# Eqalc

Convert math equations to functions.

v0.1.0 Tijme MIT

## About

I created this package, because I thought it was very annoying to have to write down the equation in both math notation and code. This package allows you to write down the equation in math notation and convert it to a function.

## Usage

You can use this package in two ways:

- You can set the equation equal to a variable, and then use it in one of the functions like this:

```
#let eq = $ y = 2x + 3 $
#eq // To show the equation
#math-to-table(eq)
```

- You can use a label to access the equation like this:

```
$ y = 2x + 3 $ <eq>
#context math-to-table(<eq>)
```

> **Warning**
> Beware the you must use a unique label for each equation. And that you use a `#context` block to use a label.

- math-to-code()
- math-to-data()
- math-to-func()
- math-to-table()

### math-to-code

Converts a math expression to code.

Example: `#math-to-code($x^2$)` will output `calc.pow(x,2)`.

#### Parameters

`math-to-code(math: content label) -> content`

**math** `content` or `label`

The math expression.

### math-to-data

Math to any data you might need.

Example: `#math-to-data($f(x)=x^2$)` will output:

```
#(
  func: (x => calc.pow(x,2)),
  str: "calc.pow(x,2)",
  x: "x",
  x-math: $x$,
  fx: "f(x)",
  fx-math: $f(x)$,
)
```

**Parameters**

`math-to-data(math: content label) -> (func: function, str: string, x: string, x-math: content, fx: string, fx-math: content)`

**math** `content` or `label`

The math expression.

### math-to-func

Creates a function from a math expression.

Example: `#math-to-func($x^2$)` will output `#(x => calc.pow(x,2))`.

**Parameters**

`math-to-func(math: content label) -> function`

**math** `content` or `label`

The math expression.

### math-to-table

Creates a table of function values.

Example: `#math-to-table($x^2$, min: 1, max: 5, step: 1)` will output:

```
x    | 1 | 2 | 3 | 4 | 5 |
-------------------------
f(x) | 1 | 4 | 9 | 16| 25|
```

But in an actual table.

**Parameters**

```
math-to-table(
  math: content label,
  min: integer,
  max: integer,
  step: integer,
  round: integer,
  name: content
) -> content
```

**math** `content` or `label`

The function to evaluate.

**min** `integer`

The minimum value of the domain.

Default: `0`

**max** `integer`

The maximum value of the domain.

Default: `5`

**step** `integer`

The step size.

Default: `1`

**round** `integer`

The integer of decimal places to round to.

Default: `2`

**name** `content`

The name of the function. Defaults to the first part of the math expression.

Default: `none`

## Utility functions

- get-variable()
- label-to-math()
- math-to-str()

**get-variable**

Gets the main variable from a math expression.

**Parameters**

```
get-variable(math-str: string) -> string
```

**math-str** `string`

The math expression.

### label-to-math

Converts a label to a math expression.

**Parameters**

```
label-to-math(label: label) -> content
```

**label** `label`

A label representing a math expression.

### math-to-str

Converts math equations to strings.

**Parameters**

```
math-to-str(
  eq: content label,
  get-first-part: boolean,
  depth: integer
) -> string
```

**eq** `content` or `label`

The math expression.

**get-first-part** `boolean`

Get the part before the equals sign. This is used to get the function name.

Default: `false`

**depth** `integer`

The depth of the recursion. This is used for debugging.

Default: `0`