t4t

v0.2.0 MIT

A utility package for typst package authors

Jonas Neugebauer

https://github.com/jneug/typst-tools4typst

Table of contents

N.1. Module `is`

- neg()
- eq()
- neq()
- n()
- non()
- not-none()
- not-n()
- one-not-none()
- a()
- aut()
- not-auto()
- not-a()
- empty()
- not-empty()
- any()
- not-any()
- has()
- type()
- dict()
- arr()
- content()
- label()
- color()
- stroke()
- loc()
- bool()
- any-type()
- same-type()
- all-of-type()
- none-of-type()

`#neg(test)`

Creates a new test function, that is `true`, when `test` is `false`.

Can be used to create negations of tests like:

```
#let not-raw = is.neg(is.raw)
```

Argument
`test` `none`

test to negate.

`#eq(compare, value)`

Tests if values `compare` and `value` are equal.

Argument
`compare` `none`

first value

Argument
`value` `none`

second value

`#neq(compare, value)`

Tests if values `compare` and `value` are not equal.

Argument
`compare` `none`

first value

Argument
`value` `none`

second value

`#n(..values)`

Tests if any one of `values` is equal to `none`.

Argument
`..values` `none`

values to test

`#non()`

Alias for `n()`.

`#not-none(..values)`

Tests if none of `values` is equal to `none`.

Argument
`..values` `none`

values to test

`#not-n()`

Alias for `not-none()`

`#one-not-none(..values)`

Tests, if at least one value in `values` is not equal to `none`.

Useful for checking mutliple optoinal arguments for a valid value:

```
#if is.one-not-none(..args.pos()) [
  #args.pos().find(is.not-none)
]
```

Argument
`..values` `none`

values to test

#a(..values)

Tests if any one of `values` is equal to `auto`.

Argument

`..values` `none`

values to test

#aut()

Alias for `a()`

#not-auto(..values)

Tests if none of `values` is equal to `auto`.

Argument

`..values` `none`

values to test

#not-a()

Alias for `not-auto()`

#empty(value)

Tests, if `value` is *empty*.

A value is considered *empty* if it is an empty array, dictionary or string, or the value `none`.

Argument

`value` `none`

value to test

#not-empty(value)

Tests, if `value` is not *empty*.

See `empty()` for an explanation what *empty* means.

Argument

`value` `none`

value to test

`#any(..compare, value)`

Tests, if `value` is not *empty*.

See `empty()` for an explanation what *empty* means.

Argument

`value` `none`

value to test

`#not-any(..compare, value)`

Tests if `value` is not equals to any one of the other passed in values.

Argument

`..compare` `none`

values to compare to

Argument

`value` `none`

value to test

`#has(..keys, value)`

Tests if `value` contains all the passed `keys`.

Either as keys in a dictionary or elements in an array. If `value` is neither of those types, `false` is returned.

Argument

`..keys` `none`

keys or values to look for

Argument

`value` `none`

value to test

`#type(t, value)`

Tests if `value` is of type `t`.

Argument

`t` `none`

name of the type

Argument

`value` `none`

value to test

`#dict(value)`

Tests if `value` is of type dictionary.

Argument

`value` `none`

value to test

`#arr(value)`

Tests if `value` is of type array.

Argument

`value` `none`

value to test

`#content(value)`

Tests if `value` is of type content.

Argument

`value` `none`

value to test

`#label(value)`

Tests if `value` is of type label.

Argument

`value` `none`

value to test

`#color(value)`

Tests if `value` is of type color.

Argument

`value` `none`

value to test

#stroke(value)

Tests if `value` is of type stroke.

Argument

`value` none

value to test

#loc(value)

Tests if `value` is of type location.

Argument

`value` none

value to test

#bool(value)

Tests if `value` is of type boolean.

Argument

`value` none

value to test

#any-type(..types, value)

Tests if types `value` is any one of `types`.

Argument

`..types` none

type names to check against

Argument

`value` none

value to test

#same-type(..values)

Tests if all passed in values have the same type.

Argument

`..values` `none`

values to test

`#all-of-type(t, ..values)`

Tests if all of the passed in values have the type `t`.

Argument

`t` `none`

type to test against

Argument

`..values` `none`

values to test

`#none-of-type(t, ..values)`

Tests if none of the passed in values has the type `t`.

Argument

`t` `none`

type to test against

Argument

`..values` `none`

values to test

`#elem(func, value)`

Tests if `value` is a content element with `value.func() == func`.

If `func` is a string, `value` will be compared to `repr(value.func())`, instead. Both of these effectively do the same:

```
#is.elem(raw, some_content)
#is.elem("raw", some_content)
```

Argument

`func` `none`

element function

Argument

`value` none

value to test

`#sequence(value)`

Tests if `value` is a sequence of content.

N.2. Module `def`

- if-true()
- if-false()
- if-none()
- if-auto()
- if-any()
- if-not-any()
- if-empty()
- if-arg()
- as-arr()

`#if-true(test, default, do: none, value)`

Returns `default` if `test` is `true`, `value` otherwise.

If `test` is `false` and `do` is set to a function, `value` is passed to `do`, before being returned.

Argument

`test` none

a test result

Argument

`default` none

default value to return

Argument

`do: none` none

postprocessor for `value`

Argument

`value` none

the valu eto test

`#if-false(test, default, do: none, value)`

Returns `default` if `test` is `false`, `value` otherwise.

If `test` is `true` and `do` is set to a function, `value` is passed to `do`, before being returned.

Argument

`test` none

a test result

Argument

`default` `none`

default value to return

Argument

`do: none` `none`

postprocessor for `value`

Argument

`value` `none`

the valu eto test

```
#if-none(default, do: none, value)
```

Returns `default` if `value` is `none`, `value` otherwise.

If `value` is not `none` and `do` is set to a function, `value` is passed to `do`, before being returned.

Argument

`default` `none`

default value to return

Argument

`do: none` `none`

postprocessor for `value`

Argument

`value` `none`

the valu eto test

```
#if-auto(default, do: none, value)
```

Returns `default` if `value` is `auto`, `value` otherwise.

If `value` is not `auto` and `do` is set to a function, `value` is passed to `do`, before being returned.

Argument

`default` `none`

default value to return

Argument

`do: none` `none`

postprocessor for `value`

Argument

`value` `none`

the valu eto test

`#if-any(..compare, default, do: none, value)`

Returns `default` if `value` is equal to any value in `compare`, `value` otherwise.

```
#def.if-any(
  none, auto,     // ..compare
  1pt,            // default
  thickness       // value
)
```

If `value` is in `compare` and `do` is set to a function, `value` is passed to `do`, before being returned.

Argument

`..compare` `none`

list of values to compare `value` to

Argument

`default` `none`

default value to return

Argument

`do: none` `none`

postprocessor for `value`

Argument

`value` `none`

value to test

`#if-not-any(..compare, default, do: none, value)`

Returns `default` if `value` is not equal to any value in `compare`, `value` otherwise.

```
#def.if-not-any(
  left, right, top, bottom,   // ..compare
  left,                       // default
  position                    // value
)
```

If `value` is in `compare` and `do` is set to a function, `value` is passed to `do`, before being returned.

Argument

`..compare` `none`

list of values to compare `value` to

Argument

`default` `none`

default value to return

Argument

`do: none` `none`

postprocessor for `value`

Argument

`value` `none`

value to test

`#if-empty(default, do: none, value)`

Returns `default` if `value` is empty, `value` otherwise.

If `value` is not empty and `do` is set to a function, `value` is passed to `do`, before being returned.

Depends on `is.empty()`. See there for an explanation of *empty*.

Argument

`default` `none`

default value to return

Argument

`do: none` `none`

postprocessor for `value`

Argument

`value` `none`

value to test

`#if-arg(default, do: none, args, key)`

Returns `default` if `key` is not an existing key in `args.named()`, `args.named().at(key)` otherwise.

If `value` is not in `args` and `do` is set to a function, the value is passed to `do`, before being returned.

Argument

`default` `none`

default value to return

Argument

`do: none` none

postprocessor for `value`

Argument

`args` none

arguments to test

Argument

`key` none

key to look for

`#as-arr(..values)`

Always returns an array containing all `values`.

Any arrays in `values` will be flattened into the result. This is useful for arguments, that can have one element or an array of elements:

```
#def.as-arr(author).join(", ")
```

N.3. Module `alias`

N.4. Module `assert`

- that()
- that-not()
- eq()

`#that()`

Asserts that the passed `test` is `true`.

`#that-not(test, message: "")`

Asserts that the passed `test` is `false`.

`#eq()`

Asserts that the passed values are equal.

`#ne()`

Asserts that the passed values are not equal.

`#neq()`

Alias for `ne()`

N.5. Module `get`

- dict()
- dict-merge()
- args()
- text()
- stroke-paint()
- stroke-thickness()

`#dict(..dicts)`

Create a new dictionary from the passed `values`.

All named arguments are stored in the new dictionary as is. All positional arguments are grouped in key/value-pairs and inserted into the dictionary:

```
#get.dict("a", 1, "b", 2, "c", d:4, e:5)
   // gives (a:1, b:2, c:none, d:4, e:5)
```

`#dict-merge(..dicts)`

Recursivley merges the passed in dictionaries.

```
#get.dict-merge(
    (a: 1),
    (a: (one: 1, two:2)),
    (a: (two: 4, three:3))
)
   // gives (a:(one:1, two:4, three:3))
```

Based on work by @johannes-wolf for johannes-wolf/typst-canvas.

`#args(args, prefix: "")`

Creats a function to extract values from an argument sink `args`.

The resulting function takes any number of positional and named arguments and creates a dictionary with values from `args.named()`. Positional arguments to the function are present in the result, if they are present in `args.named()`. Named arguments are always present, either with their value from `args.named()` or with the provided value.

A `prefix` can be specified, to extract only specific arguments. The resulting dictionary will have all keys with the prefix removed, though.

```
#let my-func( ..options, title ) = block(
    ..get.args(options)(
        "spacing", "above", "below",
        width:100%
    )
)[
    #text(..get.args(options, prefix:"text-")(
        fill:black, size:0.8em
    ), title)
]
```

```
#my-func(
    width: 50%,
    text-fill: red, text-size: 1.2em
)[#lorem(5)]
```

`#text(element, sep: "")`

Recursively extracts the text content of a content element.

If present, all child elements are converted to text and joined with `sep`.

`#stroke-paint(stroke, default: rgb("#000000"))`

Returns the color of `stroke`. If no thickness information is available, `default` is used. **Deprecated since Typst 0.7.0**: use `stroke.thickness` instead.

Based on work by @PgBiel for PgBiel/typst-tablex.

`#stroke-thickness(stroke, default: 1pt)`

Returns the thickness of `stroke`. If no thickness information is available, `default` is used. **Deprecated since Typst 0.7.0**: use `stroke.thickness` instead.

N.6. Module `math`

Part I. Index