..options)
```

Parses, processes then prints a number. The number can be given as an integer, a float, a string, as some plain content or math content! The different forms of input should extend to all other functions with arguments that take a number, they will be parsed all the same. However it should be noted that:

- When giving a number as an integer or float with an exponent in the number, it will not be seen by Metro (e.g. `3.4e3` will be seen as `3400` and not "3.4 with an exponent of 3").
- When using one of Metro's function within math mode, Typst considers dashes as subtraction symbols which breaks identifier names. So any options with dashes will not be able to be used when in math mode.

123	`#num(123)\`
1234	`#num("1234")\`
12 345	`#num[12345]\`
0.123	`$num(0.123)$\`
0.1234	`#num("0,1234")\`
0.123 45	`#num[.12345]\`
3.45×10^{-4}	`#num(e: -4)[3.45]\`
-10^{10}	`#num("-1", e: 10, print-unity-mantissa: false)`

number `Literal`

The number to format.

pm `Literal` (default: `none`)

The uncertainty of the number.

e `Literal` (default: `none`)

The exponent of the number. It can also be given as an integer in the number argument when it is of type string or content. It should be prefixed with an "e" or "E".

1×10^{10}	`#num("1e10")\`
1×10^{10}	`#num[1E10]`

pwr `Literal` (default: `none`)

The power of the number, it will be attached to the top. No processing is currently done to the power. It can also be passed as an integer in the number parameter when it is of type string or content. It should be prefixed after the exponent with an "`^`".

1^{2}	`#num("1^2")\`
1^{2}	`$num(1^2)$`

2.2.1 Options

2.2.1.1 Parsing

input-decimal-markers `Array<Literal>` (default: `('\.', ',')`)

An array of characters that indicate the sepration between the integer and decimal parts of a number. More than one inupt decimal marker can be used, it will be converted by the package to the appropriate output marker.

retain-explicit-decimal-marker `Switch` (default: `false`)

Allows a trailing decimal marker with no decimal part present to be printed.

10
10.

```
#num[10.]\
#num(retain-explicit-decimal-marker: true)[10.]
```

retain-explicit-plus `Switch` (default: `false`)

Allows a leading plus sign to be printed.

345
+345

```
#num[+345]\
#num(retain-explicit-plus: true)[+345]
```

retain-negative-zero `Switch` (default: `false`)

Allows a negative sign on an entirely zero value.

0
−0

```
#num[-0]\
#num(retain-negative-zero: true)[-0]
```

parse-numbers `Switch` (default: `auto`)

Turns the entire parsing system on and off. It allows the use of arbitrary values in numbers. When the option is `auto`, numbers will be attempt to be parsed but will quietly stop if it fails to do so. The number will then be printed as given. If the option is `false`, no parsing will even be attempted. If `true`, Metro will panic if the number cannot be parsed.

$\sqrt{3}$
$\sqrt{4}$

```
$num(sqrt(3))$\
#metro-setup(parse-numbers: false)
$num(sqrt(4))$\
#metro-setup(parse-numbers: true)
// Will panic:
// $num(sqrt(5))$\
```

2.2.1.2 Post Processing

drop-exponent `Switch` (default: `false`)

When `true` the exponent will be dropped (*after* the processing of exponent)

0.01×10^{3}
0.01

```
#num("0.01e3")\
#num("0.01e3", drop-exponent: true)
```

drop-uncertainty `Switch` (default: `false`)

When `true` the uncertainty will be dropped.

0.01 ± 0.02
0.01

```
#num("0.01", pm: 0.02)\
#num("0.01", pm: 0.02, drop-uncertainty: true)\
```

drop-zero-decimal `Switch` (default: `false`)

When `true`, if the decimal is zero it will be dropped before setting the minimum numbers of digits.

2.1
2.0
2.1
2

```
#num[2.1]\
#num[2.0]\
#metro-setup(drop-zero-decimal: true)
#num[2.1]\
#num[2.0]\
```

exponent-mode `Choice` (default: `"input"`)

How to convert the number to scientific notation. Note that the calculated exponent will be added to the given exponent for all options.

input Does not perform any conversions, the exponent will be displayed as given.
scientific Converts the number such that the integer will always be a single digit.
fixed Convert the number to use the exponent value given by the `fixed-exponent` option.
engineering Converts the number such that the exponent will be a multiple of three.
threshold Like the `scientific` option except it will only convert the number when the exponent would be outside the range given by the `exponent-thresholds` option.

0.001
0.0100
1200
1×10^{-3}
1.00×10^{-2}
1.200×10^{3}
1×10^{-3}
10.0×10^{-3}
1.200×10^{3}
$00.000\,01 \times 10^{2}$
$00.000\,100 \times 10^{2}$
12.00×10^{2}

```
#let nums = [
  #num[0.001]\
  #num[0.0100]\
  #num[1200]\
]
#nums
#metro-setup(exponent-mode: "scientific")
#nums
#metro-setup(exponent-mode: "engineering")
#nums
#metro-setup(exponent-mode: "fixed", fixed-exponent: 2)
#nums
```

exponent-thresholds `Array<Integer>` (default: `(-3, 3)`)

Used to control the range of exponents that won't trigger when the `exponent-mode` is "threshold". The first value is the minimum inclusive, and the last value is the maximum inclusive.

```
#let inputs = (
  "0.001",
  "0.012",
  "0.123",
  "1",
  "12",
  "123",
  "1234"
)

#table(
  columns: (auto,)*3,
  [Input], [Threshold $-3:3$], [Threshold $-2:2$],
  ..for i in inputs {(
    num(i),
    num(i, exponent-mode: "threshold"),
    num(i, exponent-mode: "threshold", exponent-thresholds: (-2, 2)),
  )}
)
```

Input	Threshold $-3:3$	Threshold $-2:2$
0.001	1×10^{-3}	1×10^{-3}
0.012	0.012	1.2×10^{-2}
0.123	0.123	0.123
1	1	1
12	12	12
123	123	1.23×10^{2}
1234	1.234×10^{3}	1.234×10^{3}

fixed-exponent `Integer` (default: `0`)

The exponent value to use when `exponent-mode` is "fixed". When zero, this may be used to remove scientific notation from the input.

1.23×10^{4}	`#num("1.23e4")\`
12 300	`#num("1.23e4", exponent-mode: "fixed", fixed-exponent: 0)`

round-mode `Choice` (default: `"none"`)

How the package should round numerical input.

none No rounding is performed.

1.234 56
14.23

```
#num(1.23456)\
#num(14.23)
```

figures Round to a number of significant figures.

1.2
14

```
#metro-setup(round-mode: "figures")
#num(1.23456)\
#num(14.23)
```

places Round to a number of decimal places.

1.23
14.23

```
#metro-setup(round-mode: "places")
#num(1.23456)\
#num(14.23)
```

round-precision `Integer` (default: 2)

Controls the number of significant figures or decimal places to round to.

1.235
14.230
1.23
14.2

```
#metro-setup(round-mode: "places", round-precision: 3)
#num(1.23456)\
#num(14.23)\
#metro-setup(round-mode: "figures", round-precision: 3)
#num(1.23456)\
#num(14.23)\
```

round-pad `Switch` (default: `true`)

Controls when rounding may “extend” a short number to more digits (or figures).

12.30
12.3

```
#metro-setup(round-mode: "figures", round-precision: 4)
#num(12.3)\
#num(12.3, round-pad: false)\
```

round-direction `Choice` (default: `"nearest"`)

Determines which direction a value is rounded toward.

nearest Gives the common outcome that values round depending on whether the preceding digit is greater or less than 5.

0.05
0.05

```
#metro-setup(round-mode: "places")
#num(0.054)\
#num(0.046)
```

down Values are always rounded down. It may be thought of as “truncation”.

0.05
0.04

```
#metro-setup(round-mode: "places", round-direction: "down")
#num(0.054)\
#num(0.046)
```

up Values are always rounded up.

0.06
0.05

```
#metro-setup(round-mode: "places", round-direction: "up")
#num(0.054)\
#num(0.046)
```

round-half `Choice` (default: `"up"`)

Determines how numbers that are exactly half are rounded to the the `"nearest"`.

up The number is rounded up.

0.06
0.05

```
#metro-setup(round-mode: "figures", round-precision: 1)
#num(0.055)\
#num(0.045)\
```

even The number is rounded to the nearest even part.

0.06
0.04

```
#metro-setup(
  round-mode: "figures",
  round-precision: 1,
  round-half: "even"
)
#num(0.055)\
#num(0.045)\
```

round-minimum `Number` (default: `0`)

There are cases in which rounding will result in the number reaching zero. It may be desirable to show results as below a threshold value. This can be achieved by setting this option to the threshold value. There will be no effect when rounding to a number of significant figures as it is not possible to obtain the value zero in these cases.

0.01
0.00
0.01
< 0.01

```
#metro-setup(round-mode: "places")
#num(0.0055)\
#num(0.0045)\
#metro-setup(round-minimum: 0.01)
#num(0.0055)\
#num(0.0045)\
```

round-zero-positive `Switch` (default: `true`)

When rounding negative numbers to a fixed number of places, a zero value may result. Usually this is expressed as an unsigned value, but in some cases retaining the negative sign may be desirable. This behaviour can be controlled using this option.

0.00
−0.00

```
#metro-setup(round-mode: "places")
#num(-0.001)\
#metro-setup(round-zero-positive: false)
#num(-0.001)
```

minimum-decimal-digits `Integer` (default: `0`)

May be used to pad the decimal component of a number to a given size.

0.123
0.123
0.1230

```
#num(0.123)\
#num(0.123, minimum-decimal-digits: 2)\
#num(0.123, minimum-decimal-digits: 4)
```

minimum-integer-digits `Integer` (default: `0`)

May be used to pad the integer component of a number to a given size.

123
123
0123

```
#num(123)\
#num(123, minimum-integer-digits: 2)\
#num(123, minimum-integer-digits: 4)
```

2.2.1.3 Printing

group-digits `Choice` (default: `"all"`)

Whether to group digits into blocks to increase the ease of reading of numbers. Takes the values `all`, `none`, `decimal` and `integer`. Grouping can be acitivated separately for the integer and decimal parts of a number using the appropriately named values.

12 345.678 90	`#num[12345.67890]\`
12345.67890	`#num(group-digits: "none")[12345.67890]\`
12345.678 90	`#num(group-digits: "decimal")[12345.67890]\`
12 345.67890	`#num(group-digits: "integer")[12345.67890]`

group-separator `Literal` (default: `sym.space.thin`)

The separator to use between groups of digits.

12 345	`#num[12345]\`
12,345	`#num(group-separator: ",")[12345]\`
12 345	`#num(group-separator: " ")[12345]`

group-minimum-digits `Integer` (default: `5`)

Controls how many digits must be present before grouping is applied. The number of digits is considered separately for the integer and decimal parts of the number: grouping does not “cross the boundary”.

1234	`#num[1234]\`
12 345	`#num[12345]\`
1 234	`#num(group-minimum-digits: 4)[1234]\`
12 345	`#num(group-minimum-digits: 4)[12345]\`
1234.5678	`#num[1234.5678]\`
12 345.678 90	`#num[12345.67890]\`
1 234.567 8	`#num(group-minimum-digits: 4)[1234.5678]\`
12 345.678 90	`#num(group-minimum-digits: 4)[12345.67890]`

digit-group-size `Integer` (default: `3`)

Controls the number of digits in each group. Finer control can be achieved using `digit-group-first-size` and `digit-group-other-size`: the first group is that immediately by the decimal point, the other value applies to the second and subsequent groupings.

1 234 567 890	`#num[1234567890]\`
12345 67890	`#num(digit-group-size: 5)[1234567890]\`
1 23 45 67 890	`#num(digit-group-other-size: 2)[1234567890]`

output-decimal-marker `Literal` (default: `.`)

The decimal marker used in the output. This can differ from the input marker.

1.23	`#num(1.23)\`
1,23	`#num(output-decimal-marker: ",")[1.23]`

exponent-base `Literal` (default: `10`)

The base of an exponent.

1×2^2	`#num(exponent-base: "2", e: 2)[1]`

exponent-product `Literal` (default: `sym.times`)

The symbol to use as the product between the number and its exponent.

1×10^{2}	`#num(e: 2, exponent-product: sym.times)[1]\`
$1 \cdot 10^{2}$	`#num(e: 2, exponent-product: sym.dot)[1]`

output-exponent-marker `Literal` (default: `none`)

When not `none`, the value stored will be used in place of the normal product and base combination.

1e2	`#num(output-exponent-marker: "e", e: 2)[1]\`
1E2	`#num(output-exponent-marker: "E", e: 2)[1]`

bracket-ambiguous-numbers `Switch` (default: `true`)

There are certain combinations of numerical input which can be ambiguous. This can be corrected by adding brackets in the appropriate place.

$(1.2 \pm 0.3) \times 10^{4}$	`#num(e: 4, pm: 0.3)[1.2]\`
$1.2 \pm 0.3 \times 10^{4}$	`#num(bracket-ambiguous-numbers: false, e: 4, pm: 0.3)[1.2]`

bracket-negative-numbers `Switch` (default: `false`)

Whether or not to display negative numbers in brackets.

−15 673	`#num[-15673]\`
(15 673)	`#num(bracket-negative-numbers: true)[-15673]`

tight-spacing `Switch` (default: `false`)

Compresses spacing where possible.

2×10^{3}	`#num(e: 3)[2]\`
$2{\times}10^{3}$	`#num(e: 3, tight-spacing: true)[2]`

print-implicit-plus `Switch` (default: `false`)

Force the number to have a sign. This is used if given and if no sign was present in the input.

345	`#num(345)\`
+345	`#num(345, print-implicit-plus: true)`

It is possible to set this behaviour for the exponent and mantissa independently using `print-mantissa-implicit-plus` and `print-exponent-implicit-plus` respectively.

print-unity-mantissa `Switch` (default: `true`)

Controls the printing of a mantissa of 1.

1×10^{4}	`#num(e: 4)[1]\`
10^{4}	`#num(e: 4, print-unity-mantissa: false)[1]`

print-zero-exponent `Switch` (default: `false`)

Controls the printing of an exponent of 0.

444	`#num(e: 0)[444]\`
444×10^{0}	`#num(e: 0, print-zero-exponent: true)[444]`

print-zero-integer `Switch` (default: `true`)

Controls the printing of an integer component of 0.

0.123	`#num(0.123)\`
.123	`#num(0.123, print-zero-integer: false)`

zero-decimal-as-symbol `Switch` (default: `false`)

Whether to show entirely zero decimal parts as a symbol. Uses the symbol stroed using `zero-symbol` as the replacement.

123.00
123.—
123.[—]

```
#num[123.00]\
#metro-setup(zero-decimal-as-symbol: true)
#num[123.00]\
#num(zero-symbol: [[#sym.bar.h]])[123.00]
```

zero-symbol `Literal` (default: `sym.bar.h`)

The symbol to use when `zero-decimal-as-symbol` is `true`.

2.3 Units

```
#unit(unit, ..options)
```

Typsets a unit and provides full control over output format for the unit. The type passed to the function can be either a string or some math content.

When using the function in math mode, Typst accepts single characters but multiple characters together are expected to be variables. So Metro defines units and prefixes which be can imported to be used.

```
#import "@preview/metro:0.2.0": unit, units, prefixes
#unit($units.kg m/s^2$)
// because `units` and `prefixes` here are modules you can import what you need
#import units: gram, metre, second
#import prefixes: kilo
$unit(kilo gram metre / second^2)$
// You can also just import everything instead
#import units: *
#import prefixes: *
$unit(joule / mole / kelvin)$
```

$\mathrm{kg}\,\mathrm{m}\,\mathrm{s}^{-2}$
$\mathrm{kg}\,\mathrm{m}\,\mathrm{s}^{-2}$
$\mathrm{J}\,\mathrm{mol}^{-1}\,\mathrm{K}^{-1}$

When using strings there is no need to import any units or prefixes as the string is parsed. Additionally several variables have been defined to allow the string to be more human readable. You can also use the same syntax as with math mode.

```
// String
#unit("kilo gram metre per square second")\
// Math equivalent
#unit($kilo gram metre / second^2$)\
// String using math syntax
#unit("kilo gram metre / second^2")
```

$\mathrm{kg}\,\mathrm{m}\,\mathrm{s}^{-2}$
$\mathrm{kg}\,\mathrm{m}\,\mathrm{s}^{-2}$
$\mathrm{kg}\,\mathrm{m}\,\mathrm{s}^{-2}$

`per` used as in "metres *per* second" is equivalent to a slash `/`. When using this in a string you don't need to specify a numerator.

```
#unit("metre per second")\
$unit(metre/second)$\

#unit("per square becquerel")\
#unit("/becquerel^2")
```

$\mathrm{m}\,\mathrm{s}^{-1}$
$\mathrm{m}\,\mathrm{s}^{-1}$

Bq^{-2}
Bq^{-2}

`square` and `cubic` apply their respective powers to the units after them, while `squared` and `cubed` apply to units before them.

```
#unit("square becquerel")\
#unit("joule squared per lumen")\
#unit("cubic lux volt tesla cubed")
```

Bq^2
$\mathrm{J}^2\,\mathrm{lm}^{-1}$
$\mathrm{lx}^3\,\mathrm{V}\,\mathrm{T}^3$

Generic powers can be inserted using the `tothe` and `raiseto` functions. `tothe` specifically is equivalent to using caret ^.

```
#unit("henry tothe(5)")\
#unit($henry^5$)\
#unit("henry^5")

#unit("raiseto(4.5) radian")\
#unit($radian^4.5$)\
#unit("radian^4.5")
```

H^5
H^5
H^5

$\mathrm{rad}^{4.5}$
$\mathrm{rad}^{4.5}$
$\mathrm{rad}^{4.5}$

You can also use the `sqrt` function for half powers. If you want to maintain the square root, you must set the `power-half-as-sqrt` option.

$\mathrm{H}^{0.5}$ `$unit(sqrt(H))$\`
$\sqrt{\mathrm{H}}$ `#unit("sqrt(H)", power-half-as-sqrt: true)\`

Generic qualifiers are available using the `of` function which is equivalent to using an underscore _. Note that when using an underscore for qualifiers in a string with a space, to capture the whole qualifier use brackets `()`.

```
#unit("kilogram of(metal)")\
#unit($kilogram_"metal"$)\
#unit("kilogram_metal")

#metro-setup(qualifier-mode: "bracket")
#unit("milli mole of(cat) per kilogram of(prod)")\
#unit($milli mole_"cat" / kilogram_"prod"$)\
#unit("milli mole_(cat) / kilogram_(prod)")
```

$\mathrm{kg}_{\mathrm{metal}}$
$\mathrm{kg}_{\mathrm{metal}}$
$\mathrm{kg}_{\mathrm{metal}}$

$\mathrm{mmol(cat)}\,\mathrm{kg(prod)}^{-1}$
$\mathrm{mmol(cat)}\,\mathrm{kg(prod)}^{-1}$
$\mathrm{mmol(cat)}\,\mathrm{kg(prod)}^{-1}$

2.3.1 Options

inter-unit-product `Literal` (default: `sym.space.thin`)

The separator between each unit. The default setting is a thin space: another common choice is a centred dot.

$\mathrm{F^2\,lm\,cd}$

$\mathrm{F^2 \cdot lm \cdot cd}$

```
#unit("farad squared lumen candela")\
#unit("farad squared lumen candela", inter-unit-product: $dot.c$)
```

per-mode `Choice` (default: `"power"`)

Use to alter the handling of `per`.

power Reciprocal powers

$\mathrm{J\,mol^{-1}\,K^{-1}}$

$\mathrm{m\,s^{-2}}$

```
#unit("joule per mole per kelvin")\
#unit("metre per second squared")
```

fraction Uses the `math.frac` function (also known as `$ / $`) to typeset positive and negative powers of a unit separately.

$\frac{\mathrm{J}}{\mathrm{mol\,K}}$

$\frac{\mathrm{m}}{\mathrm{s^2}}$

```
#unit("joule per mole per kelvin", per-mode: "fraction")\
#unit("metre per second squared", per-mode: "fraction")
```

symbol Separates the two parts of a unit using the symbol in `per-symbol`. This method for displaying units can be ambiguous, and so brackets are added unless `bracket-unit-denominator` is set to `false`. Notice that `bracket-unit-denominator` only applies when `per-mode` is set to symbol.

$\mathrm{J/(mol\,K)}$

$\mathrm{m/s^2}$

```
#metro-setup(per-mode: "symbol")
#unit("joule per mole per kelvin")\
#unit("metre per second squared")
```

per-symbol `Literal` (default: `sym.slash`)

The symbol to use to separate the two parts of a unit when `per-symbol` is `"symbol"`.

```
#unit("joule per mole per kelvin", per-mode: "symbol", per-symbol: [ div ])
```

J div (mol K)

bracket-unit-denominator `Switch` (default: `true`)

Whether or not to add brackets to unit denominators when `per-symbol` is `"symbol"`.

```
#unit("joule per mole per kelvin", per-mode: "symbol", bracket-unit-
denominator: false)
```

$\mathrm{J/mol\,K}$

sticky-per `Switch` (default: `false`)

Normally, `per` applies only to the next unit given. When `sticky-per` is `true`, this behaviour is changed so that `per` applies to all subsequent units.

$\mathrm{Pa\,Gy^{-1}\,H}$

$\mathrm{Pa\,Gy^{-1}\,H^{-1}}$

```
#unit("pascal per gray henry")\
#unit("pascal per gray henry", sticky-per: true)
```

qualifier-mode `Choice` (default: `"subscript"`)

Sets how unit qualifiers can be printed.

subscript

```
#unit("kilogram of(pol) squared per mole of(cat) per hour")
```

$\mathrm{kg}_{\mathrm{pol}}^{2}\ \mathrm{mol}_{\mathrm{cat}}^{-1}\ \mathrm{h}^{-1}$

bracket

```
#unit("kilogram of(pol) squared per mole of(cat) per hour", qualifier-mode:
"bracket")
```

$\mathrm{kg(pol)}^{2}\ \mathrm{mol(cat)}^{-1}\ \mathrm{h}^{-1}$

combine Powers can lead to ambiguity and are automatically detected and brackets added as appropriate.

dBi

```
#unit("deci bel of(i)", qualifier-mode: "combine")
```

phrase Used with `qualifier-phrase`, which allows for example a space or other linking text to be inserted.

```
#metro-setup(qualifier-mode: "phrase", qualifier-phrase: sym.space)
#unit("kilogram of(pol) squared per mole of(cat) per hour")\
#metro-setup(qualifier-phrase: [ of ])
#unit("kilogram of(pol) squared per mole of(cat) per hour")
```

$\mathrm{kg\ pol}^{2}\ \mathrm{mol\ cat}^{-1}\ \mathrm{h}^{-1}$

$\mathrm{kg\ of\ pol}^{2}\ \mathrm{mol\ of\ cat}^{-1}\ \mathrm{h}^{-1}$

power-half-as-sqrt `Switch` (default: `false`)

When `true` the power of 0.5 is shown by giving the unit sumbol as a square root. This

$\mathrm{Hz}^{0.5}$

$\sqrt{\mathrm{Hz}}$

```
#unit("Hz tothe(0.5)")\
#unit("Hz tothe(0.5)", power-half-as-sqrt: true)
```

2.4 Quantities

```
#qty(number, unit, ..options)
```

This function combines the functionality of `num` and `unit` and formats the number and unit together. The `number` and `unit` arguments work exactly like those for the `num` and `unit` functions respectively.

$1.23\,\mathrm{J\,mol^{-1}\,K^{-1}}$
$0.23 \times 10^{7}\,\mathrm{cd}$
$1.99/\mathrm{kg}$
$1.345\,\frac{\mathrm{C}}{\mathrm{mol}}$

```
#qty(1.23, "J / mol / kelvin")\
$qty(.23, candela, e: 7)$\
#qty(1.99, "per kilogram", per-mode: "symbol")\
#qty(1.345, "C/mol", per-mode: "fraction")
```

2.4.1 Options

allow-quantity-breaks `Switch` (default: `false`)

Controls whether the combination of the number and unit can be split across lines.

```
#box(width: 3.25cm)[
  Some filler text #qty(10, "m")\
  #metro-setup(allow-quantity-breaks: true)
  Some filler text #qty(10, "m")
]
```

Some filler text

10 m

Some filler text 10

m

quantity-product `Literal` (default: `sym.space.thin`)

The product symbol between the number and unit.

```
#qty(2.67, "farad")\
#qty(2.67, "farad", quantity-product: sym.space)\
#qty(2.67, "farad", quantity-product: none)
```

2.67 F

2.67 F

2.67F

separate-uncertainty `Choice` (default: `"bracket"`)

When a number has multiple parts, then the unit must apply to all parts of the number.

bracket Places the entire numerical part in brackets and use a single unit symbol.

$(12.3 \pm 0.4)\,\mathrm{kg}$

```
#qty(12.3, "kg", pm: 0.4)
```

repeat Prints the unit for each part of the number.

$12.3\,\mathrm{kg} \pm 0.4\,\mathrm{kg}$

```
#qty(12.3, "kg", pm: 0.4, separate-uncertainty: "repeat")
```

single Prints only one unit symbol: mathematically incorrect.

$12.3 \pm 0.4\,\mathrm{kg}$

```
#qty(12.3, "kg", pm: 0.4, separate-uncertainty: "single")
```

2.5 List, Products and Ranges

```
#num-list(..numbers-options)
```

Lists of numbers may be processed using the `num-list` function. Each number should be given as a positional argument. The numbers are formatted using `num`.

10, 30, 50 and 70

```
#num-list(10, 30, 50, 70)
```

```
#num-product(..numbers-options)
```

Runs of products can be created using the `num-product` function. It acts in the same way `num-list` does.

10 × 30

```
#num-product(10, 30)
```

```
#num-range(number1, number2, ..options)
```

Simple ranges of numbers can be handled using the `num-range` function. It inserts a phrase or other text between the two numbers.

10 to 30

```
#num-range(10, 30)
```

The above list, product and range functions also have a `qty` variant where the last positional argument will be considered as a unit.

10 m, 30 m and 45 m
10 m × 30 m × 45 m
10 m to 30 m

```
#qty-list(10, 30, 45, metre)\
#qty-product(10, 30, 45, metre)\
#qty-range(10, 30, metre)\
```

The above function names cannot be used in math mode, instead equivalently named functions are provided that have the dash removed (e.g. `num-list` and `numlist`).

2.5.1 Options

list-separator `Literal` (default: `[, ]`)

The separator to place between each item in the a list of numbers.

0.1, 0.2 and 0.3
0.1; 0.2 and 0.3

```
#num-list(0.1, 0.2, 0.3) \
#num-list(
  list-separator: [; ],
  0.1, 0.2, 0.3,
)
```

list-final-separator `Literal` (default: `[ and ]`)

The separator before the last item of a list.

0.1, 0.2, 0.3
0.1 and 0.2 and 0.3

```
#num-list(
  list-final-separator: [, ],
  0.1, 0.2, 0.3
) \
#num-list(
  list-separator: [ and ],
  list-final-separator: [ and ],
  0.1, 0.2, 0.3
)
```

list-pair-separator `Literal` (default: `[ and ]`)

The to use for exactly two items of a list.

0.1 and 0.2
0.1, and 0.2

```
#num-list(0.1, 0.2) \
#num-list(
  list-pair-separator: [, and ],
  0.1, 0.2
)
```

product-mode `Choice` (default: `"symbol"`)

Products of numbers can be output using either a product symbol or a phrase.

symbol The symbol in `product-symbol` is used.

5 × 100 × 2

```
#num-product(5, 100, 2)
```

phrase The phrase in `product-phrase` is used.

5 by 100 by 2

```
#num-product(5, 100, 2, product-mode: "phrase")
```

product-symbol `Literal` (default: `sym.times`)

The symbol to use when `product-mode` is `"symbol"`.

5 · 100 · 2

```
#num-product(5, 100, 2, product-symbol: sym.dot.c)
```

product-phrase `Literal` (default: `[ by ]`)

The phrase to use when `product-mode` is `"phrase"`.

5 BY 100 BY 2

```
#num-product(5, 100, 2, product-symbol: [ BY ])
```

range-open-phrase `Literal` (default: `none`)

The phrase to open ranges with.

10 to 12
from 5 to 100

```
#num-range(10, 12)\
#num-range(5, 100, range-open-phrase: "from ")
```

range-phrase `Literal` (default: `[ to ]`)

The word or symbol to be inserted between the two entries of the range.

5 to 100
5–100

```
#num-range(5, 100)\
#num-range(5, 100, range-phrase: sym.dash)\
```

list-exponents
product-exponents `Choice` (default: `"individual"`)
range-exponents

Controls how lists, products and ranges can be “compressed” by combining the exponent parts.

individual Leaves the exponent with the matching value.

$5 \times 10^3, 7 \times 10^3, 9 \times 10^3$ and 1×10^4
$5 \times 10^3 \times 7 \times 10^3 \times 9 \times 10^3 \times 1 \times 10^4$
5×10^3 to 7×10^3

```
#num-list("5e3", "7e3", "9e3", "1e4")\
#num-product("5e3", "7e3", "9e3", "1e4")\
#num-range("5e3", "7e3")
```

combine The first exponent entry is taken and applied to all other entries, with the exponent itself placed at the end.

$5, 7, 9$ and 10×10^3
$5 \times 7 \times 9 \times 10 \times 10^3$
5 to 7×10^3

```
#metro-setup(
  list-exponents: "combine",
  product-exponents: "combine",
  range-exponents: "combine",
)
#num-list("5e3", "7e3", "9e3", "1e4")\
#num-product("5e3", "7e3", "9e3", "1e4")\
#num-range("5e3", "7e3")
```

combine-bracket Like `"combine"` but the list, product or range is wrapped in brackets, with the exponent outside.

$(5, 7, 9 \text{ and } 10) \times 10^3$
$(5 \times 7 \times 9 \times 10) \times 10^3$
$(5 \text{ to } 7) \times 10^3$

```
#metro-setup(
  list-exponents: "combine-bracket",
  product-exponents: "combine-bracket",
  range-exponents: "combine-bracket",
)
#num-list("5e3", "7e3", "9e3", "1e4")\
#num-product("5e3", "7e3", "9e3", "1e4")\
#num-range("5e3", "7e3")
```

list-units
product-units `Choice` (default: `"repeat"`)
range-units

Determines how `qty-list`, `qty-product` and `qty-range` functions print units.

repeat Each number will be printed with a unit.

2 T, 4 T, 6 T and 8 T
2 m × 4 m
2 ℃ to 4 ℃

```
#qty-list(2, 4, 6, 8, tesla)\
#qty-product(2, 4, metre)\
#qty-range(2, 4, degreeCelsius)
```

single The unit will only be placed at the end of the collection.

2, 4, 6 and 8 T
2 × 4 m
2 to 4 ℃

```
#metro-setup(
  list-units: "single",
  product-units: "single",
  range-units: "single",
)
#qty-list(2, 4, 6, 8, tesla)\
#qty-product(2, 4, metre)\
#qty-range(2, 4, degreeCelsius)
```

bracket Like `"single"` except brackets are placed around the collection.

(2, 4, 6 and 8) T
(2 × 4) m
(2 to 4) ℃

```
#metro-setup(
  list-units: "bracket",
  product-units: "bracket",
  range-units: "bracket",
)
#qty-list(2, 4, 6, 8, tesla)\
#qty-product(2, 4, metre)\
#qty-range(2, 4, degreeCelsius)
```

list-open-bracket
product-open-bracket `Literal` (default: `sym.paren.l`)
range-open-bracket

The opening bracket to be used when the collection is placed in brackets.

list-close-bracket
product-close-bracket `Literal` (default: `sym.paren.r`)
range-close-bracket

The closing bracket to be used when the collection is placed in brackets.

2.6 Complex Numbers

```
#complex(real, imag, ..unit-options)
```

Typesets the complex number, the first positional argument will be the real component and the second will be the coefficient of the imaginary component. If the second argument is either of the angle type or ends in “deg” or “rad”, the complex number will be considered to be in polar form and the first argument will be the radius. A unit can be optionally given as the third positional argument.

Note that when giving the angle as an angle type in radains, it will be output in degrees by default. This is due to angle types being unit agnostic. This behaviour can be changed with the `complex-angle-unit` option.

2.6.1 Options

complex-mode `Choice` (default: `"input"`)

The format in which complex values are printed.

input The complex value is printed as-given.

$1 + \mathrm{i}$

$1\angle 45°$

```
#complex(1, 1)\
#complex(1, 45deg)\
```

cartesian The output will be formatted in Cartesian form.

$1 + \mathrm{i}$

$0.71 + 0.71\mathrm{i}$

```
#metro-setup(complex-mode: "cartesian")
#complex(1, 1)\
#complex(1, 45deg, round-mode: "places")\
```

polar The output will be formatted in polar form.

$1.41\angle 45°$

$1\angle 45°$

```
#metro-setup(complex-mode: "polar")
#complex(1, 1, round-mode: "places", round-pad: false)\
#complex(1, 45deg)\
```

output-complex-root `Literal` (default: `math.upright("i")`)

The output complex root symbol.

$1 + 2\mathrm{i}$

$1 + 2\mathrm{j}$

```
#complex(1, 2, output-complex-root: "i")\
#complex(1, 2, output-complex-root: "j")\
```

complex-root-position `Choice` (default: `"after-number"`)

The position of the complex root can be adjusted to place it either before or after the associated numeral in a complex number by using this option.

$67 - 0.9\mathrm{i}$

$67 - \mathrm{i}0.9$

```
#complex(67, -0.9)\
#complex(67, -0.9, complex-root-position: "before-number")\
```

complex-angle-unit `Choice` (default: `"degrees"`)

The output unit of the angle component of a complex number in polar form.

$1\angle 57.295\,779\,513\,082\,32°\ \Omega$

$1\angle 1\ \Omega$

```
#complex(1, 1rad, ohm)\
#complex(1, 1rad, complex-angle-unit: "radians", ohm)
```

complex-symbol-angle `Literal` (default: `sym.angle`)

The symbol used to denote the angle of a complex number in polar form.

$1\mathrm{A}1°\ \Omega$

```
#complex(1, 1deg, ohm, complex-symbol-angle: math.upright("A"))
```

complex-symbol-degree `Literal` (default: `sym.degree`)

The symbol use for the units of degrees of a complex number in polar form.

$1\angle 1\mathrm{d}\,\Omega$

```
#complex(1, 1deg, ohm, complex-symbol-degree: math.upright("d"))
```

print-complex-unity `Switch` (default: `false`)

When the complex part of a number is exactly 1, it is possible to either print or suppress the value.

$\mathrm{i}\,\Omega$

```
#complex(0, 1, ohm)\
```

$1\mathrm{i}\,\Omega$

```
#complex(0, 1, ohm, print-complex-unity: true)\
```

2.7 Angles

```
#ang(..ang-options)
```

Typsets angles. The angle can be given as a single decimal number or 2 to 3 positional arguments of degrees, minutes and second, which is called the “arc format” in this document.

10°
12.3°
4.5°
1°2′3″
0°0′1″
10°0′0″
0°1′

```
#ang(10)\
#ang(12.3)\
#ang("4,5")\
#ang(1, 2, 3)\
#ang(0, 0, 1)\
#ang(10, 0, 0)\
#ang(0, 1)\
```

2.7.1 Options

angle-mode `Choice` (default: `"input"`)

The format in which angles are printed.

input The angle is printed as given.

2.67°
2°3′4″

```
#ang(2.67)\
#ang(2, 3, 4)\
```

arc The output will be formatted as an arc (degrees/minutes/seconds).

2°40′12″
2°3′4″

```
#metro-setup(angle-mode: "arc")
#ang(2.67)\
#ang(2,3,4)
```

decimal The output will be formatted as a decimal value.

2.67°
2.051 111 111 111 111°

```
#metro-setup(angle-mode: "decimal")
#ang(2.67)\
#ang(2,3,4)
```

number-angle-product `Literal` (default: `none`)

The separator between the number and angle symbol. This is independent of the related `quantity-product` option used by the `qty` function.

2.67°
2.67 °

```
#ang(2.67)\
#ang(2.67, number-angle-product: sym.space)
```

angle-separator `Literal` (default: `none`)

The separation of the different parts of an angle when printed in arc format.

6°7′6.5″
6° 7′ 6.5″

```
#ang(6, 7, 6.5)\
#ang(6, 7, 6.5, angle-separator: sym.space)
```

angle-symbol-degree `Literal` (default: `sym.degree`)

The symbol to use for the degree unit of an arc angle.

angle-symbol-minute `Literal` (default: `units.arcminute`)

The symbol to use for the minute unit of an arc angle.

angle-symbol-second `Literal` (default: `sym.arcsecond`)

The symbol to use for the second unit of an arc angle.

6d7m6.5s

```
#metro-setup(
  angle-symbol-degree: math.upright("d"),
  angle-symbol-minute: math.upright("m"),
  angle-symbol-second: math.upright("s"),
)
#ang(6, 7, 6.5)
```

3 Meet the Units

The following tables show the currently supported prefixes, units and their abbreviations. Note that unit abbreviations that have single letter commands are not available for import for use in math. This is because math mode already accepts single letter variables.

Unit	Command	Symbol
ampere	`ampere`	A
candela	`candela`	cd
kelvin	`kelvin`	K
kilogram	`kilogram`	kg
metre	`metre`	m
mole	`mole`	mol
second	`second`	s

Table 1: SI base units.

Unit	Command	Symbol	Unit	Command	Symbol
becquerel	`becquerel`	Bq	newton	`newton`	N
degree Celsius	`degreeCelsius`	℃	ohm	`ohm`	Ω
coulomb	`coulomb`	C	pascal	`pascal`	Pa
farad	`farad`	F	radian	`radian`	rad
gray	`gray`	Gy	siemens	`siemens`	S
hertz	`hertz`	Hz	sievert	`sievert`	Sv
henry	`henry`	H	steradian	`steradian`	sr
joule	`joule`	J	tesla	`tesla`	T
lumen	`lumen`	lm	volt	`volt`	V
katal	`katal`	kat	watt	`watt`	W
lux	`lux`	lx	weber	`weber`	Wb

Table 2: Coherent derived units in the SI with special names and symbols.

Unit	Command	Symbol
astronomicalunit	`astronomicalunit`	au
bel	`bel`	B
dalton	`dalton`	Da
day	`day`	d
decibel	`decibel`	dB
degree	`degree`	°
electronvolt	`electronvolt`	eV
hectare	`hectare`	ha
hour	`hour`	h
litre	`litre`	L
	`liter`	L
minute (plane angle)	`arcminute`	′
minute (time)	`minute`	min
second (plane angle)	`arcsecond`	″
neper	`neper`	Np
tonne	`tonne`	t

Table 3: Non-SI units accepted for use with the International System of Units.

Unit	Command	Symbol
byte	`byte`	B

Table 4: Non-SI units.

Prefix	Command	Symbol	10^x	Prefix	Command	Symbol	10^x
quecto	`quecto`	q	−30	deca	`deca`	da	1
ronto	`ronto`	r	−27	hecto	`hecto`	h	2
yocto	`yocto`	y	−24	kilo	`kilo`	k	3
atto	`atto`	a	−21	mega	`mega`	M	6
zepto	`zepto`	z	−18	giga	`giga`	G	9
femto	`femto`	f	−15	tera	`tera`	T	12
pico	`pico`	p	−12	peta	`peta`	P	15
nano	`nano`	n	−9	exa	`exa`	E	18
micro	`micro`	µ	−6	zetta	`zetta`	Z	21
milli	`milli`	m	−3	yotta	`yotta`	Y	24
centi	`centi`	c	−2	ronna	`ronna`	R	27
deci	`deci`	d	−1	quetta	`quetta`	Q	30

Table 5: SI prefixes

Prefix	Command	Symbol	2^x
kibi	`kibi`	Ki	10
mebi	`mebi`	Mi	20
gibi	`gibi`	Gi	30
tebi	`tebi`	Ti	40
pebi	`pebi`	Pi	50
exbi	`exbi`	Ei	60
zebi	`zebi`	Zi	70
yobi	`yobi`	Yi	80

Table 6: Binary prefixes

Unit	Abbreviation	Symbol
femtogram	fg	fg
picogram	pg	pg
nanogram	ng	ng
microgram	ug	µg
milligram	mg	mg
gram	g	g
kilogram	kg	kg
picometre	pm	pm
nanometre	nm	nm
micrometre	um	µm
millimetre	mm	mm
centimetre	cm	cm
decimetre	dm	dm
metre	m	m
kilometre	km	km
attosecond	as	as
femtosecond	fs	fs
picosecond	ps	ps
nanosecond	ns	ns
microsecond	us	µs
millisecond	ms	ms
second	s	s
femtomole	fmol	fmol
picomole	pmol	pmol
nanomole	nmol	nmol
micromole	umol	µmol
millimole	mmol	mmol
mole	mol	mol
kilomole	kmol	kmol
picoampere	pA	pA
nanoampere	nA	nA
microampere	uA	µA
milliampere	mA	mA
ampere	A	A
kiloampere	kA	kA
microlitre	uL	µL
millilitre	mL	mL
litre	L	L
hectolitre	hL	hL
millihertz	mHz	mHz
hertz	Hz	Hz
kilohertz	kHz	kHz
megahertz	MHz	MHz
gigahertz	GHz	GHz
terahertz	THz	THz
millinewton	mN	mN
newton	N	N
kilonewton	kN	kN
meganewton	MN	MN
pascal	Pa	Pa
kilopascal	kPa	kPa
megapascal	MPa	MPa
gigapascal	GPa	GPa
milliohm	mohm	mΩ
kilohm	kohm	kΩ
megohm	Mohm	MΩ
picovolt	pV	pV
nanovolt	nV	nV
microvolt	uV	µV
millivolt	mV	mV
volt	V	V
kilovolt	kV	kV
watt	W	W
nanowatt	nW	nW
microwatt	uW	µW
milliwatt	mW	mW
kilowatt	kW	kW
megawatt	MW	MW
gigawatt	GW	GW
joule	J	J
microjoule	uJ	uJ
millijoule	mJ	mJ
kilojoule	kJ	kJ
electronvolt	eV	eV
millielectronvolt	meV	meV
kiloelectronvolt	keV	keV
megaelectronvolt	MeV	MeV
gigaelectronvolt	GeV	GeV
teraelectronvolt	TeV	TeV
kilowatt hour	kWh	kWh
farad	F	F
femtofarad	fF	fF
picofarad	pF	pF
nanofarad	nF	nF
microfarad	uF	µF
millifarad	mF	mF
henry	H	H
femtohenry	fH	fH
picohenry	pH	pH
nanohenry	nH	nH
millihenry	mH	mH
microhenry	uH	µH
coulomb	C	C
nanocoulomb	nC	nC
millicoulomb	mC	mC
microcoulomb	uC	µC
kelvin	K	K
decibel	dB	dB
astrnomicalunit	au	au
becquerel	Bq	Bq
candela	cd	cd
dalton	Da	Da
gray	Gy	Gy
hectare	ha	ha
katal	kat	kat
lumen	lm	lm
neper	Np	Np
radian	rad	rad
sievert	Sv	Sv
steradian	sr	sr
weber	Wb	Wb
kilobyte	kB	kB
megabyte	MB	MB
gigabyte	GB	GB
terabyte	TB	TB
petabyte	PB	PB
exabyte	EB	EB
kibibyte	KiB	KiB
mebibyte	MiB	MiB
gibibyte	GiB	GiB
tebibyte	TiB	TiB
pebibyte	PiB	PiB
exbibyte	EiB	EiB

Table 7: Unit abbreviations

4 Creating

The following functions can be used to define custom units, prefixes, powers and qualifiers that can be used with the `unit` function.

4.1 Units

```
#declare-unit(unit, symbol, ..options)
```

Declare's a custom unit to be used with the `unit` and `qty` functions.

unit `string`

The string to use to identify the unit for string input.

symbol `Literal`

The unit's symbol. A string or math content can be used. When using math content it is recommended to pass it through `unit` first.

```
#let inch = "in"
#declare-unit("inch", inch)
#unit("inch / s")\
#unit($inch / s$)
```

in s^{-1}

in s^{-1}

4.2 Prefixes

```
#create-prefix(symbol)
```

Use this function to correctly create the symbol for a prefix. Metro uses Typst's `math.class` function with the `class` parameter `"unary"` to designate a prefix. This function does it for you.

symbol `Literal`

The prefix's symbol. A string or math content can be used. When using math content it is recommended to pass it through `unit` first.

```
#declare-prefix(prefix, symbol, power-tens)
```

Declare's a custom prefix to be used with the `unit` and `qty` functions.

prefix `string`

The string to use to identify the prefix for string input.

symbol `Literal`

The prefix's symbol. This should be the output of the `create-prefix` function specified above.

power-tens `Number`

The power ten of the prefix.

```
#let myria = create-prefix("my")
#declare-prefix("myria", myria, 4)
#unit("myria meter")\
#unit($myria meter$)
```

mym

mym

4.3 Powers

```
#declare-power(before, after, power)
```

This function adds two symbols for string input, one for use before a unit, the second for use after a unit, both of which are equivalent to the `power`.

before `string`

The string that specifies this power before a unit.

after `string`

The string that specifies this power after a unit.

power `Number`

The power.

```
#declare-power("quartic", "tothefourth", 4)
#unit("kilogram tothefourth")\
#unit("quartic metre")
```

kg^{4}

m^{4}

4.4 Qualifiers

```
#declare-qualifier(qualifier, symbol)
```

This function defines a custom qualifier for string input.

qualifier `string`

The string that specifies this qualifier.

symbol `Literal`

The qualifier's symbol. Can be string or content.

```
#declare-qualifier("polymer", "pol")
#declare-qualifier("catalyst", "cat")
#unit("gram polymer per mole catalyst per hour")
```

$\mathrm{g}_{\mathrm{pol}}\ \mathrm{mol}_{\mathrm{cat}}^{-1}\ \mathrm{h}^{-1}$