# Metro

fenjalien and Mc-Zen
https://github.com/fenjalien/typst-units
Version 0.1.0

## 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

## 2.1 Options

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

Options for Metro's can be modified by using the `metro-setup` function. It takes an argument sink and saves any named parameters found. The options for each function are specified in their respective sections.

All options and function parameters 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`** Takes a float or integer.

## 2.2 Numbers

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

Formats a number. Exponents and uncertainties can be given using the named parameters. All parameters listed can be given as a string, content or a number. Note that number parsing is very limited and fragile at the moment.

Also note that explicitly written parts of a number when using a number type will be lost as Typst automatically parses them.

**number** `Literal`

The number to format.

**pm** `Literal` (default: `none`)

Uncertainty.

**e** `Literal` (default: `none`)

Exponent.

| | |
|---|---|
| 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]` |
| $3.45 \times 10^{-4}$ | `#num(e: -4)[3.45]` |
| $-10^{10}$ | `#num("-1", e: 10, print-unity-mantissa: false)` |

### 2.2.1 Options

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

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

Allows a leading plus sign to be printed.

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

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

Allows a negative sign on an entirely zero value.

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

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

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

| | |
|---|---|
| 0.123 | `#num(0.123)` |
| 0.123 | `#num(0.123, minimum-decimal-digits: 2)` |
| 0.1230 | `#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 | `#num(123)` |
| 123 | `#num(123, minimum-integer-digits: 2)` |
| 0123 | `#num(123, minimum-integer-digits: 4)` |

**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.

| | |
|---|---|
| $1 \times 2^{2}$ | `#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.

| | |
|---|---|
| $1 \times 10^{2}$ | `#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.

$2 \times 10^3$ `#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.

$1 \times 10^4$ `#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]`

$444 \times 10^0$ `#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 `#num[123.00]`

123.— `#metro-setup(zero-decimal-as-symbol: true)`

123.[—] `#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 math Typst accepts single characters but multiple characters together are expected to be variables. So Metro defines units and prefixes which you can import to be use.

```
#import "@preview/metro:0.1.0": 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: *
$unit(joule / mole / kelvin)$
```

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

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")
```

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

`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")
```

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

$\mathrm{Bq^{-2}}$
$\mathrm{Bq^{-2}}$

`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")
```

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

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")
```

$\mathrm{H^{5}}$
$\mathrm{H^{5}}$
$\mathrm{H^{5}}$

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

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)")
```

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

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

### 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.

$\mathrm{F^2\,lm\,cd}$ `#unit("farad squared lumen candela")`
$\mathrm{F^2 \cdot lm \cdot cd}$ `#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

$\mathrm{J\,mol^{-1}\,K^{-1}}$ `#unit("joule per mole per kelvin")`
$\mathrm{m\,s^{-2}}$ `#unit("metre per second squared")`

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

$\frac{\mathrm{J}}{\mathrm{mol\,K}}$ `#unit("joule per mole per kelvin", per-mode: "fraction")`
$\frac{\mathrm{m}}{\mathrm{s^2}}$ `#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.

$\mathrm{J/(mol\,K)}$ `#metro-setup(per-mode: "symbol")`
$\mathrm{m/s^2}$ `#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)
```
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.

| | |
|---|---|
| $\mathrm{Pa\,Gy^{-1}\,H}$ | `#unit("pascal per gray henry")` |
| $\mathrm{Pa\,Gy^{-1}\,H^{-1}}$ | `#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")
```
$\mathrm{kg(pol)^2\,mol(cat)^{-1}\,h^{-1}}$

**bracket**

```
#unit("kilogram of(pol) squared per mole of(cat) per hour", qualifier-mode:
"bracket")
```
$\mathrm{kg(pol)^2\,mol(cat)^{-1}\,h^{-1}}$

**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")
```
$\mathrm{kg\ pol^2\ mol\ cat^{-1}\ h^{-1}}$
$\mathrm{kg\ of\ pol^2\ mol\ of\ cat^{-1}\ h^{-1}}$

**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.

| | |
|---|---|
| $\mathrm{Hz^{0.5}}$ | `#unit("Hz tothe(0.5)")` |
| $\sqrt{\mathrm{Hz}}$ | `#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: 4.5cm)[Some filler text #qty(10, "m")]
#metro-setup(allow-quantity-breaks: true)
#box(width: 4.5cm)[Some filler text #qty(10, "m")]
```

Some filler text
10 m
Some filler text 10
m

# 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.

| Prefix | Command | Symbol | Power | Prefix | Command | Symbol | Power |
|---|---|---|---|---|---|---|---|
| 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 4: SI 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 |

| Unit | Abbreviation | Symbol |
|---|---|---|
| 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 |

| Unit | Abbreviation | Symbol |
|---|---|---|
| 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 |

Table 5: Unit abbreviations

# 4 Creating

The following functions can be used to define cutom 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($inch / s$)
```

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

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

## 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($myria meter$)
```

$\mathrm{mym}$

$\mathrm{mym}$

## 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")
```
$\mathrm{kg}^{4}$
$\mathrm{m}^{4}$

## 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")
```
$\mathrm{g}_{\mathrm{pol}}\,\mathrm{mol}_{\mathrm{cat}}^{-1}\,\mathrm{h}^{-1}$