

Package ‘optional’

July 22, 2025

Type Package

Title Optional Types and Pattern Matching

Version 2.0.1

Date 2022-04-27

Author Antoine Champion

Maintainer Antoine Champion <antoine.champion@outlook.com>

Description Introduces optional types with `some()` and `none`, as well as `match_with()` from functional languages.

License BSL

Imports methods, magrittr

RoxygenNote 6.0.1

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2022-04-27 17:40:03 UTC

Contents

fallthrough	2
make_opt	2
match_with	3
none	4
option	5
opt_unwrap	6
some	7

Index	8
--------------	----------

fallthrough	<i>Fallthrough function</i>
-------------	-----------------------------

Description

Permit a pattern matching to continue even if its argument is executed.

Usage

```
fallthrough(fun)
```

Arguments

fun	A result function used in make_opt()
-----	--------------------------------------

Details

fallthrough(fun) can be applied to a result function fun inside a match_with() pattern. If there is a match, this will make the pattern matching continue through the other conditions at the end of the result function fun. match_with(variable,pattern, fallthrough(result-function),...

Examples

```
library(magrittr)

a <- 4
match_with(a,
  . %>% if (. %% 2 == 0).,
  fallthrough( function() "This number is even" ),
  . %>% if ( sqrt(.) == round(sqrt(.)) ).,
  function() "This number is a perfect square"
)
## [1] "This number is even" "This number is a perfect square"
```

make_opt	<i>Make optional</i>
----------	----------------------

Description

Make an existing function accepting and returning optionals.

Usage

```
make_opt(fun, stop_if_none = FALSE, fun_if_none = NULL)
```

Arguments

fun	The function to make optional, might be any function.
stop_if_none	If true, f_opt() will stop and return none if one of the arguments provided is none. Else, none will be sent as NULL to the function. *Default: FALSE*
fun_if_none	If not null, will be executed if an argument is none. *Default: NULL*

Details

1. Every optional argument passed to f_opt() will be converted to its original type before being sent to f(). If one or more of them is none, several behaviors are available (see argument list).
2. If f() returns null, or if an error is thrown during its execution, then f_opt() returns none. Else it will return option(f(...)).

Value

The optional function. To be used with the same parameters than fun().

See Also

option(), none(), match_with()

Examples

```
c_opt <- make_opt(c)
c_opt(option(2), none, option(5))
## [1] 2 5
c_opt()
## [1] "None"
```

match_with

Match With

Description

Function to check a variable using pattern matching.

Usage

```
match_with(x, ...)
```

Arguments

x	The variable to pattern-match
...	Pairs of one pattern (value or list or magrittr sequence) and one result function

Details

`match_with(variable, pattern, result-function, ...` If `variable` matches a `pattern`, `result-function` is called. For comparing optional types, it is a better habit to use `match_with` than a conditional statement.

- Each `pattern` can be either:
 - an object or a primitive type (direct comparison with `variable`),
 - a list (match if `variable` is in the list),
 - a `magrittr` functional sequence that matches if it returns `variable`. The dot `.` denotes the variable to be matched.
- If `result-function` takes no arguments, it will be called as `is`. Else, the only argument that will be sent is `variable`. You can also use the `fallthrough` function `fallthrough()` to permit the matching to continue even if the current pattern is matched.

See Also

`option()`, `none`

Examples

```
library(magrittr)

a <- 5
match_with(a,
  . %>% option(.),      paste,
  none, function() "Error!"
)
## [1] 5

match_with(a,
  1,                    function() "Matched exact value",
  list(2, 3, 4),        function(x) paste("Matched in list:", x),
  . %>% if (. > 4) .,    function(x) paste("Matched in condition:", x)
)
## [1] "Matched in condition: 5"
```

none

None

Description

Indicates an invalid variable. Might be returned by an optional function (see `?make_opt()`)

Usage

none

Format

An object of class `optional` of length 1.

See Also

`option()`, `opt_unwrap()`

Examples

```
a <- none
a
## [1] None
```

`option`

option

Description

Make a variable optional.

`option` is an object wrapper which indicates whether the object is valid or not.

Usage

```
option(arg)
```

Arguments

`arg` The variable to make optional

Details

Note that `option(option(i)) == option(i)` and `option(none) == FALSE`

Operators and `print` will have the same behavior with an optional than with its base type.

Value

`arg` as optional

See Also

`none`, `opt_unwrap()`, `make_opt()`

Examples

```
a <- option(5)
class(a)
## [1] "optional"

a == 5
## [1] TRUE

a
## [1] 5
```

opt_unwrap

Option Unwrap

Description

Cast an optional object to its base type.

Usage

```
opt_unwrap(opt)
```

Arguments

opt The optional variable to cast back

Details

Since an optional can be used the same way as its base type, there is no known scenario where this function might be useful.

Value

The object wrapped in opt. NULL if opt is none.

See Also

make_opt(), match_with()

Examples

```
a <- option(5)
class(a)
## [1] "optional"
a <- opt_unwrap(a)

class(a)
## [1] "numeric"
```

some	<i>some</i>
------	-------------

Description

Check if a optional object equals none

Usage

```
some(arg)
```

Arguments

arg The variable to check existence

Value

TRUE if arg is an optional variable and if it is not none, else returns FALSE

See Also

option(), none a <- option(1) some(a) ## [1] TRUE b <- none some(b) ## [1] FALSE

Index

* **datasets**

none, [4](#)

fallthrough, [2](#)

make_opt, [2](#)

match_with, [3](#)

none, [4](#)

opt_unwrap, [6](#)

option, [5](#)

some, [7](#)